Jupyter Tutorial

Release 0.8.0

Veit Schiele

Jul 29, 2021
# Contents

## 1 Introduction
1.1 Status ........................................... 3
1.2 Target group .................................... 3
1.3 Structure of the Jupyter tutorial .............. 3
1.4 Why Jupyter? .................................... 4
1.5 Jupyter infrastructure .......................... 4

## 2 First steps
2.1 Install Jupyter Notebook ....................... 5
2.2 Create notebook ................................ 7
2.3 Example ....................................... 10
2.4 Installation .................................. 13
2.5 Follow us .................................... 15
2.6 Pull-Requests ................................. 15

## 3 Workspace
3.1 IPython ...................................... 17
3.2 Jupyter ....................................... 50

## 4 Read, persist and provide data
4.1 Open data .................................... 141
4.2 Serialisation formats ......................... 142
4.3 Requests ..................................... 152
4.4 BeautifulSoup ................................. 157
4.5 Intake ....................................... 158
4.6 PostgreSQL .................................. 172
4.7 NoSQL databases .............................. 197
4.8 gRPC ........................................ 206
4.9 FastAPI ...................................... 213
4.10 Glossary ..................................... 219

## 5 Clean up and validate data
5.1 Deduplicate data ............................... 223
5.2 String matching ................................ 228
5.3 Managing missing data with pandas ........... 230
5.4 Scikit Learn preprocessing .................... 234
5.5 Dask pipeline ................................ 236
5.6 Data validation with voluptuous (schema definitions) ........................... 240
5.7 Pandas DataFrame validation with Engarde .......................... 246
5.8 Pandas DataFrame validation with Bulwark .......................... 248
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.9</td>
<td>TDDA: Test-Driven Data Analysis</td>
<td>250</td>
</tr>
<tr>
<td>5.10</td>
<td>Hypothesis: property based testing</td>
<td>254</td>
</tr>
<tr>
<td>6</td>
<td>Visualise data</td>
<td>257</td>
</tr>
<tr>
<td>7</td>
<td>Refactoring</td>
<td>259</td>
</tr>
<tr>
<td>7.1</td>
<td>Check and improve code quality and complexity</td>
<td>259</td>
</tr>
<tr>
<td>7.2</td>
<td>Performance measurement and optimisation</td>
<td>275</td>
</tr>
<tr>
<td>8</td>
<td>Create a product</td>
<td>319</td>
</tr>
<tr>
<td>8.1</td>
<td>Manage code with Git</td>
<td>319</td>
</tr>
<tr>
<td>8.2</td>
<td>Manage data with DVC</td>
<td>344</td>
</tr>
<tr>
<td>8.3</td>
<td>Create packages</td>
<td>353</td>
</tr>
<tr>
<td>8.4</td>
<td>Document</td>
<td>374</td>
</tr>
<tr>
<td>8.5</td>
<td>Licensing</td>
<td>385</td>
</tr>
<tr>
<td>8.6</td>
<td>Citing</td>
<td>388</td>
</tr>
<tr>
<td>8.7</td>
<td>Reproduce environments</td>
<td>399</td>
</tr>
<tr>
<td>8.8</td>
<td>Testing</td>
<td>434</td>
</tr>
<tr>
<td>8.9</td>
<td>Logging</td>
<td>449</td>
</tr>
<tr>
<td>9</td>
<td>Create web applications</td>
<td>453</td>
</tr>
<tr>
<td>9.1</td>
<td>Dashboards</td>
<td>453</td>
</tr>
<tr>
<td>10</td>
<td>Index</td>
<td>509</td>
</tr>
<tr>
<td>Index</td>
<td></td>
<td>511</td>
</tr>
</tbody>
</table>
Jupyter notebooks are growing in popularity with data scientists and have become the de facto standard for rapid prototyping and exploratory analysis. They inspire experiments and innovations enormously and as well they make the entire research process faster and more reliable. In addition, many additional components are created that expand the original limits of their use and enable new uses.
1.1 Status

The users of Jupyter notebooks are diverse, from data scientists to data engineers and analysts to system engineers. Their skills and workflows are very different. However, one of the great strengths of Jupyter notebooks is that they allow these different experts to work closely together in cross-functional teams.

- **Data scientists** conduct experiments with different coefficients and summarise the results.
- **Data engineers** check the quality of the code and make it more robust, efficient and scalable.
- **Data analysts** perform systematic studies of the data using code provided by data engineers.
- **System engineers** create the hub, the kernel, extensions, etc. and ensure that this infrastructure runs as smoothly as possible.

In this tutorial, we primarily address system engineers who want to build and operate a platform based on Jupyter notebooks. Then, we explain how this platform can be used effectively by data scientists, data engineers, and analysts.

1.2 Target group

The users of Jupyter notebooks are diverse, from data scientists to data engineers and analysts to system engineers. Their skills and workflows are very different. However, one of the great strengths of Jupyter notebooks is that they allow these different experts to work closely together in cross-functional teams.

1.3 Structure of the Jupyter tutorial

From Chapter 3, the Jupyter tutorial follows the prototype of a research project:

3. **Set up the workspace** with the installation and configuration of **IPython, Jupyter with nbextensions and ipywidgets**.
4. **Collect data**, either through a **REST API** or directly from an **HTML page**.
5. **Cleaning up data** is a recurring task that includes Remove or modify redundant, inconsistent, or incorrectly formatted data.
6. **Analyse data** through exploratory analysis and **visualising data**.
7. **Refactoring** includes parameterisation, validation and performance optimisation, including through *concurrency*.

8. **Creating a product** includes *Testing, Logging* and *Document* the methods and functions as well as *creating packages*.

9. **Web applications** can either generate dashboards from Jupyter notebooks or require more comprehensive application logic, such as demonstrated in *Bokeh-Plots in Flask einbinden*, or provide data via a *RESTful API*.

### 1.4 Why Jupyter?

How can these diverse tasks be simplified? You will hardly find a tool that covers all of these tasks, and several tools are often required even for individual tasks. Therefore, on a more abstract level, we are looking for more general patterns for tools and languages with which data can be analysed and visualised and a project can be documented and presented. This is exactly what we are aiming for with *Project Jupyter*.

The Jupyter project started in 2014 with the aim of creating a consistent set of open source tools for scientific research, reproducible workflows, *computational narratives* and data analysis. In 2017, Jupyter received the *ACM Software Systems Award* – a prestigious award which, among other things, shares with Unix and the web.

To understand why Jupyter notebooks are so successful, let’s take a closer look at the core functions:

**Jupyter Notebook Format** Jupyter Notebooks are an open, JSON-based document format with full records of the user’s sessions and the code they contain.

**Interactive Computing Protocol** The notebook communicates with the computing kernel via the *Interactive Computing Protocol*, an open network protocol based on JSON data via *ZMQ* and *WebSockets*.

**Kernels** Kernels are processes that execute interactive code in a specific programming language and return the output to the user.

### 1.5 Jupyter infrastructure

A platform for the above-mentioned use cases requires an extensive infrastructure that not only allows the provision of the kernel and the parameterisation, time control and parallelisation of notebooks, but also the uniform provision of resources.

This tutorial provides a platform that enables fast, flexible and comprehensive data analysis beyond Jupyter notebooks. At the moment, however, we are not yet going into how it can be expanded to include streaming pipelines and domain-driven data stores.

However, you can also create and run the examples in the Jupyter tutorial locally.
2.1 Install Jupyter Notebook

2.1.1 Install Pipenv

pipenv is a dependency manager for Python projects. It uses Pip to install Python packages, but it simplifies dependency management. Pip can be used to install Pipenv, but the --user flag should be used so that it is only available to that user. This is to prevent system-wide packets from being accidentally overwritten:

```
$ python3 -m pip install --user pipenv
  Downloading pipenv-2018.7.1-py3-none-any.whl (5.0MB): 5.0MB downloaded
Requirement already satisfied (use --upgrade to upgrade): virtualenv in /usr/lib/python3/
  → dist-packages (from pipenv)
Installing collected packages: pipenv, certifi, pip, setuptools, virtualenv-clone
...
Successfully installed pipenv certifi pip setuptools virtualenv-clone
Cleaning up...
```

Note: If Pipenv is not available in the shell after installation, the USER_BASE/bin directory may have to be specified in PATH.

Under Linux and MacOS, USER_BASE can be determined with:

```
$ python3 -m site --user-base
/home/veit/.local
```

Then the bin directory has to be appended and added to the PATH. Alternatively, PATH can be set permanently by changing ~/.profile or ~/.bash_profile, in my case:

```
export PATH=/home/veit/.local/bin:$PATH
```

- Under Windows, the directory can be determined with py -m site --user-site and then site-packages can be replaced by

  Scripts. This then results in, for example:

```
C:\Users\veit\AppData\Roaming\Python36\Scripts
```

In order to be permanently available, this path can be entered under ``PATH`` in the control panel.
Further information on user-specific installation can be found in User Installs.

2.1.2 Create a virtual environment with jupyter

Python virtual environments allow Python packages to be installed in an isolated location for a specific application, rather than installing them globally. So you have your own installation directories and do not share libraries with other virtual environments:

```bash
$ mkdir myproject
$ cd $HOME
$ cd myproject
$ pipenv install jupyter
Creating a virtualenv for this project...
... Virtualenv location: /home/veit/.local/share/virtualenvs/myproject-9TTuTZjx
Creating a Pipfile for this project...
Installing jupyter...
...
```

2.1.3 Start jupyter notebook

```bash
$ pipenv run jupyter notebook
... [I 12:46:53 NotebookApp] The Jupyter Notebook is running at:
[I 12:46:53 NotebookApp] Use Control-C to stop this server and shut down all kernels...
(C 12:46:53 NotebookApp]
```

To access the notebook, open this file in a browser:

```
file:///Users/veit/Library/Jupyter/runtime/nbserver-7372-open.html
```

Or copy and paste one of these URLs:

```
http://localhost:8888/?token=53abd45a3002329de77f66886e4ca02539d664c2f5e6072e
```

Your standard web browser will then open with this URL.

When the notebook opens in your browser, the notebook dashboard is displayed with a list of the notebooks, files and subdirectories in the directory in which the notebook server was started. In most cases you want to start a notebook server in your project directory.
2.2 Create notebook

After the notebook server has started, we can create our first notebook.

2.2.1 Create a notebook

In your standard browser you should see the notebook dashboard with the New menu on the right. All notebook kernels are listed in this menu, but initially probably only Python 3.

After you have selected New → Python 3, a new notebook Untitled.ipynb will be created and displayed in a new tab:

2.2.2 Renaming the notebook

Next you should rename this notebook by clicking on the title Untitled:
2.2.3 The notebook user interface

There are two important terms used to describe Jupyter Notebooks: cell and kernel:

**Notebook kernel** Computational engine that executes the code contained in a notebook.

**Notebook cell** Container for text to be displayed in a notebook or for code to be executed by the notebook’s kernel.

**Code** contains code to be executed in the kernel, and the output which is shown below.

In front of the code cells are brackets that indicate the order in which the code was executed.

- `In [ ]:` indicates that the code has not yet been executed.
- `In [*]:` indicates that the execution has not yet been completed.

**Warning:** The output of cells can be used in other cells later. Therefore, the result depends on the order. If you choose a different order than the one from top to bottom, you may get different results later when you e.g. select Cell → Run All.

**Markdown** contains text formatted with Markdown, which is interpreted as soon as Run is pressed.

2.2.4 What’s an ipynb file?

This file describes a notebook in JSON format. Each cell and its contents including pictures are listed there along with some metadata. You can have a look at them if you select the notebook in the dashboard and then click on edit. E.g. the JSON file for my-first-notebook.ipynb looks like this:

```json
{
    "cells": [  
        {
            "cell_type": "markdown",
            "metadata": {},
            "source": [  
                "# My first notebook"
            ]
        },
        {
            "cell_type": "code",
            "execution_count": 1,
            "metadata": {},
            "outputs": [  
                {  
                    "name": "stdout",
                    "output_type": "stream",
                    "text": [  
                        "Hello World!\n"
                    ]
                }
            ],
            "source": [  
                "print('Hello World!')"
            ]
        }
    ]
}
```

(continues on next page)
2.2.5 Save and checkpoints

When you click on Save and Checkpoint, your ipynb file will be saved. But what is the checkpoint all about?

Every time you create a new notebook, a file is also created, which usually automatically saves your changes every 120 seconds. This checkpoint is usually located in a hidden directory called .ipynb_checkpoints/. This checkpoint file therefore enables you to restore your unsaved data in the event of an unexpected problem. You can go back to one of the last checkpoints in File → Revert to Checkpoint.

2.2.6 Tips and tricks

1. Give the notebook a title (# My title) and a meaningful foreword to describe the content and purpose of the notebook.

2. Create headings and documentation in Markdown cells to structure your notebook and explain your workflow steps. It doesn’t matter whether you do this for your colleagues or for yourself in the future.

3. Use Table of Contents (2) from the List of extensions to create a table of contents.

4. Use the notebook extension setup.

5. Use snippets from the list of extensions to add more frequently used code blocks, e.g. typical import instructions, easy to insert.

2.2. Create notebook
2.3 Example

Usually the required libraries are imported first:

[1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
```

2.3.1 Download sample data

Then we download the sample data. We can use shell commands within iPython for this by prefixing !. The following command fetches a .csv file to your current working directory:

[2]:
```
```

<table>
<thead>
<tr>
<th>% Total</th>
<th>% Received</th>
<th>% Xferd</th>
<th>Average Speed</th>
<th>Time</th>
<th>Time</th>
<th>Time</th>
<th>Current Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>861k</td>
<td>100</td>
<td>861k</td>
<td>0</td>
<td>0</td>
<td>573k</td>
<td>0:00:01</td>
</tr>
</tbody>
</table>

2.3.2 Read in sample data

Then we read the csv data into pandas as a DataFrame:

[3]:
```
df = pd.read_csv('fortune500.csv')
```

2.3.3 Examine data

pandas.DataFrame.head returns the first rows for the object based on the position.

[4]:
```
df.head()
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Rank</th>
<th>Company</th>
<th>Revenue (in millions)</th>
<th>Profit (in millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>General Motors</td>
<td>9823.5</td>
<td>806</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Exxon Mobil</td>
<td>5661.4</td>
<td>584.8</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>U.S. Steel</td>
<td>3250.4</td>
<td>195.4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>General Electric</td>
<td>2959.1</td>
<td>212.6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>Esmark</td>
<td>2510.8</td>
<td>19.1</td>
</tr>
</tbody>
</table>

pandas.DataFrame.tail returns the last rows from the object based on the position.

[5]:
```
df.tail()
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Rank</th>
<th>Company</th>
<th>Revenue (in millions)</th>
<th>Profit (in millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25495</td>
<td>2005</td>
<td>496 Wm. Wrigley Jr.</td>
<td>3648.6</td>
<td></td>
</tr>
<tr>
<td>25496</td>
<td>2005</td>
<td>497 Peabody Energy</td>
<td>3631.6</td>
<td></td>
</tr>
<tr>
<td>25497</td>
<td>2005</td>
<td>498 Wendy's International</td>
<td>3630.4</td>
<td></td>
</tr>
<tr>
<td>25498</td>
<td>2005</td>
<td>499 Kindred Healthcare</td>
<td>3616.6</td>
<td></td>
</tr>
<tr>
<td>25499</td>
<td>2005</td>
<td>500 Cincinnati Financial</td>
<td>3614.0</td>
<td></td>
</tr>
</tbody>
</table>

Profit (in millions)

| 25495 | 493 |
| 25496 | 175.4 |
Renaming the columns

The columns can be renamed with pandas.DataFrame.columns. This will make it easier for us to refer to the columns later. Afterwards we check the change with df.head():

```python
[6]: df.columns = ['year', 'rank', 'company', 'revenue', 'profit']
df.head()
```

<table>
<thead>
<tr>
<th>year</th>
<th>rank</th>
<th>company</th>
<th>revenue</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>General Motors</td>
<td>9823.5</td>
<td>806</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Exxon Mobil</td>
<td>5661.4</td>
<td>584.8</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>U.S. Steel</td>
<td>3250.4</td>
<td>195.4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>General Electric</td>
<td>2959.1</td>
<td>212.6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>Esmark</td>
<td>2510.8</td>
<td>19.1</td>
</tr>
</tbody>
</table>

Check the number of records

The function len() returns the number of items of an object.

```python
[7]: len(df)
```

[7]: 25500

This corresponds to 500 lines per year from 1955 up to 2005.

Check the data types

The function pandas.DataFrame.dtypes() returns the data types in the DataFrame.

```python
[8]: df.dtypes
```

<table>
<thead>
<tr>
<th>year</th>
<th>rank</th>
<th>company</th>
<th>revenue</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>int64</td>
<td>int64</td>
<td>object</td>
<td>float64</td>
<td>object</td>
</tr>
</tbody>
</table>

dtype: object

The column profit should be of the data type float64; that is not the case here. This indicates that the column is likely to have some values that are not numbers. Let's check this with a regular expression:

```python
[9]: non_numeric_profits = df.profit.str.contains('[^0-9.-]')
df.loc[non_numeric_profits].head()
```

<table>
<thead>
<tr>
<th>year</th>
<th>rank</th>
<th>company</th>
<th>revenue</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>228</td>
<td>1955</td>
<td>Norton</td>
<td>135.0</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

(continues on next page)
We also check whether there are other non-numerical values besides `N.A.`:

```
[10]: set(df.profit[non_numeric_profits])
[10]: {'N.A.'}
```

Next, let's examine how many records are affected:

```
[11]: len(df.profit[non_numeric_profits])
[11]: 369
```

... and how they are distributed over the years:

```
[12]: bin_sizes, _, _ = plt.hist(df.year[non_numeric_profits], bins=range(1955, 2006))
```

Here we can see that the problematic data points will not exceed 24 in a year, and with 500 data points per year, removing these values would be less than 5% of the data for that year. Let's assume that it is acceptable to us that the problematic data can be removed:

```
[13]: df = df.loc[~non_numeric_profits]
df.profit = df.profit.apply(pd.to_numeric)
```

We should now have 25131 records:

```
[14]: len(df)
[14]: 25131
```

... and for the column `profit` the data type should now be `float64`:

```
[15]: df.dtypes
```
2.4 Installation

1. Download and unpack:

```bash
$ curl -O https://codeload.github.com/veit/jupyter-tutorial/zip/main
$ unzip main
Archive: main
... creating: jupyter-tutorial-main/
```

2. Edit the Pipfile in the unpacked archive and enter your current Python version in this section:

```python
[requires]
python_version = ""
```

1. Install Pipenv

Refer to Install Pipenv

2. Install Python packages:

```bash
$ cd jupyter-tutorial-main
$ pipenv install
Creating a virtualenv for this project...
... Installing dependencies from Pipfile.lock (fbb457)...
... $ pipenv run jupyter nbextension enable highlighter/highlighter
   Enabling notebook extension highlighter/highlighter...
   - Validating: OK
```

3. Install the Jupyter Notebook Extensions Javascript and CSS files:

```bash
$ pipenv run jupyter contrib nbextension install --user
jupyter contrib nbextension install --user
Installing jupyter_contrib_nbextensions nbextension files to jupyter data directory
... Successfully installed jupyter-contrib-core-0.3.3 jupyter-contrib-nbextensions-0.5.1
jupyter-highlight-selected-word-0.2.0 jupyter-latex-envs-1.4.6
jupyter-nbextensions-configurator-0.4.1
...
$ pipenv run jupyter nbextension enable latex_envs --user --py
Enabling notebook extension latex_envs/latex_envs...
   - Validating: OK
```
4. Create HTML documentation:

Note that pandoc has to be installed. On Debian/Ubuntu you can just run

```bash
$ sudo apt-get install pandoc
```

To create the HTML documentation run these commands:

```bash
$ python3 -m venv .
$ bin/python -m pip install --upgrade pip
$ bin/python -m pip install -r docs/constraints.txt
$ bin/sphinx-build -ab html docs/ docs/_build/
```

5. Create a PDF:

For the creation of a PDF file you need additional packages.
For Debian/Ubuntu you get them with the following command:

```bash
$ sudo apt-get install texlive-latex-recommended texlive-latex-extra texlive-fonts-recommended latexmk
```

or for macOS with:

```bash
$ brew cask install mactex
...
  mactex was successfully installed!
$ curl --remote-name https://www.tug.org/fonts/getnonfreefonts/install-getnonfreefonts
$ sudo texlua install-getnonfreefonts
...
mktexlsr: Updating /usr/local/texlive/2020/texmf-dist/ls-R...
mktexlsr: Done.
```

Then you can generate a PDF with:

```bash
$ cd docs/
$ pipenv run make latexpdf
...
The LaTeX files are in _build/latex.
Run 'make' in that directory to run these through (pdf)latex
...
```

You can find the PDF at docs/_build/latex/jupytertutorial.pdf.

6. Install vnd run ale to check spelling

You can install Vale with:

```bash
$ brew install vale
```

You can install the parser for Restructuredtext with:

```bash
$ brew install docutils
```

See also:
- Vale installation
• Vale formats

Now you can check the RestructuredText files with:

```
$ cd jupyter-tutorial
$ vale docs/
✓✓✓ 0 errors, 0 warnings and 0 suggestions in 201 files.
```

2.5 Follow us

• GitHub
• Twitter
• Mastodon

2.6 Pull-Requests

If you have suggestions for improvements and additions, I recommend that you create a Fork of my GitHub Repository and make your changes there. You are also welcome to make a pull request. If the changes contained therein are small and atomic, I’ll be happy to look at your suggestions.
Setting up the workspace includes installing and configuring *IPython* and *Jupyter* with *nbextensions* and *ipywidgets*.

### 3.1 IPython

*IPython*, or *Interactive Python*, was initially an advanced Python interpreter that has now grown into an extensive project designed to provide tools for the entire life cycle of research computing. Today, IPython is not only an interactive interface to Python, but also offers a number of useful syntactic additions for the language. In addition, IPython is closely related to the *Jupyter* project.

**See also:**
- Miki Tebeka - *IPython: The Productivity Booster*

#### 3.1.1 Start the IPython shell

You can easily start IPython in a console:

```
$ pipenv run ipython
```

```
Python 3.7.0 (default, Aug 22 2018, 15:22:29)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.6.0 -- An enhanced Interactive Python. Type '?' for help.
```

In [1]:

Alternatively, you can use IPython in a Jupyter notebook. To do this, start the notebook server first:

```
$ pipenv run jupyter notebook
```

```
[I 17:35:02.419 NotebookApp] Serving notebooks from local directory: /Users/veit/vsc/˓→jupyter-tutorial
[I 17:35:02.419 NotebookApp] The Jupyter Notebook is running at:
[I 17:35:02.427 NotebookApp] http://localhost:8888/? ˓→token=72209334c2e325a68115902a63bd064db436c0c84aeced7f
[I 17:35:02.428 NotebookApp] Use Control-C to stop this server and shut down all kernels␣ ˓→(twice to skip confirmation).
[C 17:35:02.497 NotebookApp]
```

The standard browser should then be opened with the specified URL. Often this is http://localhost:8888.

Now you can start a Python process in the browser by creating a new notebook.
3.1.2 IPython magic

IPython not only enables Python to be used interactively, but also extends the Python syntax with so-called *magic commands*, which are provided with the prefix `%`. They are designed to quickly and easily solve common data analysis problems. A distinction is made between two different types of *magic commands*:

- *line magics*, denoted by a single `%` prefix, that run on a single input line
- *cell magics* which are preceded by a double symbol `%%` and which are executed within a notebook cell.

**Execute external code: %run**

If you start developing larger code, you will likely be working in both IPython for interactive exploration and a text editor to save code that you want to reuse. With the `%run` magic you can execute this code directly in your IPython session.

Imagine you created a `myscript.py` file with the following content:

```python
def square(x):
    return x ** 2

for N in range(1, 4):
    print(N, "squared is", square(N))
```

[1]: %run myscript.py

```
1 squared is 1
2 squared is 4
3 squared is 9
```

Note that after running this script, all of the functions defined in it will be available for use in your IPython session:

[2]: square(4)

[2]: 16

There are several ways you can improve the way your code runs. As usual, you can display the documentation in IPython with `%run?`.

**Run timing code: %timeit**

Another example of a Magic function is `%timeit`, which automatically determines the execution time of the following one-line Python statement. So we can e.g. output the performance of a list comprehension with:

[3]: %timeit L = [n ** 2 for n in range(1000)]

```
3.17 ms ± 800 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

The advantage of `%timeit` is that short commands automatically run multiple runs to get more robust results. For multiline instructions, adding a second `%` character creates cell magic that can process multiple input lines. For example, here is the equivalent construction using a `for` loop:

[4]: %timeit

```python
L = []
for n in range(1000):
    L.append(n ** 2)
```
We can immediately see that the list comprehension is about 10% faster than its equivalent with a `for` loop. We then describe performance measurements and optimisations in more detail in *Profiling*.

**Execute code from other interpreters**

IPython has a `%script` script magic with which you can execute a cell in a subprocess of an interpreter on your system, e.g. `bash`, `ruby`, `perl`, `zsh`, `R` etc. This can also be its own script that expects input in `stdin`. To do this, simply pass a path or a shell command to the program that is specified in the `%script` line. The rest of the cell is executed by this script, capturing `stdout` or `err` from the subprocess and displaying it.

```
[1]: %script python2
    import sys
    print 'Python %s' % sys.version

    [GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)]

[2]: %script python3
    import sys
    print('Python: %s' % sys.version)

Python: 3.7.0 (default, Aug 22 2018, 15:22:29)
    [Clang 8.0.0 (clang-800.0.42.1)]

[3]: %ruby
    puts "Ruby #{RUBY_VERSION}"

Ruby 2.3.7

[4]: %bash
    echo "$BASH"

/bin/bash
```

You can capture `stdout` and `err` from these sub-processes in Python variables:

```
[5]: %bash --out output --err error
    echo "stdout"
    echo "stderr" >&2

[6]: print(error)
    print(output)

stderr
stdout
```
Configure standard script magic

The list of aliases for the script magic is configurable. By default, some common interpreters can be used if necessary. However, you can also specify your own interpreter in ipython_config.py:

```python
c.ScriptMagics.scripts = ['R', 'pypy', 'myprogram']
c.ScriptMagics.script_paths = {'myprogram': '/path/to/myprogram'}
```

Help functions: ?, %magic and %lsmagic

Like normal Python functions, the IPython magic functions have docstrings that can be easily accessed. E.g. to read the documentation of the %timeit magic, just type:

[5]: %timeit?

Documentation for other functions can be accessed in a similar manner. To access a general description of the %magic functions available, including some examples, you can type:

[6]: %magic

For a quick list of all available magic functions, type:

[7]: %lsmagic

Available line magics:
- %alias %alias_magic %autoawait %autocall %automagic %autosave %bookmark %cat %cd
  - %clear %colors %conda %config %connect_info %cp %debug %dhist %dirs %doctest_
  - %edit %ed %env %gui %hist %history %killbgscripts %mdir %ls %mkdir %lx %ls
  - %load %load_ext %loadpy %logoff %logon %logstart %logstate %logstop %ls
  - %lsmagic %lx %macro %magic %man %matplotlib %mdir %more %mv %notebook %page_
  - %pastebin %pdb %pdef %pdoc %pfile %pinfo %pinfo2 %pip %popd %pprint
  - %precision %prun %psearch %psource %pushd %pwd %pycat %pylab %qtconsole
  - %quickref %recall %rehashx %reload_ext %rep %rerun %reset %reset_selective %rm_
  - %rmdir %run %save %sc %set_env %store %sx %system %tb %time %timeit
  - %unalias %unload_ext %who %who_ls %whos %xdel %xmode

Available cell magics:
- %%! %%HTML %%SVG %%bash %%capture %%debug %%file %%html %%javascript %%js
  - %latex %markdown %perl %%prun %pypy %python %python2 %python3 %ruby %
  - %script %sh %svg %sx %system %time %timeit %writefile

Automagic is ON, % prefix IS NOT needed for line magics.

You can also simply define your own magic functions. For more information, see Defining custom magics.
3.1.3 Shell commands in IPython

The IPython Notebook allows simple UNIX/Linux commands to be executed in a single input cell. There are no limits but when using, please keep in mind that in contrast to a regular UNIX/Linux shell, start each shell command with a `!`, for example `!ls` for the command `ls` (see below for further explanations about the command). Furthermore, each shell command is executed in its own subshell. For this reason, the results of previous shell commands are not available to you.

To begin with, the command `!ls` lists the files in the current working directory. The output is shown below the input cell, and lists the single file `shell.ipynb`:

```
[1]: !ls
shell.ipynb
```

The command `!pwd` displays the path to working directory:

```
[2]: !pwd
/Users/veit/jupyter-tutorial/docs/basics/ipython
```

The command `!echo` outputs text given as parameter to the `echo` command. The example below demonstrates how to print `Hello world`:

```
[3]: !echo "Hello world!"
Hello world!
```

### Passing values to and from the shell

There is a clever way through which you can access the output of a UNIX/Linux command as a variable in Python. Assign the output of a UNIX/Linux command to a variable as follows:

```
[4]: contents = !ls
```

Here the Python variable `contents` has been assigned the output of the command `ls`. As a result, `contents` is a list, where each list element corresponds to a line in the output. With the `print` command you output the list contents:

```
[5]: print(contents)
['create-delete.ipynb', 'file-system.ipynb', 'grep-find.ipynb', 'index.rst', 'pipes-filters.ipynb', 'regex.ipynb', 'shell-variables.ipynb']
```

You will see the same result below when executing the `pwd` command. The current directory is stored in the variable `directory`:

```
[6]: directory = !pwd
```

```
[7]: print(directory)
['/Users/veit/jupyter-tutorial/docs/basics/ipython']
```
3.1.4 Unix shell

Any command on the command line will also work in Jupyter Notebooks if prefixed with !. The results can then interact with the Jupyter namespace, see Passing values to and from the shell.

Navigate through files and directories

First let us find out where we are by running a command called pwd:

[1]: !pwd
/Users/veit/cusy/trn/jupyter-tutorial/docs/workspace/ipython/unix-shell

Here, the response is the iPython chapter of the Jupyter tutorial in my home directory /Users/veit.

On Windows the home directory will look like C:\Documents and Settings\veit or C:\Users\veit and on Linux like /home/veit.

To see the contents of our directory, we can use ls:

[2]: !ls
create-delete.ipynb index.rst shell-variables.ipynb
file-system.ipynb pipes-filters.ipynb
grep-find.ipynb regex.ipynb

• a trailing / indicates a directory
• @ indicates a link
• * indicates an executable

Depending on your default options, the shell might also use colors to indicate whether an entry is a file or a directory.

ls options and arguments

[3]: !ls -F ../
dev/ debugging.ipynb  extensions.rst  magics.ipynb  start.rst
display.ipynb  importing.ipynb  mypackage/  unix-shell/
dvc.list index.rst  shell.ipynb
length.txt sorted-length.txt

ls is the command, with the option -F and the argument ../.

• Options either start with a single dash (-) or two dashes (--), and they change the behavior of a command.
• Arguments tell the command what to operate on.
• Options and arguments are sometimes also referred as parameters.
• Each part is separated by spaces.
• Also, capitalisation is important, e.g.
• ls -s will display the size of files and directories alongside the names,
• while ls -S will sort the files and directories by size.
Show all options and arguments

ls comes with a lot of other useful options. Using man you can print out the built-in manual page for the desired UNIX/Linux-command:

```bash
[6]: !man ls
```

LS(1) BSD General Commands Manual LS(1)

NAME
ls -- list directory contents

SYNOPSIS
ls [-ABCFGHLOPRSTUW@abcdefghiklmnopqrstuwx1%] [file ...]

DESCRIPTION
For each operand that names a file of a type other than directory, ls displays its name as well as any requested, associated information. For each operand that names a file of type directory, ls displays the names of files contained within that directory, as well as any requested, associated information.

If no operands are given, the contents of the current directory are displayed. If more than one operand is given, non-directory operands are displayed first; directory and non-directory operands are sorted separately and in lexicographical order.

The following options are available:

-@ Display extended attribute keys and sizes in long (-l) output.

-1 (The numeric digit ``one'') Force output to be one entry per line. This is the default when output is not to a terminal.

...
Illegal options

If you try to use an option that isn’t supported, the commands will usually print an error message, e.g. for:

```
[7]: !ls -z
ls: illegal option -- z
usage: ls [-@ABCFGHLOPRSTUWabcdefghiklmnopqrstuvwxyz] [file ...]
```

Hidden Files

With the `-a` option you can display all files:

```
[8]: !ls -a
.
..  file-system.ipynb  regex.ipynb
..  grep-find.ipynb  shell-variables.ipynb
.ipynb_checkpoints  index.rst
create-delete.ipynb  pipes-filters.ipynb
```

In addition to the hidden directories `..` and `.` you may also see a directory called `.ipynb_checkpoints`. This file usually contains snapshots of the Jupyter notebooks.

Show directory tree

The command `tree` lists contents of directories in a tree-like format.

```
[9]: !tree
.
  ├── create-delete.ipynb
  │     └── file-system.ipynb
  │        └── grep-find.ipynb
  │              └── index.rst
  │                  └── pipes-filters.ipynb
  │                      └── regex.ipynb
  │                               └── shell-variables.ipynb

0 directories, 7 files
```

Change directory

At first it may seem irritating to some that they cannot use `!cd` to change to another directory.

```
[10]: !pwd
/Users/veit/cusy/trn/jupyter-tutorial/docs/workspace/ipython/unix-shell

[11]: !cd ..

[12]: !pwd
/Users/veit/cusy/trn/jupyter-tutorial/docs/workspace/ipython/unix-shell
```
The reason for this is that Jupyter uses a temporary subshell. If you want to change to another directory permanently, you have to use the magic command `%cd`.

```plaintext
[13]: %cd ..
/Users/veit/cusy/trn/jupyter-tutorial/docs/workspace/ipython
```

```plaintext
[14]: !pwd
/Users/veit/cusy/trn/jupyter-tutorial/docs/workspace/ipython
```

With the `%automagic` function, these can also be used without the preceding `%` character:

```plaintext
[17]: %automagic
Automagic is ON, % prefix IS NOT needed for line magics.
```

```plaintext
[18]: cd ..
/Users/veit/cusy/trn/jupyter-tutorial/docs/workspace
```

### Absolute and relative Paths

```plaintext
[19]: cd .
/Users/veit/cusy/trn/jupyter-tutorial/docs/workspace
```

```plaintext
[20]: cd ../..
/Users/veit/cusy/trn/jupyter-tutorial
```

```plaintext
[21]: cd ..
/Users/veit/cusy/trn
```

```plaintext
[22]: cd /
/
```

```plaintext
[23]: cd
/Users/veit
```

```plaintext
[24]: cd ~
/Users/veit
```

```plaintext
[25]: cd /Users/veit
/Users/veit
```
Create, update and delete files and directories

Creates a new directory test and then checks this with ls:

[1]: `!mkdir tests`

[2]:

\[
\begin{array}{lll}
\text{create-delete.ipynb} & \text{importing.ipynb} & \text{shell-variables.ipynb} \\
\text{debugging.ipynb} & \text{index.rst} & \text{shell.ipynb} \\
\text{display.ipynb} & \text{length.txt} & \text{sorted-length.txt} \\
\text{dvc.list} & \text{magics.ipynb} & \text{start.rst} \\
\text{extensions.rst} & \text{mypackage} & \text{tests} \\
\text{file-system.ipynb} & \text{pipes-filters.ipynb} & \\
\text{grep-find.ipynb} & \text{regex.ipynb} & \\
\end{array}
\]

Then we create the file test_file.txt in this directory.

[3]: `!touch tests/test_file.txt`

[4]:

\[
\text{ls tests}
\]

\[
\text{test_file.txt}
\]

Now we change the suffix of the file:

[5]: `!mv tests/test_file.txt tests/test_file.py`

[6]:

\[
\text{ls tests}
\]

\[
\text{test_file.py}
\]

Now we make a copy of this file:

[7]: `!cp tests/test_file.py tests/test_file2.py`

[8]:

\[
\text{ls tests}
\]

\[
\text{test_file.py  test_file2.py}
\]

A directory with all the files it contains is also possible recursively with the \texttt{-r} option:

[9]: `!cp -r tests tests.bak`

[10]:

\[
\text{ls tests.bak}
\]

\[
\text{test_file.py  test_file2.py}
\]

Finally, we delete the directories tests and tests.bak again:

[11]: `!rm -r tests tests.bak`

[12]:

\[
\text{ls}
\]
**Transfering files**

```bash
wget
```

[13]: `!!wget https://dvc.org/deb/dvc.list`

```bash
--2021-05-14 20:35:44--  https://dvc.org/deb/dvc.list
Auflosen des Hostnamens dvc.org (dvc.org)... 2606:4700:3036::6815:51cd, 2606:4700:3033::...ac43:a44c, 172.67.164.76, ...
Verbindungsaufbau zu dvc.org (dvc.org)|2606:4700:3036::6815:51cd|:443 ... verbunden.
HTTP-Anforderung gesendet, auf Antwort wird gewartet ... 303 See Other
Platz: https://s3-us-east-2.amazonaws.com/dvc-s3-repo/deb/dvc.list [folgenden]
--2021-05-14 20:35:45--  https://s3-us-east-2.amazonaws.com/dvc-s3-repo/deb/dvc.list
Auflosen des Hostnamens s3-us-east-2.amazonaws.com (s3-us-east-2.amazonaws.com)|s3-us-east-2.amazonaws.com)... 52.
...219.97.209
Verbindungsaufbau zu s3-us-east-2.amazonaws.com (s3-us-east-2.amazonaws.com)|52.219.97.209|:443 ... verbunden.
HTTP-Anforderung gesendet, auf Antwort wird gewartet ... 200 OK
Lange: 51 [binary/octet-stream]
Wird in »dvc.list.3« gespeichert.

dvc.list.3  100%[===================>] 51 --.-KB/s in 0s
2021-05-14 20:35:45 (1,68 MB/s) - »dvc.list.3« gespeichert [51/51]
```

- `-r` recursively crawls other files and directories
- `-np` avoids crawling to parent directories
- `-D` targets only the following domain name
- `-nH` avoids creating a subdirectory for the websites content
- `-m` mirrors with time stamping, time stamping, infinite recursion depth, and preservation of FTP directory settings
- `-q` supresses the output to the screen

3.1. IPython
cURL

Alternatively, you can use cURL, which supports a much larger range of protocols.

![cURL command example]

```
!curl -o dvc.list https://dvc.org/deb/dvc.list
```

```
% Total  % Received  % Xferd  Average Speed Time Time Time Current
Dload  Upload  Total   Spent   Left  Speed
100 85 100 85 0 0 159 0 --:--:-- --:--:-- --:--:-- 159

{  
    "cells": [
        {  
            "cell_type": "markdown", "id": "coral-investigator", "metadata": {}, "source": [
                "# Pipes and filters"
            ]
        },
        {  
            "cell_type": "markdown", "id": "coral-investigator", "metadata": {}, "source": [
                "ls shows all files and directories at this point."
            ]
        },
        {  
            "cell_type": "code", "execution_count": 1, "id": "original-stuart", "metadata": {}, "outputs": [
                {  
                    "name": "stdout", "output_type": "stream", "text": [
                        "create-delete.ipynb importing.ipynb regex.ipynb", "debugging.ipynb index.rst shell-variables.ipynb", "display.ipynb length.txt shell.ipynb", "extensions.rst magics.ipynb sorted-length.txt", "file-system.ipynb u001b[34mmypackageu001b[m start.rst\", "grep-find.ipynb pipes-filters.ipynb"
                    ]
                }
            ], "source": [
                "ls"
            ]
        },
        {  
            "cell_type": "markdown", "id": "nearby-complement", "metadata": {}, "source": [
                "With *.rst we restrict the results to all files with the suffix .rst:"
            ]
        },
        {  
            "cell_type": "code", "execution_count": 2, "id": "analyzed-attraction", "metadata": {}, "outputs": [
                {  
                    "name": "stdout", "output_type": "stream", "text": [
                        "extensions.rst magics.ipynb sorted-length.txt start.rst"
                    ]
                }
            ], "source": [
                "ls\"
            ]
        }
    ]
}
```

28 Chapter 3. Workspace
We can also output only the number of lines, words and characters in these documents:

```
!ls *.rst
```

```python
\[
\text{!wc} \ast .rst
\]
```

Now we write the number of characters in the file `length.txt` and then output the text with `cat`:

```
!wc -m \ast .rst > length.txt
```

3.1. IPython
"!cat length.txt"
]
}, {
"cell_type": "markdown", "id": "colonial-definition", "metadata": {}, "source": [
"We can also have the files sorted by the number of characters:"
]
}, {
"cell_type": "code", "execution_count": 6, "id": "middle-million", "metadata": {}, "outputs": [
{ "name": "stdout", "output_type": "stream", "text": [
" 924 index.rstrn", " 1138 start.rstrn", " 2094 extensions.rstrn", " 4156 totalrn"
]
}
], "source": [
"!sort -n length.txt"
]
}, {
"cell_type": "code", "execution_count": 7, "id": "intensive-testament", "metadata": {}, "outputs": [], "source": [
"!sort -n length.txt > sorted-length.txt"
]
}, {
"cell_type": "markdown", "id": "about-consensus", "metadata": {}, "source": [
"We can also overwrite the existing file:"
]
}, {
"cell_type": "code", "execution_count": 8, "id": "collectible-poison", "metadata": {}, "outputs": [], "source": [
"!sort -n length.txt > length.txt"
]
}, {
"cell_type": "markdown", "id": "surrounded-tribune", "metadata": {}, "source": [
"If we only want to know the total number of characters, i.e. only output the last line, we can do this with tail:
"
]
}, {
"cell_type": "code", "execution_count": 9, "id": "absolute-steel", "metadata": {}, "outputs": [], "source": [

Chapter 3. Workspace
"!tail -n 1 length.txt"
]
}, {
  "cell_type": "markdown", "id": "turkish-collect", "metadata": {}, "source": [
  "> is used to overwrite a file while >> is used to append to a file."
]
}, {
  "cell_type": "code", "execution_count": 10, "id": "perfect-identifier", "metadata": {}, "outputs": [], "source": [
    "!echo amount of characters >> length.txt"
  ]
}, {
  "cell_type": "code", "execution_count": 11, "id": "waiting-tours", "metadata": {}, "outputs": [
    {
      "name": "stdout", "output_type": "stream", "text": [
        "amount of characters\n"
      ]
    }
  ], "source": [
    "!cat length.txt"
  ]
}, {
  "cell_type": "markdown", "id": "enhanced-rates", "metadata": {}, "source": [
  "### Pipe
  | n", "n", "You can connect commands with a pipe (|). In the following one-liner, we want to display the number of characters for the shortest file:"n"
  ]
}, {
  "cell_type": "code", "execution_count": 12, "id": "proprietary-center", "metadata": {}, "outputs": [
    {
      "name": "stdout", "output_type": "stream", "text": [
        " 28 index.rst"
      ]
    }
  ], "source": [
    "!wc -l *.rst | sort -n | head"
  ]
}, {
  "cell_type": "markdown", "id": "handmade-leisure", "metadata": {}, "source": [
  3.1. IPython
  31
"!tail -n 1 length.txt"
]
“If we want to display the first lines of the main text (without the first three lines for the title):”

```
!cat index.rst | head -n 5 | tail -n 2
```


Find

grep

grep finds and prints lines in files that match a regular expression. In the following example, we search for the string Python:

```
[1]: !grep Python ../index.rst
IPython
`IPython <http://ipython.org/>`, or *Interactive Python*, was initially an advanced Python interpreter that has now grown into an extensive project. Today, IPython is not only an interactive interface to Python, but also offers a number of useful syntactic additions for the language. In addition, IPython is

* Miki Tebeka - IPython: The Productivity Booster
```

The option `-w` limits the matches to the word boundaries so that IPython is ignored:

```
[2]: !grep -w Python ../index.rst
```

```
`IPython <http://ipython.org/>`, or *Interactive Python*, was initially an advanced Python interpreter that has now grown into an extensive project. Today, IPython is not only an interactive interface to Python, but also offers a

```

-n shows the line numbers that match:

```
[3]: !grep -n -w Python ../index.rst
```

```
4:`IPython <http://ipython.org/>`, or *Interactive Python*, was initially an advanced Python interpreter that has now grown into an extensive project. Today, IPython is not only an interactive interface to Python, but also offers a

```

-v inverts our search:

```
[4]: !grep -n -v `^ ` ../index.rst
```

```
1:IPython
2:=======
3:
4:`IPython <http://ipython.org/>`, or *Interactive Python*, was initially an advanced Python interpreter that has now grown into an extensive project.
5:Today, IPython is not only an interactive interface to Python, but also offers a

```

-grep has lots of other options. To find out what they are, you can type:

```
[5]: !grep --help
```

3.1. IPython

In the following example we use the -E option and put the pattern in quotes to prevent the shell from trying to interpret it. The ^ in the pattern anchors the match to the start of the line and the . matches a single character.

```
[6]: !grep -n -E "^\.*Python" ../index.rst
1:IPython
```

`find`

find . searches in this directory whereby the search is restricted to directories with -type d.

```
[7]: !find .. -type d
 ..
  ../mypackage
  ../unix-shell
  ../unix-shell/.ipynb_checkpoints
  ../.ipynb_checkpoints
```

With -type f the search is restricted to files.

```
[8]: !find . -type f
  ./index.rst
  ./regex.ipynb
  ./create-delete.ipynb
  ./file-system.ipynb
  ./pipes-filters.ipynb
  ./shell-variables.ipynb
  ./ipynb_checkpoints/grep-find-checkpoint.ipynb
  ./ipynb_checkpoints/file-system-checkpoint.ipynb
  ./ipynb_checkpoints/regex-checkpoint.ipynb
  ./grep-find.ipynb
```

With -mtime the search is limited to the last X days, in our example to the last day:

```
[9]: !find . -mtime -1
  ./index.rst
  ./regex.ipynb
  ./create-delete.ipynb
  ./file-system.ipynb
  ./pipes-filters.ipynb
  ./shell-variables.ipynb
  ./ipynb_checkpoints
  ./ipynb_checkpoints/grep-find-checkpoint.ipynb
  ./ipynb_checkpoints/file-system-checkpoint.ipynb
```

(continues on next page)
With \texttt{-name} you can filter the search by name.

\begin{verbatim}
[10]: !find .. -name ".*.rst"
../index.rst
../unix-shell/index.rst
../extensions.rst
../start.rst
\end{verbatim}

Now we count the characters in the files with the suffix \texttt{.rst}:

\begin{verbatim}
[11]: !wc -c $(find .. -name "*.rst")
    813 ../index.rst
    216 ../unix-shell/index.rst
   2096 ../extensions.rst
   1138 ../start.rst
   4263 total
\end{verbatim}

It is also possible to search for a regular expression in these files:

\begin{verbatim}
[12]: !grep "ipython.org" $(find .. -name "*.rst")
../index.rst:"IPython <http://ipython.org/>", or *Interactive Python*, was initially an
\end{verbatim}

Finally, we filter out all results whose path contains \texttt{ipynb_checkpoints}:

\begin{verbatim}
[13]: !find . -name "*.ipynb" | grep -v ipynb_checkpoints
../regex.ipynb
../create-delete.ipynb
../file-system.ipynb
../pipes-filters.ipynb
../shell-variables.ipynb
../grep-find.ipynb
\end{verbatim}

\section*{Regular expressions}

\subsection*{Common regex metacharacters}

Square brackets define a list or range of characters to be found:

- \texttt{[abc]} matches \texttt{a}, \texttt{b} or \texttt{c}
- \texttt{[a-z]} matches any lower case letter
- \texttt{[A-Za-z]} matches any letter
- \texttt{[A-Za-z0-9]} matches any letter or any digit

Useful special characters:

- \texttt{.} matches any single character
• * matches the preceding element zero or more times, e.g. colou*r matches color, colour, colouur, etc.
• ? matches when the preceding character appears zero or one time, e.g. colou?r matches color and colour
• + matches the preceding element one or more times, e.g. .+ matches .., ..., etc.
• {N} matches the preceding character N times
• {N,} matches the preceding character N or more times
• {N,M} matches the preceding character at least N times, but not more than M times
• \ used to escape the following character when that character is a special character e.g. to find .org you have to use the regular expression \.org because . is the special character that matches any character
• ^ asserts the position at the start of the line
• $ asserts the position at the end of the line
• | means or

And then there are:
• \d matches any single digit
• \w matches any part of word character and is equivalent to [A-Za-z0-9]
• \s matches any space, tab, or newline
• \b asserts that the pattern must match at a word boundary
• /i renders an expression case-insensitive equivalent to [A-Za-z]

Examples

• the whole words colour and color case insensitive with \b[Cc]olou?r\b|\bCOLOU?R\b or /colou?r/i
• Date formats like dd-MM-yyyy with \b\d{2}-\d{2}-\d{4}\b or /\d{2}-\d{2}-\d{4}/
• dd-MM-yyyy or dd-MM-yy at the beginning of a line with ^\d{2}-\d{2}-\d{2,4}

Colour output

[1]: ```export GREP_OPTIONS="--color=always $GREP_OPTIONS"
```

[2]: ```export GREP_COLORS="ms=01;37:mc=01;37:sl=:cx=01;30:fn=35:ln=32:bn=32:se=36"
```

See also:
• www.regular-expressions.info
• Python Regular Expression HOWTO
• Python re
Jupyter Tutorial, Release 0.8.0

Shell variables
Display of all shell variables
[1]: !set
BASH=/usr/local/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:extquote:force_fignore:globasciiranges:
˓→hostcomplete:interactive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_EXECUTION_STRING=set
BASH_LINENO=()
BASH_SOURCE=()
˓→3.0")
BASH_VERSION='5.0.16(1)-release'
CLICOLOR=1
CPPFLAGS=-I/usr/local/opt/openblas/include
DIRSTACK=()
DISPLAY=/private/tmp/com.apple.launchd.WfrReZHn7e/org.macosforge.xquartz:0
...

Showing the value of a variable
[2]: !echo $HOME
/Users/veit

The path variable
It defines the shell’s search path, i.e., the list of directories that the shell looks in for runnable programs.
[1]: !echo $PATH
/Users/veit/.local/share/virtualenvs/jupyter-tutorial-G-MBNaSt/bin:/Users/veit/Library/
˓→Python/3.7/bin:/Users/veit/spack/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/
˓→Library/TeX/texbin:/usr/local/MacGPG2/bin:/opt/X11/bin:/Library/Apple/usr/bin

Creating and changing variables
Creating or overwriting variables
[3]: !export SPACK_ROOT=~/spack

3.1. IPython

37


Append additional specifications

```
[5]: !export PATH=/usr/local/opt/python@3.7/bin:$PATH
```

### 3.1.5 Show objects with display

IPython can display objects such as HTML, JSON, PNG, JPEG, SVG and Latex

#### Images

To display images (JPEG, PNG) in IPython and notebooks, you can use the `Image` class:

```
[1]: from IPython.display import Image
    Image('https://www.python.org/images/python-logo.gif')

[1]: <IPython.core.display.Image object>
```

```
[2]: from IPython.display import SVG
    SVG('https://upload.wikimedia.org/wikipedia/commons/c/c3/Python-logo-notext.svg')

[2]:
```

#### Non-embedded images

- By default, image data is embedded:
  
  Image (‘img_url’)

- However, if the `url` is given as kwarg, this is interpreted as a soft link:
  
  Image (url=’img_url’)

- `embed` can also be specified explicitly:

  Image (url=’img_url’, embed = True)

#### HTML

Python objects can declare HTML representations to be displayed in a notebook:

```
[3]: from IPython.display import HTML
```

```
[4]: %%html
    <ul>
    <li>foo</li>
    <li>bar</li>
    </ul>

[4]: <IPython.core.display.HTML object>
```
**Javascript**

With notebooks, objects can also declare a JavaScript representation. This enables e.g. data visualisations with Javascript libraries like d3.js.

```
from IPython.display import Javascript
welcome = Javascript('alert("Dies ist ein Beispiel für eine durch IPython angezeigte Javascript-Warnung.")');
display(welcome)

<IPython.core.display.Javascript object>
```

For more extensive Javascript you can also use the `%%javascript` syntax.

**LaTeX**

IPython.display also has built-in support for displaying mathematical expressions set in LaTeX and rendered in the browser with MathJax:

```
from IPython.display import Math
Math(r'\int_{-\infty}^{\infty} f(x) e^{2\pi i k} dx')
```

```
F(k) = \int_{-\infty}^{\infty} f(x)e^{2\pi ik}dx
```

```
\nabla \times \vec{B} - \frac{1}{c} \frac{\partial \vec{E}}{\partial t} = \frac{4\pi}{c} \vec{j}
```

Audio

IPython also enables interactive work with sounds. With the `display.Audio` class you can create an audio control that is embedded in the notebook. The interface is analogous to that of the `Image` class. All audio formats supported by the browser can be used.

```
from IPython.display import Audio
```

```
import numpy as np
f = 500.0
rate = 8000.0
```

(continues on next page)
L = 3
times = np.linspace(0,L,rate*L)
signal = np.sin(f*times)

Audio(data=signal, rate=rate)

Links to local files

IPython has built-in classes for generating links to local files. To do this, create a link to a single file with the FileLink object:

```
[10]: from IPython.display import FileLink, FileLinks
    FileLink('magics.ipynb')
```

Alternatively, you can generate a list with links to all files in a directory, e.g.:

```
[11]: FileLinks ('.')
```

Display notebooks

```
[12]: import os, sys, types
    import nbformat

[13]: from pygments import highlight
    from pygments.lexers import PythonLexer
    from pygments.formatters import HtmlFormatter

    from IPython.display import display, HTML

    formatter = HtmlFormatter()
    lexer = PythonLexer()

    # publish the CSS for pygments highlighting
    display(HTML('''
    <style type='text/css'>
    '''))
```
%s
</style>
"""
  % formatter.get_style_defs()
"
</IPython.core.display.HTML object>

```python
[14]:
def show_notebook(fname):
    """display a short summary of the cells of a notebook"""
    nb = nbformat.read(fname, as_version=4)
    html = []
    for cell in nb.cells:
        html.append("<h4>%s cell</h4>" % cell.cell_type)
        if cell.cell_type == 'code':
            html.append(highlight(cell.source, lexer, formatter))
        else:
            html.append("<pre>%s</pre>" % cell.source)
    display(HTML('\n'.join(html)))
show_notebook(os.path.join("mypackage/foo.ipynb"))
<IPython.core.display.HTML object>
```

### 3.1.6 foo.ipynb

```python
[1]:
def bar():
    return "bar"

[2]:
def has_ip_syntax():
    listing = !ls
    return listing

[3]:
def whatsmymame():
    return __name__
```

### 3.1.7 Import notebooks

To be able to develop more modularly, the import of notebooks is necessary. However, since notebooks are not Python files, they are not easy to import. Fortunately, Python provides some hooks for the import so that IPython notebooks can eventually be imported.

```python
[1]:
import os, sys, types

[2]:
import nbformat

  from IPython import get_ipython
  from IPython.core.interactiveshell import InteractiveShell
```

Import hooks usually have two objects:
• **Module Loader** that takes a module name (e.g. `IPython.display`) and returns a module

• **Module Finder**, which finds out if a module is present and tells Python which *loader* to use

But first, let’s write a method that a notebook will find using the fully qualified name and the optional path. E.g. `mypackage.foo` becomes `mypackage/foo.ipynb` and replaces `Foo_Bar` with `Foo Bar` if `Foo_Bar` doesn’t exist.

```
[3]: def find_notebook(fullname, path=None):
    name = fullname.rsplit('.', 1)[-1]
    if not path:
        path = ['']
    for d in path:
        nb_path = os.path.join(d, name + '.ipynb')
        if os.path.isfile(nb_path):
            return nb_path
    # let import Foo_Bar find "Foo Bar.ipynb"
    nb_path = nb_path.replace('_',' ')
    if os.path.isfile(nb_path):
        return nb_path
```

### Notebook Loader

The Notebook Loader does the following three steps:

1. Load the notebook document into memory
2. Create an empty module
3. Execute every cell in the module namespace

Because IPython cells can have an extended syntax, `transform_cell` converts each cell to pure Python code before executing it.

```
[4]: class NotebookLoader(object):
    """Module Loader for IPython Notebooks""
    def __init__(self, path=None):
        self.shell = InteractiveShell.instance()
        self.path = path

    def load_module(self, fullname):
        """import a notebook as a module""
        path = find_notebook(fullname, self.path)

        print ("importing notebook from %s" % path)

        # load the notebook object
        nb = nbformat.read(path, as_version=4)

        # create the module and add it to sys.modules
        # if name in sys.modules:
        #    return sys.modules[name]
        mod = types.ModuleType(fullname)
        mod.__file__ = path
        mod.__loader__ = self
```

(continues on next page)
mod.__dict__['get_ipython'] = get_ipython
sys.modules[fullname] = mod

# extra work to ensure that magics that would affect the user_ns
# magics that would affect the user_ns actually affect the
# notebook module's ns
save_user_ns = self.shell.user_ns
self.shell.user_ns = mod.__dict__

try:
    for cell in nb.cells:
        if cell.cell_type == 'code':
            code = self.shell.input_transformer_manager.transform_cell(cell.source)
            exec(code, mod.__dict__)
finally:
    self.shell.user_ns = save_user_ns
return mod

Notebook Finder

The Finder is a simple object that indicates whether a notebook can be imported based on its file name and that returns
the appropriate loader.

```python
[5]: class NotebookFinder(object):
    """Module Finder finds the transformed IPython Notebook"""
    def __init__(self):
        self.loaders = {}

    def find_module(self, fullname, path=None):
        nb_path = find_notebook(fullname, path)
        if not nb_path:
            return

        key = path
        if path:
            # lists aren't hashable
            key = os.path.sep.join(path)

        if key not in self.loaders:
            self.loaders[key] = NotebookLoader(path)
        return self.loaders[key]
```
Register hook

Now we register NotebookFinder with sys.meta_path:

[6]: `sys.meta_path.append(NotebookFinder())`

Check

Now our notebook `mypackage/foo.ipynb` should be importable with:

[7]:
```python
from mypackage import foo
```

importing notebook from /Users/veit/jupyter-tutorial/docs/basics/ipython/mypackage/foo.
˓→ipynb

Is the Python method bar being executed?

[8]: `foo.bar()`

[8]: 'bar'

... and the IPython syntax?

[9]: `foo.has_ip_syntax()`

[9]: ['debugging.ipynb',
  'display.ipynb',
  'importing.ipynb',
  'index.rst',
  'magics.ipynb',
  'mprun_demo.py',
  'mypackage',
  'myscript.py',
  'profiling.ipynb',
  'shell.ipynb',
  'show.ipynb',
  'start.rst']

Reusable import hook

The import hook can also easily be executed in other notebooks with

[10]: `%run importing.ipynb`

importing notebook from /Users/veit/jupyter-tutorial/docs/basics/ipython/mypackage/foo.
˓→ipynb
3.1.8 IPython extensions

IPython extensions are Python modules that change the behavior of the shell. They are identified by an importable module name and are usually located in .ipython/extensions/.

Some important extensions are already included in IPython: autoreload and storemagic. You can find other extensions in the Extensions Index or on PyPI with the IPython tag.

See also:
- IPython extensions docs

Use extensions

The %load_ext magic can be used to load extensions while IPython is running.

```python
%load_ext myextension
```

Alternatively, an extension can also be loaded each time IPython is started by listing it in the IPython configuration file:

```python
c.InteractiveShellApp.extensions = [  'myextension'
]
```

If you haven't created an IPython configuration file yet, you can do this with:

```bash
$ ipython profile create [profilename]
```

If no profile name is given, default` is used. The file is usually created in `~/.ipython/profile_default/` and named depending on the purpose: ipython_config.py is used for all IPython commands, while ipython_notebook_config.py is only used for commands in IPython notebooks.

Writing IPython extensions

An IPython extension is an importable Python module that has special functions for loading and unloading:

```python
def load_ipython_extension(ipython):    # The `ipython` argument is the currently active `InteractiveShell`    # instance, which can be used in any way. This allows you to register    # new magics or aliases, for example.

def unload_ipython_extension(ipython):  # If you want your extension to be unloadable, put that logic here.
```

See also:
- Defining custom magics
3.1.9 Debugging

IPython contains various tools to analyse faulty code, essentially the exception reporting and the debugger.

Check exceptions with %xmode

If the execution of a Python script fails, a so-called exception is usually thrown and relevant information about the cause of the error is written to a traceback. With the %xmode magic function you can control the amount of information that is displayed in IPython. Let’s look at the following code for this:

```
[1]: def func1(a, b):
    return a / b

def func2(x):
    a = x
    b = x - 1
    return func1(a, b)

[2]: func2(1)
---------------------------------------------------------------------------
ZeroDivisionError Traceback (most recent call last)
<ipython-input-2-7cb498ea7ed1> in <module>
----> 1 func2(1)

<ipython-input-1-586ccabd0db3> in func2(x)
     5 a = x
     6 b = x - 1
----> 7 return func1(a, b)

<ipython-input-1-586ccabd0db3> in func1(a, b)
     1 def func1(a, b):
     2     return a / b
     3     1 def func2(x):
     4         a = x
ZeroDivisionError: division by zero
```

Calling `func2` leads to an error and the traceback shows exactly what happened: each line shows the context of each step that ultimately led to the error. With the %xmode magic function (short for exception mode) we can control which information should be displayed to us.

%xmode takes a single argument, the mode, and there are three options: * Plain * Context *Verbose

The default setting is **Context** and outputs something like the one above. Plain is more compact and provides less information:

```
[3]: %xmode Plain
func2(1)

Exception reporting mode: Plain

Traceback (most recent call last):
```

(continues on next page)
The `Verbose` mode shows some additional information, including the arguments for any functions being called:

```
[4]: %xmode Verbose
    func2(1)
Exception reporting mode: Verbose
```

```
ZeroDivisionError Traceback (most recent call last)
<ipython-input-4-180acea4108b> in <module>
    2 func2(1)
global func2 = <function func2 at 0x106312a60>

<ipython-input-1-586ccabd0db3> in func2(x=1)
    5 a = x
    6 b = x - 1
    7 return func1(a, b)
global func1 = <function func1 at 0x1063129d8>
    a = 1
    b = 0

<ipython-input-1-586ccabd0db3> in func1(a=1, b=0)
    1 def func1(a, b):
    2     return a / b
    3     a = 1
    4     b = 0
    5 def func2(x):
    6     a = x
ZeroDivisionError: division by zero
```

This additional information can help narrow down the reason for the exception. Conversely, however, the `Verbose` mode can lead to extremely long tracebacks in the case of complex code, in which the essential points can hardly be recognized.

3.1. IPython
Debugging

Debugging can help if an error cannot be found by reading a traceback. The Python standard for interactive debugging is the Python debugger pdb. You can use it to navigate your way through the code line by line to see what is possibly causing an error. The extended version for IPython is ipdb.

In IPython, the %debug-magic command is perhaps the most convenient way to debug. If you call it after an exception has been thrown, an interactive debug prompt will automatically open during the exception. Using the ipdb prompt, you can examine the current status of the stack, examine the available variables and even run Python commands.

Let’s look at the last exception, then do some basic tasks:

```python
[5]: %debug
> <ipython-input-1-586ccabd0db3>(2)func1()
1 def func1(a, b):
----> 2 return a / b
3
4 def func2(x):
5   a = x

ipdb> print(a)
1
ipdb> print(b)
0
ipdb> quit
```

However, the interactive debugger does a lot more – we can also go up and down the stack and examine the values of variables:

```python
[6]: %debug
> <ipython-input-1-586ccabd0db3>(2)func1()
1 def func1(a, b):
----> 2 return a / b
3
4 def func2(x):
5   a = x

ipdb> up
> <ipython-input-1-586ccabd0db3>(7)func2()
3
4 def func2(x):
5   a = x
6   b = x - 1
----> 7 return func1(a, b)

ipdb> print(x)
1
ipdb> up
> <ipython-input-4-180acea4108b>(2)<module>()
1 get_ipython().run_line_magic('xmode', 'Verbose')
----> 2 func2(1)

ipdb> down
> <ipython-input-1-586ccabd0db3>(7)func2()
```
def func2(x):
a = x
b = x - 1
----> 7 return func1(a, b)

ipdb> quit

This greatly simplifies the search for the function calls that led to the error.

If you want the debugger to start automatically when an exception is thrown, you can use the %pdb-magic function to enable this behavior:

```
[7]: %pdb
%pdb on
func2(1)
```

Exception reporting mode: Plain
Automatic pdb calling has been turned ON

Traceback (most recent call last):
  File "<ipython-input-7-f80f6b5cecf3>", line 3, in <module>
    func2(1)
  File "<ipython-input-1-586ccabd0db3>", line 7, in func2
    return func1(a, b)
  File "<ipython-input-1-586ccabd0db3>", line 2, in func1
    return a / b

ZeroDivisionError: division by zero

> <ipython-input-1-586ccabd0db3>(2)func1()
  1 def func1(a, b):
  ---> 2 return a / b
  3
def func2(x):
  5 a = x

ipdb> print(b)
0
ipdb> quit

If you have a script that you want to run in interactive mode from the start, you can do so with the command %run -d.
Essential commands of the `ipdb`

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>list</code></td>
<td>Show the current location in the file</td>
</tr>
<tr>
<td><code>h</code> (help)</td>
<td>Display a list of commands or find help on a specific command</td>
</tr>
<tr>
<td><code>q</code> (quit)</td>
<td>Terminates the debugger and the program</td>
</tr>
<tr>
<td><code>c</code> (continue)</td>
<td>Exit the debugger, continue in the program</td>
</tr>
<tr>
<td><code>n</code> (next)</td>
<td>Go to the next step in the program</td>
</tr>
<tr>
<td><code>&lt;enter&gt;</code></td>
<td>Repeat the previous command</td>
</tr>
<tr>
<td><code>p</code> (print)</td>
<td>Print variables</td>
</tr>
<tr>
<td><code>s</code> (step)</td>
<td>Step into a subroutine</td>
</tr>
<tr>
<td><code>r</code> (return)</td>
<td>Return from a subroutine</td>
</tr>
</tbody>
</table>

Further information on the IPython debugger can be found at [ipdb](https://docs.python.org/3/library/ipdb.html).

3.2 Jupyter

3.2.1 Notebook

**Keyboard shortcuts**

If you know the Jupyter keyboard shortcuts, you can work much more efficiently with notebooks.

Jupyter notebooks have two different keyboard input modes.

In **edit mode** you can enter code or text in a cell. This is indicated by a green cell border.

**Command mode** binds the keyboard to notebook-level commands and is indicated by a gray cell border with a blue left border.
## Command mode

This mode is available with `esc`.

<table>
<thead>
<tr>
<th>Action</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call up edit mode</td>
<td><code>Esc</code></td>
</tr>
<tr>
<td>Execute cell, select below</td>
<td><code>Enter</code></td>
</tr>
<tr>
<td>Run cell</td>
<td><code>Esc</code></td>
</tr>
<tr>
<td>Execute cell, insert below</td>
<td><code>Enter</code></td>
</tr>
<tr>
<td>Y to the code cell</td>
<td><code>Y</code></td>
</tr>
<tr>
<td>M to the markdown cell</td>
<td><code>M</code></td>
</tr>
<tr>
<td>R to the raw view</td>
<td><code>R</code></td>
</tr>
<tr>
<td>1, 2 etc. to h1, h2 etc.</td>
<td><code>1, 2</code></td>
</tr>
<tr>
<td>↑, K to the cell above</td>
<td><code>↑, K</code></td>
</tr>
<tr>
<td>↓, J to the next cell</td>
<td><code>↓, J</code></td>
</tr>
<tr>
<td>A, B Insert cell above or below</td>
<td><code>A, B</code></td>
</tr>
<tr>
<td>X Cut out selected cell</td>
<td><code>X</code></td>
</tr>
<tr>
<td>C Copy selected cell</td>
<td><code>C</code></td>
</tr>
<tr>
<td>V Insert line above</td>
<td><code>V</code></td>
</tr>
<tr>
<td>V Insert cell below</td>
<td><code>V</code></td>
</tr>
<tr>
<td>Z Undo delete cell</td>
<td><code>Z</code></td>
</tr>
<tr>
<td>D-D Delete cell</td>
<td><code>D-D</code></td>
</tr>
<tr>
<td>M Merge cell with next</td>
<td><code>M</code></td>
</tr>
<tr>
<td>S Save and checkpoint</td>
<td><code>S</code></td>
</tr>
<tr>
<td>L Toggle line numbers</td>
<td><code>L</code></td>
</tr>
<tr>
<td>0 Toggle output</td>
<td><code>0</code></td>
</tr>
<tr>
<td>0 Toggle the scrolling of the output</td>
<td><code>0</code></td>
</tr>
<tr>
<td>Esc, Q Exit command mode</td>
<td><code>Esc, Q</code></td>
</tr>
<tr>
<td>H Help dialog for keyboard shortcuts</td>
<td><code>H</code></td>
</tr>
<tr>
<td>I-I Interrupt kernel</td>
<td><code>I-I</code></td>
</tr>
<tr>
<td>Θ-Θ Restart the kernel</td>
<td><code>Θ-Θ</code></td>
</tr>
<tr>
<td>⇓ scroll down</td>
<td><code>⇓</code></td>
</tr>
<tr>
<td>⇑ scroll up</td>
<td><code>⇑</code></td>
</tr>
<tr>
<td>ignore</td>
<td><code>ignore</code></td>
</tr>
</tbody>
</table>

## Edit mode

This mode becomes available with `Enter`. 

3.2. Jupyter
Own keyboard shortcuts

If necessary, you can also define your own keyboard shortcuts (see Keyboard Shortcut Customization).

Jupyter paths and configuration

Configuration files are usually stored in the ~/.jupyter directory. However, another directory can be specified with the environment variable JUPYTER_CONFIG_DIR. If Jupyter cannot find a configuration in JUPYTER_CONFIG_DIR, Jupyter runs through the search path with {sys.prefix}/etc/jupyter/ and then for Unix /usr/local/etc/jupyter/ and /etc/jupyter/, for Windows %PROGRAMDATA%\jupyter\.

You can have the currently used configuration directories listed with:

```
$ pipenv run jupyter --paths
```

```
config:  
/Users/veit/.jupyter  
/Users/veit/.local/share/virtualenvs/jupyter-tutorial--q5BvmfG/bin/..etc/jupyter  
/usr/local/etc/jupyter  
/etc/jupyter
... 
```
Create the configuration files

You can create a standard configuration with:

```
$ pipenv run jupyter notebook --generate-config
Writing default config to: /Users/veit/.jupyter/jupyter_notebook_config.py
```

More generally, configuration files can be created for all Jupyter applications with:

```
$ pipenv run jupyter {application} --generate-config
```

Change the configuration

... by editing the configuration file

e.g. in `jupyter_notebook_config.py`:

```
c.NotebookApp.port = 8754
```

If the values are saved as `list`, `dict` or `set`, they can also be supplemented with `append`, `extend`, `prepend`, `add` and `update`, e.g.:

```
c.TemplateExporter.template_path.append('./templates')
```

... with the command line

e.g.:

```
$ pipenv run jupyter notebook --NotebookApp.port=8754
```

There are aliases for frequently used options such as for `--port` or `--no-browser`.
The command line options override options set in a configuration file.

See also:

traitlets.config

Introduction

Jupyter Notebooks extend the console-based approach to interactive computing with a web-based application, with which the entire process can be recorded: from developing and executing the code to documenting and presenting the results. Jupyter notebooks combine three different components:

**Interactive Computing Protocol:** Open network protocol based on JSON data via ZMQ and WebSockets.

**Notebook Document Format:** Open JSON-based document format with full records of the user’s sessions and the code contained therein.

**Kernel:** Processes that execute interactive code in a particular programming language and return output to the user.
3.2.2 JupyterHub

**JupyterHub** is a multi-user server for Jupyter Notebooks, which can create and manage many different instances of Jupyter Notebooks and which acts as a proxy.

### Installation

1. Install Python3.5 and pip:

```bash
# apt-get update
# apt install python3
# python3 -V
Python 3.7.3
# apt install python3-pip
```

2. Create service user jupyter:

```bash
# useradd -s /bin/bash -md /srv/jupyter jupyter
```

3. Clone the repository as service user jupyter:

```bash
# su - jupyter
$ git clone https://github.com/veit/jupyter-tutorial.git
```

4. Install Pipenv:

```bash
$ python3 -m pip install --user pipenv
```

This installs Pipenv in USER_BASE.

5. Determine USER_BASE and enter it in PATH:

```bash
$ python3 -m site --user-base
/srv/jupyter/.local
```

Then the bin directory has to be appended and added to PATH, so:

```bash
export PATH=/srv/jupyter/.local/bin:$PATH
```

Finally the changed profile is read in with:

```bash
$ source ~/.profile
```

6. Edit the Pipfile in the unpacked archive and enter your current Python version in this section:

```python
[requires]
python_version = ""
```

1. Create a virtual environment and install JupyterHub:

```bash
$ cd jupyter-tutorial/
$ pipenv install
```

2. Install nodejs and npm:
# apt install curl
# cd ~
# curl -sL https://deb.nodesource.com/setup_10.x -o nodesource_setup.sh
# bash nodesource_setup.sh
# apt install nodejs
# nodejs -v
v10.15.3
# npm -v
6.10.2

10.x indicates the major version of nodejs.

3. Install the npm packages:

   # npm install

4. Install the HTTP-Proxy:

   # npm install -g configurable-http-proxy
   + configurable-http-proxy@4.1.0
   added 47 packages from 62 contributors in 6.208s

5. Testing the installation:

   $ pipenv run jupyterhub
   ...

   With ctrl-c you can end the process again.

**Configuration**

**JupyterHub configuration**

   $ pipenv run jupyterhub --generate-config
   Writing default config to: jupyterhub_config.py

**See also:**

- JupyterHub Configuration Basics
- JupyterHub Networking basics
System service for JupyterHub

1. Finding the Python virtual environment:

```bash
$ cd ~/jupyter-tutorial
$ pipenv --venv
/srv/jupyter/.local/share/virtualenvs/jupyter-tutorial-aFv4x91W
```

2. Add a new systemd unit file `/etc/systemd/system/jupyterhub.service` with this command:

```bash
# systemctl edit --force --full jupyterhub.service
```

Insert your according Python virtual environment.

```
[Unit]
Description=Jupyterhub

[Service]
User=root
Environment="PATH=/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/srv/jupyter/.local/share/virtualenvs/jupyter-tutorial-aFv4x91W/bin"
ExecStart=/srv/jupyter/.local/share/virtualenvs/jupyter-tutorial-aFv4x91W/bin/jupyterhub -f /srv/jupyter/jupyter-tutorial/jupyterhub_config.py

[Install]
WantedBy=multi-user.target
```

3. The JupyterHub can be managed with:

```bash
# systemctl <start|stop|status> jupyterhub
```

4. To ensure that the service is also loaded when the system is started, the following is called:

```bash
# systemctl enable jupyterhub.service
```

Created symlink /etc/systemd/system/multi-user.target.wants/jupyterhub.service → /etc/systemd/system/jupyterhub.service.

TLS encryption

Since JupyterHub includes authentication and allows the execution of any code, it should not be executed without SSL (HTTPS). To do this, an official, trustworthy SSL certificate must be created. After you have received and installed a key and a certificate, you don’t configure the JupyterHub itself, but the upstream Apache web server.

1. For this purpose, the additional modules are first activated with

```bash
# a2enmod ssl rewrite proxy proxy_http proxy_wstunnel
```

2. Then the VirtualHost can be configured in `/etc/apache2/sites-available/jupyter.cusy.io.conf`

```bash
# redirect HTTP to HTTPS
<VirtualHost 172.31.50.170:80>
    ServerName jupyter.cusy.io
    ServerAdmin webmaster@cusy.io
</VirtualHost>
```

(continues on next page)
ErrorLog ${APACHE_LOG_DIR}/jupyter.cusy.io_error.log
CustomLog ${APACHE_LOG_DIR}/jupyter.cusy.io_access.log combined

Redirect / https://jupyter.cusy.io/
</VirtualHost>

<VirtualHost 172.31.50.170:443>
  ServerName jupyter.cusy.io
  ServerAdmin webmaster@cusy.io
  
  # configure SSL
  SSLEngine On
  SSLCertificateFile /etc/ssl/certs/jupyter.cusy.io_cert.pem
  SSLCertificateKeyFile /etc/ssl/private/jupyter.cusy.io_sec_key.pem
  # for an up-to-date SSL configuration see e.g.
  # https://ssl-config.mozilla.org/
  
  # Use RewriteEngine to handle websocket connection upgrades
  RewriteEngine On
  RewriteCond %{HTTP:Connection} Upgrade [NC]
  RewriteCond %{HTTP:Upgrade} websocket [NC]
  RewriteRule /(.*) ws://127.0.0.1:8000/$1 [P,L]

  <Location />
  # preserve Host header to avoid cross-origin problems
  ProxyPreserveHost on
  # proxy to JupyterHub
  ProxyPass     http://127.0.0.1:8000/
  ProxyPassReverse http://127.0.0.1:8000/
  </Location>

  ErrorLog ${APACHE_LOG_DIR}/jupyter.cusy.io_error.log
  CustomLog ${APACHE_LOG_DIR}/jupyter.cusy.io_access.log combined
</VirtualHost>

3. This VirtualHost is activated with

   # a2ensite jupyter.cusy.io.conf

4. Finally, the status of the Apache web server is checked with

   # systemctl status apache2
   apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
   Active: active (running) (Result: exit-code) since Mon 2019-03-25 16:50:26 CET; 1 day 22h ago
   Process: 31773 ExecReload=/usr/sbin/apachectl graceful (code=exited, status=0/SUCCESS)
   Main PID: 20273 (apache2)
   Tasks: 55 (limit: 4915)
Cookie Secret

The cookie secret is used to encrypt the browser cookies that are used for authentication.

1. The cookie secret can e.g. be created with

   ```bash
   $ openssl rand -hex 32 > /srv/jupyterhub/venv/jupyterhub_cookie_secret
   ```

2. The file should not be readable by either group or anonymous:

   ```bash
   $ chmod 600 /srv/jupyterhub/venv/jupyterhub_cookie_secret
   ```

3. Finally it will be entered in the `jupyterhub_config.py` file:

   ```python
   c.JupyterHub.cookie_secret_file = 'jupyterhub_cookie_secret'
   ```

Proxy authentication token

The hub authenticates its requests to the proxy using a secret token that the hub and proxy agree on. Usually, the proxy authentication token does not need to be set, as the hub itself generates a random key. This means that the proxy has to be restarted every time unless the proxy is a subprocess of the hub.

1. Alternatively, the value can e.g. be generated with

   ```bash
   $ openssl rand -hex 32
   ```

2. It can then be entered in the configuration file, e.g. with

   ```python
   c.JupyterHub.proxy_auth_token = '18a0335b7c2e7edeaf7466894a32bea8d1c3cff4b07860298dbe353ecb179fc6'
   ```

Create service nbviewer

1. Configuring the notebook viewer as a JupyterHub service has the advantage that only users who have previously logged on to JupyterHub can call up the nbviewer instance. This can be used to protect access to notebooks as a JupyterHub service in `/srv/jupyter/jupyter-tutorial/jupyterhub_config.py`:

   ```python
   c.JupyterHub.services = [
   {
       'name': 'nbviewer',
       'url': 'http://127.0.0.1:9000',
       'cwd': '/srv/jupyterhub/nbviewer-repo',
       'command': ['/srv/jupyter/.local/share/virtualenvs/jupyter-tutorial--q5BvmfG/bin/python', '-m', 'nbviewer']
   }
   ```
name  The path name under which the notebook viewer can be reached:: /services/<name>
url  Protocol, address and port used by nbviewer
cwd  The path to the nbviewer repository
command  Command to start nbviewer

3.2.3 nbconvert

nbconvert  converts notebooks to other formats

See also:
Install Jupyter Notebook: nbconvert is part of the Jupyter ecosystem.

Installation

$ pipenv install nbconvert

Important:  To be able to use all functions of nbconvert, Pandoc and TeX (especially XeLaTeX) are required. These must be installed separately.

Install Pandoc

nbconvert uses Pandoc to convert Markdown to formats other than HTML.

•  for Ubuntu and Debian:

  $ sudo apt install pandoc

•  for Mac OSX:

  $ brew install pandoc

Install Tex

For the conversion to PDF, nbconvert uses the Tex ecosystem in preparation: A .tex file is created, which is converted into a PDF by the XeTeX engine.

•  for Ubuntu and Debian:

  $ sudo apt install texlive-xetex

•  for Mac OSX:

  MacTeX
Use on the command line

```
$ jupyter nbconvert --to FORMAT mynotebook.ipynb
```

**latex** creates a NOTEBOOK_NAME.tex file and possibly images as PNG files in a folder. With **--template** you can choose between one of two templates:

**--template article** default

Latex article, derived from the Sphinx how-to

**--template report** Latex report with table of contents and chapters

**pdf**

creates a PDF over latex. Supports the same templates as latex.

**slides** creates Reveal.js slides.

**script** kconverts the notebook into an executable script. This is the easiest way to create a Python script or a script in another language.

**Note**: If a notebook contains *Magics*, then this can possibly only be carried out in one Jupyter session.

We can e.g. convert `docs/ipython/mypackage/foo.ipynb` into a Python script with:

```
$ pipenv run jupyter nbconvert --to script docs/basics/ipython/mypackage/foo.ipynb
```

The result is then `foo.py` with:

```
#!/usr/bin/env python
# coding: utf-8
#
`foo.ipynb`

# In[1]:
def bar():
    return "bar"

# In[2]:
def has_ip_syntax():
    listing = get_ipython().getoutput('ls')
    return listing

# In[3]:
def whatsmyname():
    return __name__
```

**Note**: In order to assign notebook cells to slides, you should select *View → Cell Toolbar → Slideshow*. Then a menu is displayed in each cell at the top right with the options: *Slide, Sub-Slide, Fragment, Skip, Notes.*
**Note:** Lecture notes require a local copy of reveal.js. The following option can be specified so that nbconvert can find this: `--reveal-prefix /path/to/reveal.js`.

Further details for **FORMAT** are asciidoc, custom, html, markdown, notebook, and rst.

### nb2xls

nb2xls converts Jupyter notebooks into Excel files (.xlsx) taking into account pandas DataFrames and Matplotlib outputs. However, the input cells are not converted and only part of the Markdown is converted.

### Own exporters

See also: Customizing exporters allows you to write your own exporters.

#### 3.2.4 nbviewer

**nbviewer** is a web service: Renders Jupyter notebooks as static web pages.

**Installation**

1. The Notebook Viewer requires several binary packages that have to be installed on our system, for Ubuntu and Debian:

   ```
   $ sudo apt install libmemcached-dev libcurl4-openssl-dev pandoc libevent-dev
   ```

   for Mac OSX:

   ```
   $ brew install libmemcached openssl pandoc libevent
   ```

2. The Jupyter Notebook Viewer can then be installed in a new virtual environment with:

   ```
   $ mkdir nbviewer
   $ cd !$
   cd nbviewer
   ```

   **Note:** The notebook app outputs the error `AttributeError: module 'tornado.gen' has no attribute 'Task' with current versions of Tornado. This error does not occur with tornado<6.0, see also Delete Terminal Not Working with Tornado version 6.0.1:

   ```
   $ pipenv install "tornado<6.0"
   ```

   Now nbviewer can also be installed:

   ```
   $ pipenv install nbviewer
   ```

3. For testing, the server can be started with:
Extending the Notebook Viewer

The notebook viewer can be extended to include providers, see Extending the Notebook Viewer.

Access control

If the viewer is run as Create service nbviewer, only users who have authenticated themselves on the JupyterHub can access the nbviewer's notebooks.

3.2.5 Kernels

Install, view and start the kernel

Install a kernel

pipenv run python -m ipykernel install can be called with the following options:

--user installs the kernel for the current user and not system-wide

name <NAME> gives a name for the kernelspec. This is required in order to be able to use several IPython kernels at the same time, e.g.:

```
$ cd /path/to/your/jupyter/
$ pipenv run python -m ipykernel install --user --name mykernel --display-name "My␣˓→Kernel"
```

Installed kernelspec mykernel in /Users/veit/Library/Jupyter/kernels/mykernel

ipykernel install creates a kernelspec file in JSON format for the current Python environment, e.g.:

```
{
"display_name": "My Kernel",
"language": "python"
"argv": [
"/Users/veit/.local/share/virtualenvs/mykernel-7y9G693U/bin/python",
"-m",
"ipykernel_launcher",
"-f",
"{connection_file}"
],
}
```

display_name The name of the kernel as it should be displayed in the browser. In contrast to the kernel name used in the API, it can contain any Unicode characters.

language The name of the language of the kernel. If no suitable kernelspec key is found when loading notebooks, a kernel with a suitable language is used. In this way, a notebook written for a Python or Julia kernel can be linked to the user's Python or Julia kernel, even if it does not have the same name as the author's.

argv A list of command line arguments used to start the kernel. {connection_file} refers to a file that contains the IP address, ports, and authentication key required for the connection. Usually this JSON file is saved in a safe place of the current profile:
interrupt_mode can be either signal or message and specifies how a client should interrupt the execution of a cell on this kernel.

- **signal** sends an interrupt, e.g. SIGINT on POSIX systems
- **message** sends an interrupt_request, see also Kernel Interrupt.

env dict with environment variables to be set for the kernel. These are added to the current environment variables before the kernel starts.

metadata dict with additional attributes for this kernel. Used by clients to support the kernel selection. Metadata added here should have a namespace for the tool to read and write that metadata.

You can edit this `.kernelspec` file at a later time.

### Show available kernels

```
$ pipenv run jupyter kernelspec list
Available kernels:
  mykernel /Users/veit/Library/Jupyter/kernels/mykernel
  python2 /Users/veit/Library/Jupyter/kernels/python2
  python3 /Users/veit/.local/share/virtualenvs/jupyter-tutorial--q5BvmfG/bin/../share/jupyter/kernels/python3
```

### Start kernel

```
$ pipenv run jupyter console --kernel mykernel
Jupyter console 6.0.0
...

In [1]:
```

With `ctrl + d` you can exit the kernel again.
Python2

**Note:** IPython 6.0 has ended support for Python 2. So if you want to install IPython for Python 2, use an older version:

```bash
$ mkdir -p kernels/python2
$ cd !$
cd kernels/python2
$ pipenv --two
Creating a virtualenv for this project...
...
$ pipenv install "ipykernel<=6"
Installing ipykernel<=6...
...
$ pipenv run python2 -m ipykernel install --user --name python2 --display-name "Python 2"
Installed kernelspec python2 in /Users/veit/Library/Jupyter/kernels/python2
```

**What's new in Python 3.8?**

In Python 3.8, the syntax is simplified and support for C libraries is also improved. Below is a brief overview of some of the new features. You can get a complete overview in What's New In Python 3.8.

**Installation**

**Check**

```
[1]: !python3 -V
Python 3.8.0
```

or

```
[ ]: import sys
assert sys.version_info[:2] >= (3, 8)
```

**Assignment Expressions: Walrus operator :=**

So far, e.g. `env_base` can be determined by pip as follows:

```
[ ]: import os
[ ]: def _getuserbase():
    env_base = os.environ.get("PYTHONUSERBASE", None)
    if env_base:
        return env_base
```

This can now be simplified with:
Multiple nested if, such as in `cpython/Lib/copy.py`, can also be avoided. This

```python
from copyreg import dispatch_table

def copy(x):
    cls = type(x)
    reducer = dispatch_table.get(cls)
    if reducer:
        rv = reducer(x)
    else:
        reducer = getattr(x, '__reduce_ex__', None)
        if reducer:
            rv = reducer(4)
        else:
            reducer = getattr(x, '__reduce__', None)
            if reducer:
                rv = reducer()
            else:
                raise Error("un(deep)copyable object of type %s" % cls)
```

becomes that:

```python
def copy(x):
    cls = type(x)
    reducer = dispatch_table.get(cls)
    if reducer:= dispatch_table.get(cls):
        rv = reducer(x)
    elif reducer:= getattr(x, '__reduce_ex__', None):
        rv = reducer(4)
    elif reducer:= getattr(x, '__reduce__', None):
        rv = reducer()
    else:
        raise Error("un(deep)copyable object of type %s" % cls)
```

### Positional-only parameters

In Python 3.8 a function parameter can be specified position-related with `/`. Several Python functions implemented in C do not allow keyword arguments. This behavior can now be emulated in Python itself, e.g. for the `pow()` function:

```python
def pow(x, y, z=None, /):
    "Emulate the built in pow() function"
    r = x ** y
    return r if z is None else r**z
```
**f-strings support** = for self-documenting expressions and debugging

```python
[ ]: user = 'veit'
    member_since = date(2012, 1, 30)
    f'{{user=}} {{member_since=}}'
```

**Debug and release build use the same ABI**

So far, a consistent application binary interface (ABI) should be guaranteed by Spack. However, this did not include using Python in the debug build. Python 3.8 now also supports ABI compatibility for debug builds. The Py_TRACE_REFS macro can now be set with the `./configure --with-trace-refs` option.

**New C API**

PEP 587 adds a new C API for configuring the Python initialisation, which offers more precise control of the entire configuration and better error reports.

**Vectorcall - a fast protocol for CPython**

The protocol is not yet fully implemented; this will probably come with Python 3.9. However, you can already get a full description in PEP 590.

**Update – or not?**

The following is a brief overview of the problems you may encounter when switching to Python 3.8:

**Missing packages**

- opencv-python

**Bugs**

- Python 3.7.1 was released 4 months after the first major release with a long list of bug fixes. Something similar is to be expected with Python 3.8.

**Syntax**

- Very few code analysis tools and autoformatters can already handle the syntax changes of Python 3.8
Why update anyway?

Since the upgrade will take some time, it can be tempting to postpone the move indefinitely. Why should you concern yourself with incompatibilities in new versions when your current version works reliably?

The problem is that your Python is not supported indefinitely, nor will the libraries you use will support all older Python versions indefinitely. And the longer you delay an update, the bigger and riskier it will be. Therefore, the update to the new major version of Python is usually recommended a few months after the first release.

Porting

See Porting to Python 3.8

What's new in Python 3.9?

With Python 3.9, a new release cycle is used for the first time: in the future, new releases will appear annually (see also PEP 602). The developers hope that they will get faster feedback on new features.

With the first published release candidate, Python should also have a stable binary interface (application binary interface, ABI): there should no longer be any ABI changes in the 3.9 series, which means that extension modules no longer have to be recompiled for each version.

You can find more information in What’s New In Python 3.9.

In the following, I’ll give you a brief overview of some of the new features.

Installation

Check

[1]: `!python3 -V`

```
Python 3.9.0rc1
```

or

[2]: `import sys
assert sys.version_info[:2] >= (3, 9)`

PEP 584: Dictionary Merge and Update Operators

Operators for the built-in `dict` class are now similar to those for concatenating lists: Merge (`|`) and Update (`|=`). This eliminates various disadvantages of the previous methods `dict.update`, `{**d1, **d2}` and `collections.ChainMap`. 
Example ipykernel/ipykernel/kernelapp.py

```python
[ ]: kernel_aliases = dict(base_aliases)
kernel_aliases.update({
    'ip': 'IPKernelApp.ip',
    'hb': 'IPKernelApp.hb_port',
    'shell': 'IPKernelApp.shell_port',
    'iopub': 'IPKernelApp.iopub_port',
    'stdin': 'IPKernelApp.stdin_port',
    'control': 'IPKernelApp.control_port',
    'f': 'IPKernelApp.connection_file',
    'transport': 'IPKernelApp.transport',
})

kernel_flags = dict(base_flags)
kernel_flags.update({
    'no-stdout': (  
        {'IPKernelApp': {'no_stdout': True}},
        "redirect stdout to the null device"),
    'no-stderr': (  
        {'IPKernelApp': {'no_stderr': True}},
        "redirect stderr to the null device"),
    'pylab': (  
        {'IPKernelApp': {'pylab': 'auto'}},
        """Pre-load matplotlib and numpy for interactive use with
        the default matplotlib backend."""),
    'trio-loop': (  
        {'InteractiveShell': {'trio_loop': False}},
        'Enable Trio as main event loop.'
    ),
})
```

can be simplified with:

```python
[ ]: kernel_aliases = base_aliases | {
    'ip': 'KernelApp.ip',
    'hb': 'KernelApp.hb_port',
    'shell': 'KernelApp.shell_port',
    'iopub': 'KernelApp.iopub_port',
    'stdin': 'KernelApp.stdin_port',
    'parent': 'KernelApp.parent',
}

if sys.platform.startswith('win'):
    kernel_aliases ['interrupt'] = 'KernelApp.interrupt'

kernel_flags = base_flags | {
    'no-stdout': (  
        {'KernelApp': {'no_stdout': True}},
        "stdout auf das Nullgerät umleiten"),
    'no-stderr': (  
        {'KernelApp': {'no_stderr': True}},
        "stderr auf das Nullgerät umleiten"),
```
Example matplotlib/legend.py

```python
hm = default_handler_map.copy()
hm.update(self._custom_handler_map)
return hm
```
can be simplified with:

```python
return default_handler_map | self._handler_map
```

PEP 616: removeprefix() and removesuffix() for string methods

With `str.removeprefix(prefix)` and `str.removesuffix(suffix)` you can easily remove prefixes and suffixes. Similar methods have also been added for `bytes`, `bytearray` objects, and `collections.UserString`. All in all, this should lead to less fragile, better performing and more readable code.

Example find_recursionlimit.py

```python
if test_func_name.startswith("test_"):
    print(test_func_name[5:])
else:
    print(test_func_name)
```
can be simplified with:

```python
print (test_func_name.removeprefix ("test_"))
```

Example deccheck.py

```python
if funcname.startswith("context."):
    self.funcname = funcname.replace("context.", "")
    self.contextfunc = True
else:
    self.funcname = funcname
```
can be simplified with:

```python
self.contextfunc = funcname.startswith ("context.")
self.funcname = funcname.removeprefix ("context.")
```
PEP 585: Additional generic types

In *Type Annotations*, e.g. `list` or `dict` can be used directly as generic types – they no longer have to be imported separately from `typing`. Importing `typing` is thus deprecated.

Example

```
[ ]: def greet_all(names: list[str]) -> None:
    for name in names:
        print("Hello", name)
```

PEP 617: New PEG parser

Python 3.9 now uses a PEG (Parsing Expression Grammar) parser instead of the previous LL parser. This has i.a. the following advantages:

- the parsing of abstract syntax trees (AST) is simplified considerably
- Left recursion becomes possible
- The creation of concrete syntax trees (CST) is possible

The new parser is therefore more flexible and should be used primarily when designing new language functions. The `ast` module is already using the new parser without the output having changed.

In Python 3.10, the old parser and all functions that depend on it – mainly the obsolete `parser` module - are deleted. Only in Python 3.9 you can return to the LL parser on the command line with `-X oldparser` or with the environment variable `PYTHONOLDPARSER=1`.

What's New In Python 3.10

See also:
- What's New In Python 3.10

```
[1]: import sys
    assert sys.version_info[:2] >= (3, 10)
```

Better error messages

Syntax Errors

- When parsing code that contains unclosed parentheses or brackets the interpreter now includes the location of the unclosed bracket of parentheses instead of displaying `SyntaxError: unexpected EOF`.
- `SyntaxError` exceptions raised by the interpreter will now highlight the full error range of the expression that consistsutes the syntax error itself, instead of just where the problem is detected.
- Specialised messages for `SyntaxError` exceptions have been added e.g. for
  - missing : before blocks
  - unparenthesised tuples in comprehensions targets
– missing commas in collection literals and between expressions
– missing : and values in dictionary literals
– usage of = instead of == in comparisons
– usage of * in f-strings

**Indentation Errors**

- Many `IndentationError` exceptions now have more context.

**Attribute Errors**

- `AttributeError` will offer suggestions of similar attribute names in the object that the exception was raised from.

**Name Errors**

- `NameError` will offer suggestions of similar variable names in the function that the exception was raised from.

**Structural Pattern Matching**

Many functional languages have a `match` expression, for example Scala, Rust, F#.

A `match` statement takes an expression and compares it to successive patterns given as one or more case blocks. This is superficially similar to a switch statement in C, Java or JavaScript, but much more powerful.

```python
[2]: def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 401:
            return "Unauthorized"
        case 403:
            return "Forbidden"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something else"
```

**Note:**

Only in this case `_` acts as a wildcard that never fails and **not** as a variable name.

The cases not only check for equality, but rebind variables that match the specified pattern. For example:
[3]:
NOT_FOUND = 404
retcode = 200

match retcode:
    case NOT_FOUND:
        print('not found')

print(f"Current value of {NOT_FOUND}")

not found
Current value of NOT_FOUND=200

«If this poorly-designed feature is really added to Python, we lose a principle I’ve always taught students: «if you see an undocumented constant, you can always name it without changing the code’s meaning.» The Substitution Principle, learned in algebra? It’ll no longer apply.» – Brandon Rhodes

«... the semantics of this can be quite different from switch. The cases don’t simply check equality, they rebind variables that match the specified pattern.» – Jake VanderPlas

Symbolic constants

Patterns may use named constants. These must be dotted names to prevent them from being interpreted as capture variable:

[4]:
from enum import Enum

class Color(Enum):
    RED = 0
    GREEN = 1
    BLUE = 2

color = Color(2)

match color:
    case color.RED:
        print("I see red!")
    case color.GREEN:
        print("Grass is green")
    case color.BLUE:
        print("I'm feeling the blues :(")

I'm feeling the blues :

«... “case CONSTANT” actually matching everything and assigning to a variable named CONSTANT»
– Armin Ronacher

See also:
- Structural pattern matching for Python
- PEP 622 – Structural Pattern Matching superseded by
- PEP 634: Specification
- PEP 635: Motivation and Rationale
- PEP 636: Tutorial
R-Kernel

1. ZMQ
   For Ubuntu & Debian:
   
   ```
   $ sudo apt install libzmq3-dev libcurl4-openssl-dev libssl-dev jupyter-core jupyter-client
   ```

2. R packages
   
   ```
   $ R
   > install.packages(c('crayon', 'pbdZMQ', 'devtools'))
   ...
   --- Please select a CRAN mirror for use in this session ---
   ...
   33: Germany (Münster) [https]
   ...
   Selection: 33
   > devtools::install_github(paste0('IRkernel/', c('repr', 'IRdisplay', 'IRkernel')))
   Downloading GitHub repo IRkernel/repr@master
   from URL https://api.github.com/repos/IRkernel/repr/zipball/master
   ...
   ```

3. Deploy the kernel
   
   ```
   > IRkernel::installspec()
   ...
   [InstallKernelSpec] Installed kernelspec ir in /Users/veit/Library/Jupyter/kernels/---ir3.3.3/share/jupyter/kernels/ir
   ```

   You can also deploy the kernel system-wide:
   
   ```
   > IRkernel::installspec(user = FALSE)
   ```

See also:

- IRkernel Installation

overview

The Jupyter team manages the IPython-Kernel. In addition to Python, many other languages can be used in notebooks. The following Jupyter kernels are widely used:

- R
  - IRKernel: Docs | GitHub
  - IRdisplay: GitHub
  - Repr: GitHub

3.2. Jupyter
3.2.6 ipywidgets

ipywidgets are interactive widgets for Jupyter notebooks. They extend notebooks by the possibility that users can enter data themselves, manipulate data and see the changed results.

**Examples**

IPython includes an interactive widget architecture that combines Python code running in the kernel and JavaScript/HTML/CSS running in the browser. These widgets allow users to interactively examine their code and data.

**Interact function**

ipywidgets.interact automatically creates user interface (UI) controls to interactively explore code and data.

```python
[1]: from __future__ import print_function
    from ipywidgets import interact, interactive, fixed, interact_manual
    import ipywidgets as widgets

In the simplest case, interact automatically generates controls for function arguments and then calls the function with those arguments when you interactively edit the controls. The following is a function that returns its only argument x.

[2]: def f(x):
    return x

**Slider**

If you specify a function with an integer keyword argument (x=10), a slider is generated and bound to the function parameter:

[3]: interact(f, x=10);
```
### Checkbox

If you specify True or False, `interact` generates a checkbox:

```python
[4]: interact(f, x=True);
```
```
interactive(children=(Checkbox(value=True, description='x'), Output()), _dom_classes=('widget-interact',))
```

### Text area

If you pass a string, `interact` generates a text area:

```python
[5]: interact(f, x='Hi Pythonistas!');
```
```
interactive(children=(Text(value='Hi Pythonistas!', description='x'), Output()), _dom_classes=('widget-interact',))
```

### Decorator

`interact` can also be used as a decorator. This way you can define a function and interact with it in a single setting. As the following example shows, `interact` also works with functions that have multiple arguments:

```python
[6]: @interact(x=True, y=1.0)
    def g(x, y):
        return (x, y)
```
```
interactive(children=(Checkbox(value=True, description='x'), FloatSlider(value=1.0, description='y', max=3.0, ...),), _dom_classes=('widget-interact',))
```

### Widget list

**1:** `import ipywidgets as widgets`

### Numeric widgets

There are a variety of IPython widgets that are designed to display numeric values. The integer widgets have a similar naming scheme as their counterparts with floating point numbers. You can find the respective integer equivalent by replacing `Float` with `Int` in the widget name.
IntSlider

```python
widgets.IntSlider(
    value=7,
    min=0,
    max=10,
    step=1,
    description='Test:',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='d'
)
```

IntSlider(value=7, continuous_update=False, description='Test:', max=10)

FloatSlider

```python
widgets.FloatSlider(
    value=7.5,
    min=0,
    max=10.0,
    step=0.1,
    description='Test:',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.1f',
)
```

FloatSlider(value=7.5, continuous_update=False, description='Test:', max=10.0, readout_format='.1f')

Sliders can also be displayed vertically.

```python
widgets.FloatSlider(
    value=7.5,
    min=0,
    max=10.0,
    step=0.1,
    description='Test:',
    disabled=False,
    continuous_update=False,
    orientation='vertical',
    readout=True,
    readout_format='.1f',
)
```

FloatSlider(value=7.5, continuous_update=False, description='Test:', max=10.0, orientation='vertical', readout...


**FloatLogSlider**

The `FloatLogSlider` has a scale that makes it easy to use a slider for a wide range of positive numbers. `min` and `max` refer to the minimum and maximum exponents of the base and the `value` refers to the actual value of the slider.

```python
[5]: widgets.FloatLogSlider(
    value=10,
    base=10,
    min=-10, # max exponent of base
    max=10, # min exponent of base
    step=0.2, # exponent step
    description='Log Slider'
)
FloatLogSlider(value=10.0, description='Log Slider', max=10.0, min=-10.0, step=0.2)
```

**IntRangeSlider**

```python
[6]: widgets.IntRangeSlider(
    value=[5, 7],
    min=0,
    max=10,
    step=1,
    description='Test:',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='d',
)
IntRangeSlider(value=(5, 7), continuous_update=False, description='Test:', max=10)
```

**FloatRangeSlider**

```python
[7]: widgets.FloatRangeSlider(
    value=[5, 7.5],
    min=0,
    max=10.0,
    step=0.1,
    description='Test:',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.1f',
)
FloatRangeSlider(value=(5.0, 7.5), continuous_update=False, description='Test:', max=10.0, readout_format='.1f...
**IntProgress**

```python
[8]: widgets.IntProgress(
    value=7,
    min=0,
    max=10,
    step=1,
    description='Loading:',
    bar_style='', # 'success', 'info', 'warning', 'danger' or ''
    orientation='horizontal'
)
```

```python
IntProgress(value=7, description='Loading:', max=10)
```

**FloatProgress**

```python
[9]: widgets.FloatProgress(
    value=7.5,
    min=0,
    max=10.0,
    step=0.1,
    description='Loading:',
    bar_style='info',
    orientation='horizontal'
)
```

```python
FloatProgress(value=7.5, bar_style='info', description='Loading:', max=10.0)
```

The numerical text boxes that impose some limit on the data (range, integer-only) impose that restriction when the user presses enter.

**BoundedIntText**

```python
[10]: widgets.BoundedIntText(
    value=7,
    min=0,
    max=10,
    step=1,
    description='Text:',
    disabled=False
)
```

```python
BoundedIntText(value=7, description='Text:', max=10)
```
BoundedFloatText

```python
[11]: widgets.BoundedFloatText(
    value=7.5,
    min=0,
    max=10.0,
    step=0.1,
    description='Text:',
    disabled=False
)
```

BoundedFloatText(value=7.5, description='Text:', max=10.0, step=0.1)

IntText

```python
[12]: widgets.IntText(
    value=7,
    description='Any:',
    disabled=False
)
```

IntText(value=7, description='Any:')

FloatText

```python
[13]: widgets.FloatText(
    value=7.5,
    description='Any:',
    disabled=False
)
```

FloatText(value=7.5, description='Any:')

Boolean widgets

There are three widgets that are designed to display Boolean values.

ToggleButton

```python
[14]: widgets.ToggleButton(
    value=False,
    description='Click me',
    disabled=False,
    button_style='', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Description',
    icon='check'
)
```

3.2. Jupyter
ToggleButton(value=False, description='Click me', icon='check', tooltip='Description')

Checkbox

[15]: widgets.Checkbox(
    value=False,
    description='Check me',
    disabled=False
)

Checkbox(value=False, description='Check me')

Valid

The Valid widget offers a read-only display.

[16]: widgets.Valid(
    value=False,
    description='Valid!',
)

Valid(value=False, description='Valid!')

Selection widgets

There are several widgets for selecting single values and two for multiple values. All inherit from the same base class.

Dropdown

[17]: widgets.Dropdown(
    options=['1', '2', '3'],
    value='2',
    description='Number:',
    disabled=False,
)

Dropdown(description='Number:', index=1, options=('1', '2', '3'), value='2')

RadioButtons

[18]: widgets.RadioButtons(
    options=['pepperoni', 'pineapple', 'anchovies'],
    # value='pineapple',
    description='Pizza topping:',
    disabled=False
)

RadioButtons(description='Pizza topping:', options=('pepperoni', 'pineapple', 'anchovies'), value='pepperoni')

```python
Select

[19]: widgets.Select(
    options=['Linux', 'Windows', 'OSX'],
    value='OSX',
    # rows=10,
    description='OS:',
    disabled=False
)
```

Select(description='OS:', index=2, options=('Linux', 'Windows', 'OSX'), value='OSX')

SelectionSlider

```python
[20]: widgets.SelectionSlider(
    options=['scrambled', 'sunny side up', 'poached', 'over easy'],
    value='sunny side up',
    description='I like my eggs ...',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True
)
```

SelectionSlider(continuous_update=False, description='I like my eggs ...', index=1, options=('scrambled', 'sunny side up')

SelectionRangeSlider

index is a tuple of minimum and maximum values.

```python
[21]: import datetime
dates = [datetime.date(2015,i,1) for i in range(1,13)]
options = [(i.strftime('%b'), i) for i in dates]
widgets.SelectionRangeSlider(
    options=options,
    index=(0,11),
    description='Months (2015)',
    disabled=False)
```

SelectionRangeSlider(description='Months (2015)', index=(0, 11), options=((Jan', 0)), dates=datetime.date(2015, 1, 1)),...
**ToggleButtons**

```python
[22]: widgets.ToggleButtons(
    options=['Slow', 'Regular', 'Fast'],
    description='Speed:',
    disabled=False,
    button_style='',
    # 'success', 'info', 'warning', 'danger' or ''
    tooltips=['Description of slow', 'Description of regular', 'Description of fast'],
    # icons=['check'] * 3
)
```

ToggleButtons(description='Speed:', options=('Slow', 'Regular', 'Fast'), tooltips=('Description of slow', 'Des...'))

**SelectMultiple**

Several values can be selected by holding down the shift and/or ctrl (or command) keys and clicking the mouse or arrow keys.

```python
[23]: widgets.SelectMultiple(
    options=['Apples', 'Oranges', 'Pears'],
    value=['Oranges'],
    #rows=10,
    description='Fruits',
    disabled=False
)
```

SelectMultiple(description='Fruits', index=(1,), options=('Apples', 'Oranges', 'Pears'), value=('Oranges',))

**String-Widgets**

There are several widgets that can be used to display strings. The widgets Text and Textarea accept input; the widgets HTML and HTMLMath display a string as HTML (HTMLMath also renders mathematical formulas).

**Text**

```python
[24]: widgets.Text(
    value='Hello World',
    placeholder='Type something',
    description='String:',
    disabled=False
)
```

Text(value='Hello World', description='String:', placeholder='Type something')
Textarea

```python
[25]: widgets.Textarea(
    value='Hello World',
    placeholder='Type something',
    description='String:',
    disabled=False
)

Textarea(value='Hello World', description='String:', placeholder='Type something')
```

Label

The Label widget is useful for custom descriptions that are similar in style to the built-in descriptions.

```python
[26]: widgets.HBox([widgets.Label(value="The $m$ in $E=mc^2$:"), widgets.FloatSlider()])

HBox(children=(Label(value='The $m$ in $E=mc^2$:'), FloatSlider(value=0.0)))
```

HTML

```python
[27]: widgets.HTML(
    value="Hello <b>World</b>",
    placeholder='Some HTML',
    description='Some HTML',
)

HTML(value='Hello <b>World</b>', description='Some HTML', placeholder='Some HTML')
```

HTML Math

```python
[28]: widgets.HTMLMath(
    value=r"Some math and <i>HTML</i>: $x^2$ and $\frac{x+1}{x-1}$",
    placeholder='Some HTML',
    description='Some HTML',
)

HTMLMath(value='Some math and <i>HTML</i>: \(x^2\) and $$\frac{x+1}{x-1}$$', description='Some HTML', placeholder='Some HTML')
```

Image

```python
[29]: file = open("smiley.gif", "rb")
image = file.read()
widgets.Image(
    value=image,
    format='gif',
    width=128,
)
```

(continues on next page)
**Button**

```python
[30]: widgets.Button(
    description='Click me',
    disabled=False,
    button_style='success',
    tooltip='Click me',
    icon='check'
)
```

```python
Button(description='Click me', icon='check', style=ButtonStyle(), tooltip='Click me')
```

**Output**

The `Output` widget can record and display `stdout`, `stderr` and `IPython.display`. You can attach the output to an output widget or delete it programmatically.

```python
[31]: out = widgets.Output(layout={'border': '1px solid black'})

[32]: with out:
    for i in range(5):
        print(i, 'Hello world!')

[33]: from IPython.display import YouTubeVideo
   with out:
     display(YouTubeVideo('eWzY2nGfkXk'))

[34]: out
   Output(layout=Layout(border='1px solid black'))

[35]: out.clear_output()

Wir können Ausgaben auch direkt anhängen mit den Methoden `append_stdout`, `append_stderr` oder `append_display_data`.

```python
[36]: out = widgets.Output(layout={'border': '1px solid black'})
   out.append_stdout('Output appended with append_stdout')
   out.append_display_data(YouTubeVideo('eWzY2nGfkXk'))
   out
   Output(layout=Layout(border='1px solid black'), outputs=([{'output_type': 'stream', 'name': 'stdout', 'text': '...'}]))
```

You can find detailed documentation in `Output widgets`. 

---

**Jupyter Tutorial, Release 0.8.0**

(continued from previous page)
Play/Animation-Widget

The Play widget is useful for running animations that you want to run at a specific speed. In the following example, a slider is linked to the player.

```python
[37]: play = widgets.Play(  
    # interval=10,  
    value=50,  
    min=0,  
    max=100,  
    step=1,  
    description="Press play",  
    disabled=False  
  )
slider = widgets.IntSlider()
widgets.jslink((play, 'value'), (slider, 'value'))
widgets.HBox([play, slider])
```

DatePicker

The date picker widget works in Chrome, Firefox and IE Edge, but not currently in Safari because it does not support input type="date".

```python
[38]: widgets.DatePicker(  
    description='Pick a Date',  
    disabled=False  
  )
DatePicker(value=None, description='Pick a Date')
```

Color picker

```python
[39]: widgets.ColorPicker(  
    concise=False,  
    description='Pick a color',  
    value='blue',  
    disabled=False  
  )
ColorPicker(value='blue', description='Pick a color')
```
Controller

Controller enables the use of a game controller as an input device.

```python
[40]: widgets.Controller(
    index=0,
)
Controller()
```

Container/layout widgets

These widgets are used to store other widgets called children.

Box

```python
[41]: items = [widgets.Label(str(i)) for i in range(4)]
widgets.Box(items)
Box(children=(Label(value='0'), Label(value='1'), Label(value='2'), Label(value='3')))
```

HBox

```python
[42]: items = [widgets.Label(str(i)) for i in range(4)]
widgets.HBox(items)
HBox(children=(Label(value='0'), Label(value='1'), Label(value='2'), Label(value='3')))
```

VBox

```python
[43]: items = [widgets.Label(str(i)) for i in range(4)]
left_box = widgets.VBox([items[0], items[1]])
right_box = widgets.VBox([items[2], items[3]])
widgets.HBox([left_box, right_box])
HBox(children=(VBox(children=(Label(value='0'), Label(value='1'))),
...
VBox(children=(Label(value='2'), Label(value='3'))))
```

Accordion

```python
accordion.set_title(0, 'Slider')
accordion.set_title(1, 'Text')
accordion
Accordion(children=(IntSlider(value=0), Text(value='')), _titles={0: 'Slider', 1: 'Text'})
```
**Tabs**

In this example the children are set after the tab is created. Titles for the tabs are set in the same way they are for Accordion.

```python
[45]: tab_contents = ['P0', 'P1', 'P2', 'P3', 'P4']
children = [widgets.Text(description=name) for name in tab_contents]
for i in range(len(children)):
    tab.set_title(i, str(i))
for name in tab_contents:
    children = [widgets.Text(description=name) for name in tab_contents]
    tab = widgets.Tab(children)
    for i in range(len(children)):
        tab.set_title(i, str(i))
    tab
```

**Accordion and Tab**

Unlike the other widgets previously described, the container widgets Accordion and Tab update their `selected_index` attribute when the user changes the accordion or tab; In addition to user input, the `selected_index` can also be set programmatically.

If `selected_index = None` is chosen, all accordions will be closed or all tabs will be deselected.

In the following notebook cells the value of `selected_index` of the tab and/or accordion is displayed.

```python
[46]: tab.selected_index = 3

[47]: accordion.selected_index = None
```

**Nesting tabs and accordions**

Tabs and accordions can be nested as deeply as you want. The following example shows some tabs with an accordion as children.

```python
[48]: tab_nest = widgets.Tab()
tab_nest.children = [accordion, accordion]
tab_nest.set_title(0, 'An accordion')
tab_nest.set_title(1, 'Copy of the accordion')
for name in tab_contents:
    children = [widgets.Text(description=name) for name in tab_contents]
    tab = widgets.Tab(children)
    for i in range(len(children)):
        tab.set_title(i, str(i))
    tab
```
Widget events

Special events

[1]: `from __future__ import print_function`

Button cannot be used to represent a data type, but only for `on_click`. With the `print` function the docstring of `on_click` can be output.

[2]: `import ipywidgets as widgets`

```python
print(widgets.Button.on_click.__doc__)
```

Register a callback to execute when the button is clicked.

The callback will be called with one argument, the clicked button widget instance.

Parameters

`---------`

`remove`: bool (optional)

Set to true to remove the callback from the list of callbacks.

Examples

Button clicks are stateless, i.e. they transfer messages from the front end to the back end. If you use the `on_click` method, a button will be displayed that will print the message as soon as it is clicked.

[3]: `from IPython.display import display`

```python
button = widgets.Button(description="Click Me!")
display(button)

def on_button_clicked(b):
    print("Button clicked.")

button.on_click(on_button_clicked)

Button(description='Click Me!', style=ButtonStyle())
```

Traitlet events

Widget properties are IPython traitlets. To make changes, the `observe` method of the widget can be used to register a callback. You can see the docstring for `observe` below.

You can find more information at Traitlet events.

[4]: `print(widgets.Widget.observe.__doc__)`

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.
Parameters

----------

handler : callable
A callable that is called when a trait changes. Its
signature should be `handler(change)`, where `change` is a
dictionary. The change dictionary at least holds a 'type' key.
* `type` : the type of notification.
Other keys may be passed depending on the value of 'type'. In the
case where type is 'change', we also have the following keys:
* `owner` : the HasTraits instance
* `old` : the old value of the modified trait attribute
* `new` : the new value of the modified trait attribute
* `name` : the name of the modified trait attribute.

names : list, str, All
If names is All, the handler will apply to all traits. If a list
of str, handler will apply to all names in the list. If a
str, the handler will apply just to that name.
type : str, All (default: 'change')
The type of notification to filter by. If equal to All, then all
notifications are passed to the observe handler.

Linking widgets

To link widget attributes, you can simply link them together.

**Linking traitlet attributes in the kernel**

```
[5]: caption = widgets.Label(value='The values of slider1 and slider2 are synchronized')
sliders1, slider2 = widgets.IntSlider(description='Slider 1'),
    widgets.IntSlider(description='Slider 2')
l = widgets.link((sliders1, 'value'), (slider2, 'value'))  
display(caption, sliders1, slider2)
Label(value='The values of slider1 and slider2 are synchronized')
IntSlider(value=0, description='Slider 1')
IntSlider(value=0, description='Slider 2')
```

**Linking widgets attributes on the client side**

There might be a delay while synchronizing Traitlet attributes due to communication with the server. However, you
can also link the widget attributes to the link widgets directly in the browser. The Javascript links with `jslink` are
retained even if widgets are embedded in HTML websites without a kernel.

```
[6]: caption = widgets.Label(value='The values of range1 and range2 are synchronized')
rangel1, range2 = widgets.IntSlider(description='Range 1'),
    widgets.IntSlider(description='Range 2')
```
Continuous updates

Some widgets offer a `continuous_update` attribute with the ability to continuously update values. In the following example we can see that the delayed controls only transmit their value after the user has dragged the slider or sent the text field. The continuous slides transfer their values continuously as soon as they are changed.

```python
[a = widgets.IntSlider(description="Delayed", continuous_update=False)
 b = widgets.IntText(description="Delayed", continuous_update=False)
 c = widgets.IntSlider(description="Continuous", continuous_update=True)
 d = widgets.IntText(description="Continuous", continuous_update=True)

widgets.link((a, 'value'), (b, 'value'))
widgets.link((a, 'value'), (c, 'value'))
widgets.link((a, 'value'), (d, 'value'))
widgets.VBox([a,b,c,d])
```

Custom widget

The widget framework is based on the Comms framework, which enables the kernel to send and receive JSON to the front end. In order to create a custom widget, the widget must be defined both in the browser and in the Python kernel.

For more information on the Comms framework, see the Low Level Widget Tutorial.
Python kernel

DOMWidget

To define a widget, it must inherit from the Widget or DOMWidget base class. If the widget is to be displayed in the Jupyter notebook, your widget should inherit from DOMWidget. The DOMWidget class itself inherits from the Widget class.

_view_name

By adopting DOMWidget, the widget framework is not informed which front-end widget should be linked to the back-end widget. Instead, you have to specify this yourself using one of the following attributes:

- _view_name
- _view_module
- _view_module_version

and if applicable

- _model_name
- _model_module

```
[1]: import ipywidgets as widgets
from traitlets import Unicode, validate

class HelloWidget(widgets.DOMWidget):
    _view_name = Unicode('HelloView').tag(sync=True)
    _view_module = Unicode('hello').tag(sync=True)
    _view_module_version = Unicode('0.1.0').tag(sync=True)
```

sync=True-Traitlets

Traitlets is a framework with which Python classes can have attributes with type checking, dynamically calculated default values and callbacks when changed. The sync=True keyword argument tells the widget framework to synchronise the value with the browser; without it, the browser would not learn anything about _view_name or _view_module.

Frontend (JavaScript)

Models and Views

The front end of the IPython widget framework depends heavily on Backbone.js. Backbone.js is an Model View Controller (MVC) framework that automatically synchronises widgets defined in the backend with generic Backbone.js models in the frontend: the previously defined _view_name characteristic is used by the widget framework to display the corresponding Backbone.js-View and link it to the model.
Import @jupyter-widgets/base

First you have to use the @jupyter-widgets/base module with the define method of RequireJS.

> [2]: define('hello', ['@jupyter-widgets/base'], function(widgets) {
> });
> <IPython.core.display.Javascript object>

Define view

Next we define the widget view class and we inherit from DOMWidgetView with the .extend method.

> [3]: require.undef('hello');
> define('hello', ['@jupyter-widgets/base'], function(widgets) {
>     // Define the HelloView
>     var HelloView = widgets.DOMWidgetView.extend({
>         render: function() {
>             // Render the view.
>             this.el.textContent = 'Hello World!';
>         },
>         return { HelloView: HelloView
>     }
> });
> <IPython.core.display.Javascript object>

render method

Finally, we still have to override the basic render method to define a custom rendering logic. A handle to the standard DOM element of the widget can be called with this.el. The el property is the DOM element associated with the view.

> [4]: require.undef('hello');
> define('hello', ['@jupyter-widgets/base'], function(widgets) {
>     // Render the view.
>     this.el.textContent = 'Hello World!';
> });
> <IPython.core.display.Javascript object>
Test

The widget can now be displayed like any other widget with

```
[5]: HelloWidget()
HelloWidget()
```

Stateful widget

There’s not much you can do with the example above. To change this, you have to make the widget stateful. Instead of a static Hello World! Message, a string specified by the backend should be displayed. To do this, a new traitlet is first added. Use the name of value here to stay consistent with the rest of the widget framework and to allow your widget to be used with interaction.

Create Jupyter widgets from a template

A Cookiecutter is available with widget-cookiecutter. It contains an implementation for a placeholder widget Hello World. It also makes it easier for you to pack and distribute your Jupyter widgets.

ipywidgets libraries

Popular widget libraries are

- **qplot** 2-D plotting library for Jupyter notebooks
  - **bqplot**

- **ipycanvas** Interactive canvas elements in Jupyter notebooks

  **ipycanvas**

  provides the Web-Canvas-API. However, there are some differences:

  - The Canvas widget exposes the CanvasRenderingContext2D API directly
  - The entire API is written in snake_case instead of camelCase, so for example canvas.fillStyle = ‘red’ written in Python becomes canvas.fill_style = ‘red’ in JavaScript.

Installation

```
$ pipenv install ipycanvas
Installing ipycanvas...
...
Creating canvas elements

Before we can start creating canvas elements, first a note about the canvas grid. The origin of a grid is in the upper left corner at the coordinate \((0, 0)\). All elements are placed relative to this origin.

There are four methods of drawing rectangles:

- `fill_rect(x, y, width, height=None)` draws a filled rectangle
- `stroke_rect(x, y, width, height=None)` draws a rectangular outline
- `fill_rects(x, y, width, height=None)` draws filled rectangles
- `stroke_rects(x, y, width, height=None)` draws rectangular outlines

With `height=None`, the same value is used as with `width`.

For `*_rects`, `x`, `y`, `width` and `height` are integers, lists of integers or numpy arrays.

```
[1]: from ipycanvas import Canvas
    canvas = Canvas(size=(120, 100))
    canvas.fill_style = 'lime'
    canvas.stroke_style = 'green'
    canvas.fill_rect(10, 20, 100, 50)
    canvas.stroke_rect(10, 20, 100, 50)
    canvas
    Canvas(layout=Layout(height='100px', width='120px'), size=(120, 100))

[2]: from ipycanvas import MultiCanvas
    # Create a multi-layer canvas with 2 layers
    multi_canvas = MultiCanvas(2, size=(165, 115))
    multi_canvas[0] # Access first layer (background)
    multi_canvas[0].fill_style = 'lime'
    multi_canvas[0].stroke_style = 'green'
    multi_canvas[0].fill_rect(10, 20, 100, 50)
    multi_canvas[0].stroke_rect(10, 20, 100, 50)
    multi_canvas[1] # Access last layer
    multi_canvas[1].fill_style = 'red'
    multi_canvas[1].stroke_style = 'brown'
    multi_canvas[1].fill_rect(55, 45, 100, 50)
    multi_canvas[1].stroke_rect(55, 45, 100, 50)
    multi_canvas
    MultiCanvas(layout=Layout(height='115px', width='165px'))

[3]: import numpy as np
    from ipycanvas import Canvas
    n_particles = 75_000
```

(continues on next page)
x = np.array(np.random.rayleigh(350, n_particles), dtype=np.int32)
y = np.array(np.random.rayleigh(150, n_particles), dtype=np.int32)
size = np.random.randint(1, 3, n_particles)

canvas = Canvas(size=(1000, 500))
canvas.fill_style = 'green'
canvas.fill_rects(x, y, size)
canvas

Canvas(layout=Layout(height='500px', width='1000px'), size=(1000, 500))

Since Canvas is an ipywidget, it can

• appear several times in a notebook
• change the attributes
• Link changed attributes to other widget attributes

Delete canvas

[4]: from ipycanvas import Canvas
canvas = Canvas(size=(120, 100))
# Perform some drawings...
canvas.clear()

[5]: from ipycanvas import Canvas
canvas = Canvas(size=(165, 115))
canvas.fill_style = 'lime'
canvas.stroke_style = 'brown'
canvas.fill_rect(10, 20, 100, 50)
canvas.clear_rect(52, 42, 100, 50)
canvas.stroke_rect(55, 45, 100, 50)
canvas

Canvas(layout=Layout(height='115px', width='165px'), size=(165, 115))
Shapes

The available drawing commands are:

- `move_to(x, y)`:
- `line_to(x, y)`:
- `arc(x, y, radius, start_angle, end_angle, anticlockwise=False)`:
- `arc_to(x1, y1, x2, y2, radius)`:
- `quadratic_curve_to(cp1x, cp1y, x, y)`:
- `bezier_curve_to(cp1x, cp1y, cp2x, cp2y, x, y)`:
- `rect(x, y, width, height)`:

Draw circles

There are four different ways to draw circles:

- `fill_arc(x, y, radius, start_angle, end_angle, anticlockwise=False)`
- `stroke_arc(x, y, radius, start_angle, end_angle, anticlockwise=False)`
- `fill_arcs(x, y, radius, start_angle, end_angle, anticlockwise=False)`
- `stroke_arcs(x, y, radius, start_angle, end_angle, anticlockwise=False)`

With `_arcs`, `x`, `y` and `radius` are NumPy arrays, lists or scalar values.

```
[6]: from math import pi

from ipycanvas import Canvas

canvas = Canvas(size=(200, 200))

canvas.fill_style = 'red'
canvas.stroke_style = 'green'

canvas.fill_arc(60, 60, 50, 0, pi)
canvas.stroke_arc(60, 60, 40, 0, 2 * pi)

canvas
```

Canvas(layout=Layout(height='200px', width='200px'), size=(200, 200))

Drawing paths

A path is a list of points connected by line segments that can be different shapes, straight or curved, closed or open, different widths and colors. The following functions are available:

```
begin_path()
close_path() adds a straight line to the path leading to the beginning of the
→ current path
stroke() draws the shape along the contour
fill (rule) draws the shape using a fill within the path
```
• `begin_path()` creates a new path
• `close_path()` adds a straight line to the path leading to the beginning of the current path
• `stroke()` draws the shape along the contour
• `fill(rule)` draws the shape using a fill within the path

```
[7]: from ipycanvas import Canvas
    canvas = Canvas(size=(100, 100))
    # Draw simple triangle shape
    canvas.begin_path()
    canvas.move_to(75, 50)
    canvas.line_to(100, 75)
    canvas.line_to(100, 25)
    canvas.fill()
    canvas
Canvas(layout=Layout(height='100px', width='100px'), size=(100, 100))
```

Examples

arc

```
[8]: from math import pi
    from ipycanvas import Canvas
    canvas = Canvas(size=(200, 200))
    # Draw smiley face
    canvas.begin_path()
    canvas.arc(75, 75, 50, 0, pi * 2, True) # Outer circle
    canvas.move_to(110, 75)
    canvas.arc(75, 75, 35, 0, pi, False) # Mouth (clockwise)
    canvas.move_to(65, 65)
    canvas.arc(60, 65, 5, 0, pi * 2, True) # Left eye
    canvas.move_to(95, 65)
    canvas.arc(90, 65, 5, 0, pi * 2, True) # Right eye
    canvas.stroke()
    canvas
Canvas(layout=Layout(height='200px', width='200px'), size=(200, 200))
```
```python
from ipycanvas import Canvas

canvas = Canvas(size=(200, 200))

# Cubic curves example
canvas.begin_path()
canvas.move_to(75, 40)
canvas.bezier_curve_to(75, 37, 70, 25, 50, 25)
canvas.bezier_curve_to(20, 25, 20, 62.5, 20, 62.5)
canvas.bezier_curve_to(20, 80, 40, 102, 75, 120)
canvas.bezier_curve_to(110, 102, 130, 80, 130, 62.5)
canvas.bezier_curve_to(130, 62.5, 130, 25, 100, 25)
canvas.bezier_curve_to(85, 25, 75, 37, 75, 40)
canvas.fill()

canvas
Canvas(layout=Layout(height='200px', width='200px'), size=(200, 200))
```

**Styles and colors**

**Colors**

Canvas has two color attributes, one for strokes and one for areas; the transparency can also be changed.

- **stroke_style** expects a valid HTML color. The default is black.
- **fill_style** expects a valid HTML color. The default is black.
- **global_alpha** indicates the transparency. The default is 1.0.

**Lines**

**Line style**

Lines can be described by the following attributes:

- **line_width**
- **line_cap**
- **line_join**
- **miter_limit**
- **get_line_dash()**
- **set_line_dash(segments)**
- **line_dash_offset**
Line width

```python
[10]: from ipycanvas import Canvas

canvas = Canvas(size=(400, 280))
canvas.scale(2)

for i in range(10):
    width = 1 + i
    x = 5 + i * 20
    canvas.line_width = width
    canvas.fill_text(str(width), x - 5, 15)

    canvas.begin_path()
    canvas.move_to(x, 20)
    canvas.line_to(x, 140)
    canvas.stroke()
```

Canvas(layout=Layout(height='280px', width='400px'), size=(400, 280))

Line end

```python
[11]: from ipycanvas import Canvas

canvas = Canvas(size=(320, 360))

# Possible line_cap values
line_caps = ['butt', 'round', 'square']

canvas.scale(2)

# Draw guides
canvas.stroke_style = '#09f'
canvas.begin_path()
canvas.move_to(10, 30)
canvas.line_to(140, 30)
canvas.move_to(10, 140)
canvas.line_to(140, 140)
canvas.stroke()

# Draw lines
canvas.stroke_style = 'black'
canvas.font = '15px serif'

for i in range(len(line_caps)):
    line_cap = line_caps[i]
    x = 25 + i * 50
    canvas.fill_text(line_cap, x - 15, 15)
```

(continues on next page)
```python
canvas.line_width = 15
canvas.line_cap = line_cap
canvas.begin_path()
canvas.move_to(x, 30)
canvas.line_to(x, 140)
canvas.stroke()
```

```python
canvas = Canvas(layout=Layout(height='360px', width='320px'), size=(320, 360))
```

**Line connection**

defines the appearance of the corners where lines meet.

```python
from ipycanvas import Canvas

canvas = Canvas(size=(320, 360))

# Possible line_join values
line_joins = ['round', 'bevel', 'miter']

min_y = 40
max_y = 80
spacing = 45

canvas.line_width = 10
canvas.scale(2)
for i in range(len(line_joins)):
    line_join = line_joins[i]

    y1 = min_y + i * spacing
    y2 = max_y + i * spacing

    canvas.line_join = line_join

    canvas.fill_text(line_join, 60, y1 - 10)

    canvas.begin_path()
    canvas.move_to(-5, y1)
canvas.line_to(35, y2)
canvas.line_to(75, y1)
canvas.line_to(115, y2)
canvas.line_to(155, y1)
canvas.stroke()
```

```python
canvas = Canvas(layout=Layout(height='360px', width='320px'), size=(320, 360))
```
Line pattern

[13]: from ipycanvas import Canvas

    canvas = Canvas(size=(400, 280))
    canvas.scale(2)

    line_dashes = [
        [5, 10],
        [10, 5],
        [5, 10, 20],
        [10, 20],
        [20, 10],
        [20, 20]
    ]

    canvas.line_width = 2

    for i in range(len(line_dashes)):
        x = 5 + i * 20

        canvas.set_line_dash(line_dashes[i])
        canvas.begin_path()
        canvas.move_to(x, 0)
        canvas.line_to(x, 140)
        canvas.stroke()

Canvas(layout=Layout(height='280px', width='400px'), size=(400, 280))


Text

There are two methods of designing text:

• fill_text(text, x, y, max_width=None)
• stroke_text(text, x, y, max_width=None)

[14]: from ipycanvas import Canvas

    canvas = Canvas(size=(400, 50))
    canvas.font = '32px serif'
    canvas.fill_text('Drawing from Python is cool!', 10, 32)

Canvas(layout=Layout(height='50px', width='400px'), size=(400, 50))

[15]: from ipycanvas import Canvas

    canvas = Canvas(size=(400, 50))
    canvas.font = '32px serif'

(continues on next page)
font indicates the current text style. The default value is “12px sans-serif”. text_align specifies the text alignment. Possible values are “start”, “end”, “left”, “right” or “center”. text_baseline indicates the alignment with the baseline. Possible values are “top”, “hanging”, “middle”, “alphabetic”, “ideographic” and “bottom”. The default value is “alphabetic”. direction indicates the text direction. Possible values are “ltr”, “rtl”, “inherit”. The default value is “inherit”.

Of course, fill_style and stroke_style can also be used to color the text.

- font indicates the current text style. The default value is "12px sans-serif".
- text_align specifies the text alignment. Possible values are "start", "end", "left", "right" or "center".
- text_baseline indicates the alignment with the baseline. Possible values are "top", "hanging", "middle", "alphabetic", "ideographic" and "bottom". The default value is "alphabetic".
- direction indicates the text direction. Possible values are "ltr", "rtl", "inherit". The default value is "inherit".

Of course, fill_style and stroke_style can also be used to color the text.

### Images

... from a Numpy array

With put_image_data(image_data, dx, dy) an image can be displayed, where image_data specifies a Numpy array and dx and dy the upper left corner of the image.

```python
[16]: import numpy as np

from ipycanvas import Canvas

x = np.linspace(-1, 1, 600)
y = np.linspace(-1, 1, 600)

x_grid, y_grid = np.meshgrid(x, y)

blue_channel = np.array(np.sin(x_grid**2 + y_grid**2) * 255, dtype=np.int32)
red_channel = np.zeros_like(blue_channel) + 200
green_channel = np.zeros_like(blue_channel) + 50

image_data = np.stack((red_channel, blue_channel, green_channel), axis=2)

canvas = Canvas(size=(image_data.shape[0], image_data.shape[1]))
canvas.put_image_data(image_data, 0, 0)

canvas

Canvas(layout=Layout(height='600px', width='600px'), size=(600, 600))
```
Status

The status can be specified with two values:

- `save()` saves the status of the canvas element.
- `restore()` restores the last saved status of the canvas element. This method can be called as often as required.

Transformations

- `translate(x, y)` moves the canvas element
- `rotate(angle)` rotates the canvas element clockwise
- `scale(x, y=None)` scales the canvas element

See also

- API reference
- `pythreejs` Jupyter Three.js bridge
  - `pythreejs`
- `ipyvolume` IPyvolume is a Python library for visualizing 3D volumes and glyphs (e.g. 3D scatter plots).
  - `ipyvolume`
- `ipyleaflet` Jupyter-Leaflet.js bridge
  - `ipyleaflet`
- `ipywebrtc` WebRTC and MediaStream API for Jupyter notebooks

`ipywebrtc`

`ipywebrtc` provides WebRTC and the MediaStream-API in Jupyter notebooks. This allows e.g. to create screenshots from a MediaStream and analyse them further with `skimage`. With `ipywebrtc` you can not only read video, image, audio and widget data but also record stream objects. It even provides a simple chat function.

Installation

`ipywebrtc` is installed in both the kernel and the Jupyter environment:

```
$ pipenv install ipywebrtc
```
Examples

Example VideoStream

[1]: from ipywebrtc import VideoStream, VideoRecorder

[2]: !cd ../../../../data/

Saving information to 'Big.Buck.Bunny.mp4.dvc'.

To track the changes with git, run:

    git add Big.Buck.Bunny.mp4.dvc

[3]: video = VideoStream.from_url('../../../../data/Big.Buck.Bunny.mp4')
video
VideoStream(video=Video(value=b'../../../../data/Big.Buck.Bunny.mp4', format='url'))

Record

A record button can be created with MediaRecorder.record, for videos with:

[4]: recorder = VideoRecorder(stream=video)
recorder
VideoRecorder(stream=VideoStream(video=Video(value=b'../../../../data/Big.Buck. ˓→Bunny.mp4', format='url')), vid...
geometry=SphereGeometry(radius=1),
    material=MeshLambertMaterial(color='red'),
    position=[2, 1, 0]
)
c = PerspectiveCamera(
    position=[0, 5, 5], up=[0, 1, 0],
    children=[DirectionalLight(color='white', position=[3, 5, 1], intensity=0.5)]
)
scene = Scene(children=[ball, c, AmbientLight(color='#777777')])
renderer = Renderer(
    camera=c,
    scene=scene,
    controls=[OrbitControls(controlling=c)]
)
renderer

Renderer(camera=PerspectiveCamera(children=(DirectionalLight(color='white',
                                          intensity=0.5, position=(3.0, 5.0,...

The following `webgl_stream` is updated after something changes in the scene above. You can do this by moving the ball with the mouse.

```
[8]: webgl_stream = WidgetStream(widget=renderer)

webgl_stream

WidgetStream(widget=Renderer(camera=PerspectiveCamera(children=(DirectionalLight(color='white',
                                          intensity=0.5,...

Alternatively, you can also use a slider:

```
[9]: from ipywidgets import FloatSlider

slider = FloatSlider(
    value=7.5,
    step=0.1,
    description='Test:',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='.1f',
)

slider

FloatSlider(value=7.5, continuous_update=False, description='Test:', readout_format='.1f')

ipysheet Interactive tables to use IPython widgets in tables of Jupyter notebooks.
**ipysheet**

**ipysheet** connects ipywidgets with tabular data. It basically adds two widgets: a Cell widget and a Sheet widget. There are also auxiliary functions for creating table rows and columns as well as for formatting and designing cells.

**Installation**

**ipysheet** can be easily installed with Pipenv:

```bash
$ pipenv install ipysheet
```

**Import**

```
[1]: import ipysheet
```

**Cell formatting**

```
[2]: sheet1 = ipysheet.sheet()
cell0 = ipysheet.cell(0, 0, 0, numeric_format='0.0', type='numeric')
cell1 = ipysheet.cell(1, 0, "Hello", type='text')
cell2 = ipysheet.cell(0, 1, 0.1, numeric_format='0.000', type='numeric')
cell3 = ipysheet.cell(1, 1, 15.9, numeric_format='0.00', type='numeric')
cell4 = ipysheet.cell(2, 2, "14-02-2019", date_format='DD-MM-YYYY', type='date')
```

**Examples**

**Interactive table**

```
[3]: from ipywidgets import FloatSlider, IntSlider, Image

slider = FloatSlider()
sheet2 = ipysheet.sheet()
cell1 = ipysheet.cell(0, 0, slider, style={'min-width': '122px'})
cell3 = ipysheet.cell(1, 0, 42., numeric_format='0.00')
cell_sum = ipysheet.cell(2, 0, 42., numeric_format='0.00')

@ipysheet.calculation(inputs=[(cell1, 'value'), cell3], output=cell_sum)
def calculate(a, b):
    return a + b
```

```
Numpy

```python
[4]: import numpy as np
   from ipysheet import from_array, to_array
   arr = np.random.randn(6, 10)
   sheet = from_array(arr)
   sheet

Sheet(cells=(Cell(column_end=9, column_start=0, row_end=5, row_start=0,squeeze_column=False, squeeze_row=False),)

[5]: arr = np.array([True, False, True])
   sheet = from_array(arr)
   sheet

Sheet(cells=(Cell(column_end=0, column_start=0, numeric_format=None, row_end=2, row_start=0,squeeze_row=False),)

[6]: to_array(sheet)
[6]: array([[ True],
          [False],
          [ True]])

Table search

[7]: import numpy as np
   import pandas as pd
   from ipysheet import from_dataframe
   from ipywidgets import Text, VBox, link
   df = pd.DataFrame({'A': 1.,
                      'B': pd.Timestamp('20130102'),
                      'C': pd.Series(1, index=list(range(4)), dtype='float32'),
                      'D': np.array([False, True, False, False], dtype='bool'),
                      'E': pd.Categorical(['test', 'train', 'test', 'train']),
                      'F': 'foo'})
   df.loc[[0, 2], ['B']] = np.nan

s = from_dataframe(df)
search_box = Text(description='Search:')
```

(continues on next page)
Plot editable tables

```python
[8]:

from traitlets import link
from ipywidgets import HBox
import bqplot.pyplot as plt
from ipysheet import sheet, cell, column

size = 18
scale = 100.
np.random.seed(0)
x_data = np.arange(size)
y_data = np.cumsum(np.random.randn(size) * scale)

fig = plt.figure()
axes_options = {
    'x': {'label': 'Date', 'tick_format': '%m/%d'},
    'y': {'label': 'Price', 'tick_format': '0.0f'}}

scatt = plt.scatter(x_data, y_data, colors=['red'], stroke='black')
fig.layout.width = '70%'
sheet1 = sheet(rows=size, columns=2)
x_column = column(0, x_data)
y_column = column(1, y_data)

link((scatt, 'x'), (x_column, 'value'))
link((scatt, 'y'), (y_column, 'value'))

HBox((sheet1, fig))
```

For further reading

- Interactive spreadsheets in Jupyter
- GitHub
- Docs

$qgrid$ Widget based on SlickGrid for rendering pandas DataFrames in Jupyter notebooks
qgrid

qgrid uses SlickGrid to sort, filter and manipulate DataFrames in Jupyter notebooks. In this way you can sort and filter DataFrames and edit the DataFrames by double-clicking on cells.

Installation

qgrid is installed in both the kernel and the Jupyter environment:

```
$ pipenv install qgrid
```

In the Jupyter environment, qgrid must also be activated as notebook extension:

```
$ pipenv run jupyter nbextension enable --py --sys-prefix qgrid
Enabling notebook extension qgrid/extension...
- Validating: OK
```

[1]: `import qgrid`

API

You can find the API documentation at https://qgrid.readthedocs.io/.

Alternatively, you can access the API documentation with the IPython ? Operator:

[2]: `qgrid.show_grid?`

[3]: `qgrid.set_defaults?`

[4]: `qgrid.set_grid_option?`

Examples

Examples with `show_grid` and `grid_options`

First we get data with `get_data_yahoo` and render it as usual:

[5]: `import pandas as pd
go to next page`
```python
start=pd.Timestamp('2011-01-01'),
end=pd.Timestamp('2014-01-01'),
adjust_price=True,
)
spy
```

<table>
<thead>
<tr>
<th>Date</th>
<th>High</th>
<th>Low</th>
<th>Open</th>
<th>Close</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011-01-03</td>
<td>107.046555</td>
<td>105.452601</td>
<td>106.299915</td>
<td>106.585152</td>
<td>138725200.0</td>
</tr>
<tr>
<td>2011-01-04</td>
<td>106.853615</td>
<td>106.820058</td>
<td>106.526436</td>
<td>137409700.0</td>
<td></td>
</tr>
<tr>
<td>2011-01-05</td>
<td>107.147247</td>
<td>106.190875</td>
<td>107.080132</td>
<td>133975300.0</td>
<td></td>
</tr>
<tr>
<td>2011-01-06</td>
<td>107.239505</td>
<td>107.122056</td>
<td>106.870377</td>
<td>122519000.0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2013-12-26</td>
<td>164.312317</td>
<td>163.740672</td>
<td>163.758526</td>
<td>164.222992</td>
<td>63365000.0</td>
</tr>
<tr>
<td>2013-12-27</td>
<td>164.508762</td>
<td>164.437319</td>
<td>164.214020</td>
<td>61814000.0</td>
<td></td>
</tr>
<tr>
<td>2013-12-30</td>
<td>164.365892</td>
<td>163.972884</td>
<td>164.231905</td>
<td>56857000.0</td>
<td></td>
</tr>
<tr>
<td>2013-12-31</td>
<td>164.964279</td>
<td>164.285442</td>
<td>164.410502</td>
<td>86119900.0</td>
<td></td>
</tr>
</tbody>
</table>

Adj_Ratio

<table>
<thead>
<tr>
<th>Date</th>
<th>Adj_Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011-01-03</td>
<td>0.838923</td>
</tr>
<tr>
<td>2011-01-04</td>
<td>0.838923</td>
</tr>
<tr>
<td>2011-01-05</td>
<td>0.838923</td>
</tr>
<tr>
<td>2011-01-06</td>
<td>0.838923</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2013-12-26</td>
<td>0.893196</td>
</tr>
<tr>
<td>2013-12-27</td>
<td>0.893196</td>
</tr>
<tr>
<td>2013-12-30</td>
<td>0.893196</td>
</tr>
<tr>
<td>2013-12-31</td>
<td>0.893196</td>
</tr>
</tbody>
</table>

[754 rows x 6 columns]

Now we render the DataFrame with `qgrid`. You can then quickly scroll, sort and filter through hundreds of thousands of lines. If you double click on a cell you can edit it; and the changes change the values stored in the DataFrame.

```python
[6]: qgrid.show_grid(spy)
```

QgridWidget(grid_options={'fullWidthRows': True, 'syncColumnCellResize': True, 'forceFitColumns': True, 'defaultColumnWidth': ...

While the `show_grid` function uses a number of optional parameters to configure the behavior of the table, `grid_options` allows you to pass a dict with the `grid_options` of SlickGrid. In our example below, we'll use `forceFitColumns` and `defaultColumnWidth` to improve `qgrid`'s ability to handle a large number of columns.

```python
[7]: qgrid.show_grid(spy, grid_options={'forceFitColumns': False, 'defaultColumnWidth': 200})
```

QgridWidget(grid_options={'fullWidthRows': True, 'syncColumnCellResize': True, 'forceFitColumns': False, 'default...
Example of a DataFrame with a multi-index

[8]:
```python
import qgrid
import pandas as pd
from pandas_datareader import wb

df_countries = wb.download(indicator='NY.GDP.PCAP.KD', country=['all'], start=2005, end=2008)
df_countries.columns = ['GDP per capita (constant 2005 US$)']
qugrid.show_grid(df_countries)
```

Example with interval column

[9]:
```python
import numpy as np
import pandas as pd
import qgrid

td = np.cumsum(np.random.randint(1, 15*60, 1000))
start = pd.Timestamp('2017-04-17')
df_interval = pd.DataFrame([(start + pd.Timedelta(seconds=d)) for d in td], columns=['time'])

defreq = '15Min'
start = df_interval['time'].min().floor(defreq)
end = df_interval['time'].max().ceil(defreq)
bins = pd.date_range(start, end, freq=defreq)

df_interval['time_bin'] = pd.cut(df_interval['time'], bins)
qugrid.show_grid(df_interval, show_toolbar=True)
```

Example with unnamed columns and ipywidgets.Layout

[10]:
```python
import numpy as np
import pandas as pd
import qgrid
import ipywidgets as ipyw

randn = np.random.randn

df_types = pd.DataFrame(np.random.randint(1, 14, 14))
qugrid_widget = qgrid.show_grid(df_types, show_toolbar=False)
qugrid_widget.layout = ipyw.Layout(width='20%')
qugrid_widget
```

3.2. Jupyter
Example of columns with different data types

```python
[11]: import pandas as pd
    import qgrid
    df = pd.DataFrame({'A': [1.2, 'foo', 4], 'B': [3, 4, 5]})
    df = df.set_index(pd.Index(['bar', 7, 3.2]))
    view = qgrid.show_grid(df)
    view
```

QgridWidget(grid_options={'fullWidthRows': True, 'syncColumnCellResize': True, 'forceFitColumns': True, 'defau...

Example with `nan` and `None`

```python
[12]: import pandas as pd
    import numpy as np
    import qgrid
    df = pd.DataFrame([(pd.Timestamp('2017-02-02'), None, 3.4), (np.nan, 2, 4.7), (pd.Timestamp('2017-02-03'), 3, None)])
    qgrid.show_grid(df)
```

QgridWidget(grid_options={'fullWidthRows': True, 'syncColumnCellResize': True, 'forceFitColumns': True, 'defau...

**ipydatagrid**  
ipydatagrid is a fast and versatile datagrid widget.

**ipyvuetify**  
Vuety widgets in Jupyter notebooks

ipyvuetify

ipyvuetify provides Jupyter widgets based on vuety UI components and implementing Google’s Material Design with the Vue.js-Framework framework.

**Installation**

```bash
$ pipenv install ipyvuetify
Installing ipyvuetify...
...
$ pipenv run jupyter nbextension enable --py --sys-prefix ipyvuetify
Enabling notebook extension jupyter-vuetify/extension...
    - Validating: OK
```
Examples

Imports

[1]: import ipywidgets
import ipyvuetify as v
from threading import Timer

from traits import (Any, Unicode, List)

Menu

[2]: def on_menu_click(widget, event, data):
    if len(layout.children) == 1:
        layout.children = layout.children + [info]
        info.children = [f'Item {items.index(widget)+1} clicked']

items = [v.ListItem(children=[
    v.ListItemTitle(children=[
        f'Item {i}']
    )
    for i in range(1, 5)]
]

for item in items:
    item.on_event('click', on_menu_click)

menu = v.Menu(offset_y=True,
              v_slots=[{
                'name': 'activator',
                'variable': 'menuData',
                'children': v.Btn(v_on='menuData.on', class_='ma-2', color='primary',
                                  children=[
                                    'menu',
                                    v.Icon(right=True, children=[
                                        'arrow_drop_down'
                                    ])
                                ]),
                }],
            children=[
                v.List(children=items)]
            )

info = v.Chip(class_='ma-2')

layout = v.Layout(children=[
    menu
])

layout
Layout(children=[Menu(children=[List(children=[ListItem(children=[ListItemTitle(children=['Item 1'])])]), ListIt...)

Buttons

[3]: v.Layout(children=[
    v.Btn(color='primary', children=['primary']),
    v.Btn(color='error', children=['error']),
    v.Btn(disabled=True, children=['disabled']),
    v.Btn(children=['reset']),
])
Layout(children=[Btn(children=['primary'], color='primary'), Btn(children=['error'], color='error'), Btn(children=['disabled'], color='disabled'), Btn(children=['reset'])])

[4]: v.Layout(children=[
    v.Btn(color='primary', flat=True, children=['flat']),
    v.Btn(color='primary', round=True, children=['round']),
    v.Btn(color='primary', flat=True, icon=True, children=[v.Icon(children=['thumb_up'])]),
    v.Btn(color='primary', outline=True, children=['outline']),
    v.Btn(color='primary', fab=True, large=True, children=[v.Icon(children=['edit'])])],
)
Layout(children=[Btn(children=['flat'], color='primary'), Btn(children=['round'], color='primary'), Btn(children=['outline'], color='primary'), Btn(children=['edit'], color='primary')])

[5]: def toggleLoading():
    button2.loading = not button2.loading
    button2.disabled = button2.loading

def on_loader_click(*args):
    toggleLoading()
    Timer(2.0, toggleLoading).start()

button2 = v.Btn(loading=False, children=['loader'])
button2.on_event('click', on_loader_click)

v.Layout(children=[button2])
Layout(children=[Btn(children=['loader'], loading=False)])

[6]: toggle_single = v.BtnToggle(v_model=2, class_='mr-3', children=[
    v.Btn(flat=True, children=[v.Icon(children=['format_align_left'])]),
    v.Btn(flat=True, children=[v.Icon(children=['format_align_center'])]),
    v.Btn(flat=True, children=[v.Icon(children=['format_align_right'])]),
    v.Btn(flat=True, children=[v.Icon(children=['format_align_justify'])]),
])
toggle_multi = v.BtnToggle(v_model=[0,2], multiple=True, children=[
(continues on next page)
v.Btn(flat=True, children=[v.Icon(children=['format_bold'])]),
v.Btn(flat=True, children=[v.Icon(children=['format_italic'])]),
v.Btn(flat=True, children=[v.Icon(children=['format_underline'])]),
v.Btn(flat=True, children=[v.Icon(children=['format_color_fill'])]),
]
]
]

v.Layout(children=[
toggle_single,
toggle_multi,
])

Layout(children=[BtnToggle(children=[Btn(children=[Icon(children=['format_align_left'])), Btn(children=[Icon(...

[7]: v.Layout(children=[
    v.Btn(color='primary', children=[
        v.Icon(left=True, children=['fingerprint']),'Icon left'
    ]),
    v.Btn(color='primary', children=[
        'Icon right',
        v.Icon(right=True, children=['fingerprint'])),
    ],
    v.Tooltip(bottom=True, children=[
        v.Btn(slot='activator', color='primary', children=[
            'tooltip'
        ]),
        'Insert tooltip text here'
    ])
])

Layout(children=[Btn(children=[Icon(children=['fingerprint'], left=True), 'Icon left
→'], color='primary'), Btn(...

[8]: lorum_ipsum = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.'

v.Layout(children=[
    v.Dialog(width='500', v_slots=[
        'name': 'activator',
        'variable': 'x',
        'children': v.Btn(v_on='x.on', color='success', dark=True, children=[
            'Open dialog'
        ])
    ],
    ],
    children=[
        v.Card(children=[
            v.CardTitle(class_='headline gray lighten-2', primary_title=True,
            children=[
                'Lorem ipsum'])),
        v.CardText(children=[
            lorum_ipsum])
    ])

(continues on next page)
Slider

```
[9]: slider = v.Slider(v_model=25)
slider2 = v.Slider(thumb_label=True, v_model=25)
slider3 = v.Slider(thumb_label='always', v_model=25)

ipywidgets.jslink((slider, 'v_model'), (slider2, 'v_model'))
```

```
Container(children=[Slider(v_model=25), Slider(thumb_label=True, v_model=25)])
```

Tabs

```
[10]: tab_list = [v.Tab(children=['Tab ' + str(i)] for i in range(1,4))]
content_list = [v.TabItem(children=[lorum_ipsum]) for i in range(1,4)]
tabs = v.Tabs(v_model=1, children=tab_list + content_list)
tabs
```

```
Tabs(children=[Tab(children=['Tab 1']), Tab(children=['Tab 2']), Tab(children=['Tab 3'])], TabItem(children=['L...])
```

Accordion

```
[11]: vepc1 = v.ExpansionPanel(children=[
    v.ExpansionPanelHeader(children=['item1']),
    v.ExpansionPanelContent(children=['First Text'])])

vepc2 = v.ExpansionPanel(children=[
    v.ExpansionPanelHeader(children=['item2']),
    v.ExpansionPanelContent(children=['Second Text'])])

vep = v.ExpansionPanels(children=[vepc1, vepc2])
vl = v.Layout(class_='pa-4', children=[vep])
vl
```
You can search for all available components and attributes in the [Vuetify documentation](https://vuetifyjs.com). Ipyvuetify is based on the syntax of Vue.js and Vuetify, but there are also some differences:

<table>
<thead>
<tr>
<th>Description</th>
<th>Vuetify</th>
<th>Ipyvuetify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component names are written in CamelCase and the v-prefix is removed</td>
<td><code>&lt;v-list-tile .../&gt;</code></td>
<td><code> ListTile(...)</code></td>
</tr>
<tr>
<td>Child components and text are defined in the child traitlet</td>
<td><code>&lt;v-btn&gt;text &lt;v-icon .../&gt;&lt;/v-btn&gt;</code></td>
<td><code>Btn(children=['text', Icon(...)])</code></td>
</tr>
<tr>
<td>Flag attributes require a Boolean value</td>
<td><code>&lt;v-btn round ...&gt;</code></td>
<td><code>Btn(round=True ...)</code></td>
</tr>
<tr>
<td>Attributes are snake_case</td>
<td><code>&lt;v-menu offset-y ...&gt;</code></td>
<td><code>Menu(offset_y=True ...)</code></td>
</tr>
<tr>
<td>The v_model attribute (value in ipywidgets) receives the value directly</td>
<td><code>&lt;v-slider v-model= &quot;some_property&quot; ...&gt;</code></td>
<td><code>Slider(v_model=25...)</code></td>
</tr>
<tr>
<td>The scope of slot cannot currently be specified</td>
<td><code>&lt;v-menu&gt;&lt;template slot:activator=&quot;{ on }&quot;&gt; &lt;v-btn v-on=on&gt;</code></td>
<td><code>Menu(children=[Btn(slot='activator', ...)])</code></td>
</tr>
<tr>
<td>Event listeners are defined with on_event</td>
<td><code>&lt;v-btn @click='someMethod()' ...&gt;</code></td>
<td><code>def some_method(widget, event, data): mit button. on_event('click', some_method)</code></td>
</tr>
<tr>
<td>Regular HTML tags can be created with the Html class</td>
<td><code>&lt;div&gt;...&lt;/div&gt;</code></td>
<td><code>Html(tag='div', children=[])</code></td>
</tr>
<tr>
<td>An underscore must be added to the class and style attributes</td>
<td><code>&lt;v-btn class=&quot;mr-3&quot; style=&quot;...&quot; &gt;</code></td>
<td><code>Btn(class_='mr-3', style_='...')</code></td>
</tr>
</tbody>
</table>

**VuetifyTemplate**

You can get a closer match with the Vue/Vuetify API with VuetifyTemplate. For this you create a subclass of VuetifyTemplate and define your own traitlets. The traitlets can be accessed via the template as if they were in a Vue model. Methods can be defined with the prefix vue_, e.g. `def vue_button_click(self, data)`, which can then be called with `@click="button_click(e)"`. In the following I show you a table with search, sorting and number of lines:

```python
import pandas as pd
import traitlets
import ipyvuetify as v
import json

class PandasDataFrame(v.VuetifyTemplate):
    """
    Vuetify DataTable rendering of a pandas DataFrame
    """
    Args:
    data (DataFrame) - the data to render
    title (str) - optional title
    """
```

(continues on next page)
headers = traitlets.List([]).tag(sync=True, allow_null=True)
items = traitlets.List([]).tag(sync=True, allow_null=True)
search = traitlets.Unicode('').tag(sync=True)
title = traitlets.Unicode('DataFrame').tag(sync=True)
index_col = traitlets.Unicode('').tag(sync=True)
template = traitlets.Unicode(''
	<template>
	<v-card>
		<v-card-title>
		<span class="title font-weight-bold">{{ title }}</span>
		<v-spacer></v-spacer>
		<v-text-field
		v-model="search"
		append-icon="search"
		label="Search ..."
		single-line
		hide-details
		</v-text-field>
		<v-data-table
		:headers="headers"
		:items="items"
		:search="search"
		:item-key="index_col"
		:rows-per-page-items="[25, 50, 250, 500]"
		>
		<template v-slot:no-data>
		<v-alert :value="true" color="error" icon="warning">
		Sorry, nothing to display here :(
		</v-alert>
		</template>
		<template v-slot:no-results>
		<v-alert :value="true" color="error" icon="warning">
		Your search for "{{ search }}" found no results.
		</v-alert>
		</template>
		<template v-slot:items="rows">
		<td v-for="(element, label, index) in rows.item"
		@click=cell_click(element)
		>
		{{ element }}
		</td>
		</template>
		</v-data-table>
		</v-card>
	</template>
''').tag(sync=True)

def __init__(self, *args,

data=pd.DataFrame(),
title=None,
**kwargs):
    super().__init__(*args, **kwargs)
data = data.reset_index()
self.index_col = data.columns[0]
headers = [{
    "text": col,
    "value": col
} for col in data.columns]
headers[0].update({'align': 'left', 'sortable': True})
self.headers = headers
self.items = json.loads(data.to_json(orient='records'))
if title is not None:
    self.title = title

iris = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv')
test = PandasDataFrame(data = iris, title='Iris')
test

[13]: v.Banner(single_line=True,
    v_slots=[{
        'name': 'icon',
        'children': v.Icon(children=['thumb_up'])
    }, {
        'name': 'actions',
        'children': v.Btn(text=True, color='deep-purple accent-4', children=['Action'])
    }],
    children=['One line message text string with two actions on tablet / Desktop'])
Installation

`ipympl` is installed in both the kernel and the Jupyter environment

```bash
pipenv install ipympl
Installing ipympl...
Adding ipympl to Pipfile's [packages]...
✓ Installation Succeeded
```

Then you can activate the Jupyter backend in notebooks by using the following `Matplotlib-Magic`:

```
[1]: %matplotlib widget
```

Examples

Simple Matplotlib interaction

```
[2]: import matplotlib.pyplot as plt
    import numpy as np

    plt.figure(1)
    plt.plot(np.sin(np.linspace(0, 20, 100)))
    plt.show()

    Canvas(toolbar=Toolbar(toolitems=[('Home', 'Reset original view', 'home', 'home'), ('Back', 'Back to previous ...
```

3D plot: subplot3d_demo.py

```
[3]: from mpl_toolkits.mplot3d import axes3d

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Grab some test data.
    X, Y, Z = axes3d.get_test_data(0.05)

    # Plot a basic wireframe.
    ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)

    fig.canvas.layout.max_width = '800px'

    plt.show()

    Canvas(toolbar=Toolbar(toolitems=[('Home', 'Reset original view', 'home', 'home'), ('Back', 'Back to previous ...
```
More complex example from the Matplotlib gallery

```python
[4]: import numpy as np
    import matplotlib.pyplot as plt

    np.random.seed(0)

    n_bins = 10
    x = np.random.randn(1000, 3)

    fig, axes = plt.subplots(nrows=2, ncols=2)
    ax0, ax1, ax2, ax3 = axes.flatten()

    colors = ['red', 'tan', 'lime']
    ax0.hist(x, n_bins, density=1, histtype='bar', color=colors, label=colors)
    ax0.legend(prop={'size': 10})
    ax0.set_title('bars with legend')

    ax1.hist(x, n_bins, density=1, histtype='bar', stacked=True)
    ax1.set_title('stacked bar')

    ax2.hist(x, n_bins, histtype='step', stacked=True, fill=False)
    ax2.set_title('stack step (unfilled)')

    # Make a multiple-histogram of data-sets with different length.
    x_multi = [np.random.randn(n) for n in [10000, 5000, 2000]]
    ax3.hist(x_multi, n_bins, histtype='bar')
    ax3.set_title('different sample sizes')

    fig.tight_layout()
    plt.show()
```

Embed Jupyter widgets

Jupyter widgets can be serialised and then embedded in other contexts:

- static web pages
- Sphinx documentation
- HTML converted notebooks on Nbviewer

The npm package `@jupyter-widgets/html-manager` allows embedding in two different ways:

- embedding the standard elements that can be used on any website
- embedding with `RequireJS` also for custom widgets.
Embed widgets in HTML pages

The widgets menu provides several options for this:

**Save Notebook Widget State**  A notebook file is saved with the current widget status as metadata. This allows to be rendered with the widgets in the browser.

**Clear Notebook Widget State**  The widget status metadata is deleted from the notebook file.

**Embed widgets**  The menu item offers a dialog box with an HTML page on which the current widgets are embedded. The RequireJS embedder is used to support custom widgets.

---

**Note:** The first script tag loads RequireJS from a CDN. However, RequireJS should be made available on the site itself and this script tag should be deleted.

---

**Note:** The second script tag loads the RequireJS widget embedder. This defines suitable modules and then sets up a function for rendering all widget views contained on the page.

If you only embed standard widgets and don’t use RequireJS, you can replace the first two script tags with a script tag that loads the standard script.

**Download Widget State**  The option downloads a JSON file that contains the serialized status of all widget models currently in use in the application/vnd.jupyter.widget-state+json format specified in the @jupyter-widgets/schema npm package.

---

Sphinx integration

**Jupyter Sphinx**

`jupyter_sphinx` enables jupyter-specific functions in Sphinx. It can be installed with `pip`.

**Configuration**

Adds `jupyter_sphinx.embed_widgets` to the list of extensions in the `conf.py` file.

Then you can use the following directives in reStructuredText:

**Ipywidgets-setup**

```python
from ipywidgets import VBox, jsdlink, IntSlider, Button
```

**Ipywidgets-display**

```python
s1, s2 = IntSlider(max=200, value=100), IntSlider(value=40)
b = Button(icon='legal')
jsdlink((s1, 'value'), (s2, 'max'))
VBox([s1, s2, b])
```
Example

```python
.. ipywidgets-setup::

    from ipywidgets import VBox, jsdlink, IntSlider, Button

.. ipywidgets-display::

    :hide-code:

    s1, s2 = IntSlider(max=200, value=100), IntSlider(value=40)
b = Button(icon='legal')
jsdlink((s1, 'value'), (s2, 'max'))
VBox([s1, s2, b])
```

Options

The `ipywidgets-setup` and `ipywidgets-display` directives have the following options:

- **ipywidgets-setup** with the option :show: to display the setup code as a code block
- **ipywidgets-display**
  - with the following options:
    - :hide-code: doesn’t show the code, only the widget
    - Widget
      - :code-below: shows the code after the widget
      - :alt: Alternate text if the widget cannot be rendered

See also:

Options

### 3.2.7 nbextensions

Jupyter Notebook Extensions contains a collection of extensions. These are mostly written in Javascript and are loaded locally in your browser.

See also:

- Docs
- Github
### Installation

1. **Installation with Pipenv:**

   ```bash
   $ pipenv install jupyter_contrib_nbextensions
   Installing jupyter_contrib_nbextensions...
   ...
   ```

2. **Installation of the associated Javascript and CSS files:**

   ```bash
   $ pipenv run jupyter contrib nbextension install --user
   [I 20:57:19 InstallContribNbextensionsApp] jupyter contrib nbextension install --user
   [I 20:57:19 InstallContribNbextensionsApp] Installing jupyter_contrib_nbextensions nbextension files to jupyter data directory
   ...
   ```

3. **Check the installation:**

   ```bash
   $ pipenv run jupyter nbextension list
   Known nbextensions:
   config dir: /Users/veit/.jupyter/nbconfig
   notebook section
   nbextensions_configurator/config_menu/main enabled
   - Validating: problems found:
     - require? X nbextensions_configurator/config_menu/main
   contrib_nbextensions_help_item/main enabled
   - Validating: OK
   tree section
   nbextensions_configurator/tree_tab/main enabled
   - Validating: problems found:
     - require? X nbextensions_configurator/tree_tab/main
   config dir: /Users/veit/.local/share/virtualenvs/jupyter-tutorial--q5BvmfG/bin/../etc/jupyter/nbconfig
   notebook section
   jupyter-js-widgets/extension enabled
   - Validating: OK
   ```

4. **Latex environments**

   ```bash
   $ pipenv run jupyter nbextension install --py latex_envs --user
   Installing /Users/veit/.local/share/virtualenvs/jupyter-tutorial--q5BvmfG/lib/python3.7/site-packages/latex_envs/static -> latex_envs
   ...
   - Validating: OK
   To initialize this nbextension in the browser every time the notebook (or other app) loads:
   ```

   ```bash
   jupyter nbextension enable latex_envs --user --py
   ...
   ```

   ```bash
   $ pipenv run jupyter nbextension enable --py latex_envs --user
   ```

(continues on next page)
Enabling notebook extension latex_envs/latex_envs...
- Validating: OK

5. **yapf** Code Prettyfier

... for Python:

```bash
$ pipenv install yapf
Installing yapf...
Collecting yapf
  Downloading https://files.pythonhosted.org/packages/79/22/d711c0803b6c3cc8c96eb54509f23fec1e3c078d5bfc6eb11094e762e7bc/yapf-0.28.0-py2.py3-none-any.whl (180kB)
Installing collected packages: yapf
Successfully installed yapf-0.28.0
```

... for Javascript:

```bash
$ npm install js-beautify
```

```bash
+ js-beautify@1.10.0
added 29 packages from 21 contributors and audited 32 packages in 2.632s
found 0 vulnerabilities
```

... for R:

```bash
$ Rscript -e 'install.packages(c("formatR", "jsonlite"), repos="http://cran.rstudio.com")'
Installiere Pakete nach '/usr/local/lib/R/3.6/site-library'
```

6. Highlighter

```bash
$ pipenv run jupyter nbextension install https://rawgit.com/jfbercher/small_nbextensions/master/highlighter.zip --user
Installing /srv/jupyter/.local/share/virtualenvs/jupyterhub-aFv4x91W/lib/python3.5/site-packages/highlighter.zip
Enabling notebook extension highlighter/highlighter...
- Validating: OK
```

7. **nbTranslate**

```bash
$ pipenv install jupyter_latex_envs --upgrade --user
Installing jupyter_latex_envs...
```

```bash
$ pipenv run jupyter nbextension install --py latex_envs --user
Installing /srv/jupyter/.local/share/virtualenvs/jupyterhub-aFv4x91W/lib/python3.5/site-packages/latex_envs/static -> latex_envs
```

```bash
$ pipenv run jupyter nbextension enable --py latex_envs
```
List of extensions

You can activate and configure the notebook extensions by clicking on the Nbextensions tab. There you have access to the extensions, which can be activated/deactivated via checkboxes. In addition, documentation and configuration options are displayed for each extension.

Hereinafter I will give a brief overview of some of the notebook extensions.

(some) LaTeX environments for Jupyter notebook enables the use of Markdown cells for LaTeX commands and environments. In addition, two menus are added: LaTeX_envs for quick selection of the suitable LaTeX environment and Some configuration options for further options:

The notebook can then be exported as an HTML or LaTeX document.

The configuration of the LaTeX environments is done in user_envs.json and for the styles in latex_env.css. Additional environments can be added in user_envs.json or in thmsInNb4.js (→ LaTeX-Environments doc).

jupyter-autopep8 formats/beautifies Python code in cells. The extension uses autopep8 and can therefore only be used with a Python kernel.

A Code Prettifier formats/beautifies code notebook code cells. The current notebook kernel is used, which is why the Prettifier package used must be available in this kernel. Sample implementations are provided for ipython,
R, and Javascript kernels.

Limit Output limits the number of characters that a code cell outputs as text or HTML. This also breaks endless loops. You can set the number of characters with the `ConfigManager`:

```python
from notebook.services.config import ConfigManager
cm = ConfigManager().update('notebook', {'limit_output': 1000})
```

**Nbextensions edit menu item** adds an edit menu to open the configuration page of `nbextensions`.

**Printview** adds an icon to display the print preview of the current notebook in a new browser tab.

**Ruler** adds a ruler after a certain number of characters. The number of characters can be specified with the `ConfigManager`:

```python
from notebook.services.config import ConfigManager
ip = get_ipython()
cm = ConfigManager(parent=ip)
cm.update('notebook', {'ruler_column': [80]})
```

**Scratchpad notebook extension** adds a note cell to the notebook. In this cell you can run code from the current kernel without changing the document.

**Snippets** adds a configurable menu item to notebooks to insert snippets, boilerplate and code examples.

You can also define your own menu items, see also **Snippets**.

**Table of Contents (2)** makes it possible to collect all headings and display them in a floating window, as a sidebar or in a navigation menu.

If headings shouldn’t be displayed in the table of contents, you can do this with:

```markdown
## My title <a class="tocSkip">
```

The table of contents can also be exported by specifying a corresponding template, e.g.

```
$ jupyter nbconvert mynotebook.ipynb --template toc2
```

General documentation on templates can be found in **Customizing exporters**.
Tree-filter filters in the Jupyter dashboard by file name.

A 2to3 converter converts Python2 to Python3 code in a code cell using the lib2to3 library.

Codefolding enables code folding in code cells.

```python
In [ ]:
class MyClass(object):
    ""
    This is a test class
    ""
def afun(param):
    """something gets computed here"
    return param+1**2
```

Usually code folding is retained when exporting with nbconvert. This can either be changed in jupyter_nbconvert_config.py with:

```
c.CodeFoldingPreprocessor.remove_folded_code=True = True
```

or on the command line with:

```
$ jupyter nbconvert --to html --CodeFoldingPreprocessor.remove_folded_code=True --mynotebook.ipynb
```

Collapsible Headings enables notebooks to have collapsible sections separated by headings.

Datestamper inserts the current time and date in one cell.

Hinterland enables autocompletion.

Variable Inspector collects all defined variables and displays them in a floating window.

Purpose automatically loads a series of latex commands from the latexdefs.tex file when a notebook is opened.

Create plugin

In addition to the existing notebook extensions, other plugins can also be added. The directory in which jupyter_contrib_nbextensions/nbextensions is located can be found with pip show:

```
$ pipenv run pip show jupyter_contrib_nbextensions
Name: jupyter-contrib-nbextensions
Version: 0.5.1
Summary: A collection of Jupyter nbextensions.
Home-page: https://github.com/ipython-contrib/jupyter_contrib_nbextensions.git
Author: ipython-contrib and jupyter-contrib developers
Author-email: jupytercontrib@gmail.com
License: BSD
Location: /Users/veit/.local/share/virtualenvs/jupyter-tutorial--q5BvmfG/lib/python3.7/site-packages
Requires: lxml, jupyter-contrib-core, nbconvert, jupyter-latex-envs, jupyter-core, pyyaml, jupyter-nbextensions-configurator, notebook, traitlets, jupyter-highlight-selected-word, tornado, ipython-genutils
Required-by:
```

This directory contains the individual notebook extensions, e.g. with the following structure:
### main.js

Contains the actual logic of the extension, e.g.:

```javascript
define([
    'require',
    'base/js/namespace',
], function (
    requirejs
, $,
    Jupyter,
) {
    "use strict";

    // define default values for config parameters
    var params = {
        my_config_value : 100
    };

    var initialize = function () {
        $.extend(true, params, Jupyter.notebook.config.myextension);

        $('!<link/>')
        .attr({
            rel: 'stylesheet',
            type: 'text/css',
            href: requirejs.toUrl('./myextension.css')
        })
        .appendTo('head');
    }

    var load_ipython_extension = function () {
        return Jupyter.notebook.config.loaded.then(initialize);
    }

    return {
        load_ipython_extension : load_ipython_extension
    };
});
```

### main.yaml

YAML file that describes the extension for the Jupyter Extensions Configurator.

- **Type**: Jupyter Notebook Extension
- **Compatibility**: 3.x, 4.x, 5.x, 6.x
- **Name**: My notebook extensions
- **Main**: main.js
- **Link**: README.md
- **Description**: My notebook extension helps with the use of Jupyter notebooks.
Parameters:
- none

More information about the options supported by the configurator can be found on GitHub: jupyter_nbextensions_configurator.

readme.md  Markdown file that describes the extension and how it can be used. This is also displayed in the Nbextensions tab.

See also:
- Notebook extension structure

Setup Jupyter Notebook Extension

This is an extension that fixes some problems when working with notebooks that Joel Grus presented at JupyterCon 2018: I Don’t Like Notebooks:
- it asks you to name the notebook
- it creates a template to improve the documentation
- it imports and configures frequently used libraries

Installation

1. Find out where the notebook extensions are installed:

   $ pipenv run pip show jupyter_contrib_nbextensions
   Name: jupyter-contrib-nbextensions
   Version: 0.5.1
   Summary: A collection of Jupyter nbextensions.
   Home-page: https://github.com/ipython-contrib/jupyter_contrib_nbextensions.git
   Author: ipython-contrib and jupyter-contrib developers
   Author-email: jupytercontrib@gmail.com
   License: BSD
   Location: /Users/veit/.local/share/virtualenvs/jupyter-tutorial--q5BvmfG/lib/
   →python3.7/site-packages
   Requires: lxml, jupyter-contrib-core, nbconvert, jupyter-latex-envs, jupyter-core, ...
   →pyyaml, jupyter-nbextensions-configurator, notebook, traitlets, jupyter-highlight-
   →selected-word, tornado, ipython-genutils
   Required-by:

2. Download the Setup directory in jupyter_contrib_nbextensions/nbextensions/.

3. Install the extension with

   $ pipenv run jupyter contrib nbextensions install --user
   ...
   [I 10:54:46 InstallContribNbextensionsApp] Installing /Users/veit/.local/share/
   →virtualenvs/jupyter-tutorial--q5BvmfG/lib/python3.7/site-packages/jupyter_contrib_-
   →nbextensions/nbextensions/setup → setup
   →Jupyter/nbextensions/setup/
4. Activate the Setup extension in Nbextensions.

Finally you can create a new notebook, which then has the following structure: setup.ipynb.

See also:
- Set Your Jupyter Notebook up Right with this Extension
- GitHub

setup.ipynb

Introduction

State notebook purpose here

Imports

Import libraries and write settings here.

```ipython
# Data manipulation
import pandas as pd
import numpy as np

# Options for pandas
pd.options.display.max_columns = 50
pd.options.display.max_rows = 30

# Display all cell outputs
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'

from IPython import get_ipython
ipython = get_ipython()

# autoreload extension
if 'autoreload' not in ipython.extension_manager.loaded:
```

(continues on next page)
%load_ext autoreload

%autoreload 2

# Visualizations
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import iplot, init_notebook_mode
init_notebook_mode(connected=True)

import cufflinks as cf
cf.go_offline(connected=True)
cf.set_config_file(theme='white')

Analysis/Modeling

Do work here

Results

Show graphs and stats here

Conclusions and Next Steps

Summarize findings here

ipylayout

ipylayout is based on GoldenLayout, a multi-screen layout manager for web applications.

Installation

ipylayout can be easily installed with pipenv:

$ pipenv install ipylayout
Installing ipylayout...
...  

If not already done, ipywidgets will also be installed.
Example

For the following example you also need the Python packages ipyleaflet and ipympl.

```python
[1]: %matplotlib widget
import ipyleaflet
import ipywidgets
import matplotlib.pyplot as plt
import numpy as np
plt.ioff()

[2]: # create a plot
fig = plt.figure()
fig.canvas.header_visible = False
fig.canvas.layout.min_height = '300px'
fig.canvas.layout.width = '100%'
plt.title('Plotting: y=sin(x)')
x = np.linspace(0, 20, 500)
lines = plt.plot(x, np.sin(x))

[3]: # create a slider
slider = ipywidgets.FloatSlider()

[4]: # create a map
m = ipyleaflet.Map(
    center=(52.204793, 360.121558),
    zoom=4
)

[5]: # create a layout
l = ipywidgets.Layout(layout=ipywidgets.Layout(width='100%', height='800px'))
l.theme = 'light' # light or dark
l.config = {
    'content': [{
        'type': 'row',
        'content': [{
            'type': 'component',
            'componentName': 'name0',
            'componentState': { 'label': 'A' }
        },
        'type': 'column',
        'content': [{
            'type': 'component',
            'componentName': 'name1',
            'componentState': { 'label': 'B' }
        },
```

(continues on next page)
l.components = {'name0': slider, 'name1': m, 'name2': fig.canvas}

[6]:

Layout(config={'content': [{'type': 'row', 'content': [{'type': 'component', 'componentName': 'name0', 'componentState': {'label': 'C'}}, {'type': 'component', 'componentName': 'name1'}, {'type': 'component', 'componentName': 'name2'}], 'content': []})
**ipylayout**

*ipylayout* basiert auf *GoldenLayout*, einem Multi-Screen-Layout-Manager für Webanwendungen.

**Installation**

*ipylayout* kann einfach mit *pipenv* installiert werden:

```bash
$ pipenv install ipylayout
Installing ipylayout...
```

Sofern noch nicht geschehen, wird auch *ipywidgets* mitinstalliert.

**Beispiel**

Für das folgende Beispiel benötigt Ihr zusätzlich noch die Python-Pakete *ipyleaflet* und *ipyml*.

*ipylayout* can also be used together with *Voila*.
3.2.8 nbsphinx

nbsphinx is a Sphinx extension that provides a parser for *.ipynb files: Jupyter Notebook code cells are displayed in both HTML and LaTeX output. Notebooks with no output cells saved are automatically created during the Sphinx build process.

Installation

$ pipenv install sphinx nbsphinx

Requirements

• nbconvert

Configuration

Configure Sphinx

1. Creating a documentation with Sphinx:

$ pipenv run python3 -m sphinx.cmd.quickstart

2. The Sphinx configuration file conf.py is then located in the newly created directory. In this, ``nbsphinx is added as an extension and notebook checkpoints are excluded:

```plaintext
extensions = [
    ...,'nbsphinx',
]
...
exclude_patterns = [
    ...
    '**/.ipynb_checkpoints',
]
```

You can find an example in the conf.py file of the Jupyter tutorial.

You can make further configurations for nbsphinx.

Timeout In the standard setting of nbsphinx, the timeout for a cell is set to 30 seconds. You can change this for your Sphinx project in the conf.py file with

```plaintext
nbsphinx_timeout = 60
```

Alternatively, you can also specify this for individual code cells in the metadata of the code cell:

```json
{
    "cells": [
    {
        "cell_type": "markdown",
        "nbsphinx": {
```
If the timeout is to be deactivated, -1 can be specified.

**Custom formats**  Libraries such as jupyter save notebooks in other formats, e.g. as R-Markdown with the suffix Rmd. So that these can also be executed by nbsphinx, further formats can be specified in the Sphinx configuration file conf.py with nbsphinx_custom_formats, e.g.

```python
import jupytext
nbsphinx_custom_formats = {
    '.Rmd': lambda s: jupytext.reads(s, '.Rmd'),
}
```

**Configure cells**

**Don’t show cell**

```json
{
    "cells": [
        {
            "cell_type": "markdown",
            "metadata": {
                "nbsphinx": "hidden"
            }
        },
    ],
}
```

**nbsphinx-toctree**  With this instruction Sphinx will create a table of contents within a notebook cell, e.g.

```json
{
    "cells": [
        {
            "cell_type": "markdown",
            "metadata": {
                "nbsphinx-toctree": {
                    "maxdepth": 2
                }
            }
        },
        "source": [
            "The following title is rendered as `\`toctree caption\`\`.
            
            
            ## Content
            
            [A notebook](a-notebook.ipynb)
            
            [An external HTML link](https://jupyter-tutorial.readthedocs.io/)
        ]
    ]
}
```
Further options you will find in the Sphinx documentation.

**Build**

1. Now you can add your *.ipynb file in the table of contents of your index.rst file, see e.g. jupyter-tutorial/ipython/index.rst.

2. Finally, you can generate the pages, e.g. HTML with

   ```bash
   $ pipenv run python3 -m sphinx <source-dir> <build-dir>
   ```

   or

   ```bash
   $ pipenv run python3 -m sphinx <source-dir> <build-dir> -j <number-of-processes>
   ```

   where -j is the number of processes to run in parallel.

   If you want to create a LaTeX file, you can do so with

   ```bash
   $ pipenv run python3 -m sphinx <source-dir> <build-dir> -b latex
   ```

3. Alternatively, you can have the documentation generated automatically with sphinx-autobuild. It can be installed with

   ```bash
   $ pipenv run python3 -m pip install sphinx-autobuild
   ```

   The automatic creation can then be started with

   ```bash
   $ pipenv run python3 -m sphinx_autobuild <source-dir> <build-dir>
   ```

   This starts a local web server that provides the generated HTML pages at http://localhost:8000/. And every time you save changes in the Sphinx documentation, the corresponding HTML pages are regenerated and the browser view is updated.

   You can also use this to automatically generate the LaTeX output:

   ```bash
   $ pipenv run python3 -m sphinx_autobuild <source-dir> <build-dir> -b latex
   ```

4. Another alternative is publication on readthedocs.org.

   To do this, you first have to create an account at https://readthedocs.org/ and then connect your GitLab, Github or Bitbucket account.
Markdown cells

Equations  Equations can be specified *inline* between ``$`` characters, e.g.

\[ e^{i\pi} = -1 \]

Equations can also be expressed line by line e.g.

\[
\begin{equation}
\int_{-\infty}^{\infty} f(x) \delta(x - x_0) \, dx = f(x_0)
\end{equation}
\]

See also:

* Equation Numbering

Quotes  nbsphinx supports the same syntax for quotations as nbconvert:

```
<cite data-cite="kluyver2016jupyter">Kluyver et al. (2016)</cite>
```

Info and warning boxes

```
<div class="alert alert-info">
**Note:** This is a note!
</div>
```

Links to other notebooks

```
[a link to a notebook in a subdirectory](subdir/notebook-in-a-subdir.ipynb)
```

Links to *.rst* files

```
[reStructuredText file](rst-file.rst)
```

Links to local files

```
[Pipfile](Pipfile)
```

Code cells

Javascript  Javascript can be used for the generated HTML, e.g.:

```
\%

var text = document.createTextNode("Hello, I was generated with JavaScript!");
// Content appended to "element" will be visible in the output area:
element.appendChild(text);
```
3.2.9 Use cases

In some companies, Jupyter notebooks are used to explore the ever-increasing amounts of data. These include:

- Netflix
  - Beyond Interactive: Notebook Innovation at Netflix
  - Part 2: Scheduling Notebooks at Netflix

- Bloomberg BQuant platform
  - Bloomberg BQuant (BQNT)

- PayPal
  - PayPal Notebooks: Data science and machine learning at scale, powered by Jupyter

- Société Générale
  - Jupyter & Python in the corporate LAN
CHAPTER
FOUR

READ, PERSIST AND PROVIDE DATA

You can get an overview of public repositories with research data e.g. in Open data.

In addition to specific Python libraries for accessing remote storage media and geodata, we will introduce you to different serialisation formats and three tools in more detail that make data accessible:

- Requests
- BeautifulSoup
- Intake

See also:

- pandas I/O API  The pandas I/O API is a set of top level functions that return a pandas object. In most cases corresponding write methods are also available.
- Scrapy  Framework for extracting data from websites as JSON, CSV or XML files.
- Pattern  Python module for data mining, natural language processing, ML and network analysis.
- Web Scraping Reference  Overview of web scraping with Python.

We introduce PostgreSQL, SQLAlchemy and PostGIS for storing relational data, Python objects and geodata.

For the storage of other data types we introduce you to different NoSQL databases and concepts.

Next we will show you how to provide data with FastAPI or gRPC.

With DVC we present you a tool that allows data provenance, i.e. the traceability of the origin of the data and the way they are created.

Finally in the next chapter you will learn some good practices and helpful Python packages to clean up and validate data.

4.1 Open data

You can get an overview of public repositories with research data e.g. in

- Registry of research data repositories (re3data)
- Awesome Public Datasets
- Public APIs
- Machine learning datasets
- Roboflow Computer Vision Datasets
- DBpedia
4.2 Serialisation formats

Data serialisation converts structured data to a format that allows sharing and or storing of the data. Before serialising data you have to decide how the data should be structured – flat or nested. The differences in the two styles are shown in the examples below:

**Flat JSON style:**

```json
{
    "id" : "veit",
    "first_name" : "Veit",
    "last_name" : "Schiele",
}
```

**Nested JSON style:**

```json
{
    "id" : "veit",
    "first_name" : "Veit",
    "last_name" : "Schiele",
}
```
4.2.1 Serialising data

If the data to be serialised is in flat style, Python offers two methods to serialise the data.

repr

```python
a = { "id" : "veit", "first_name": "Veit", "last_name": "Schiele" }

# Return a printable representation of the input
print(repr(a))

# Write content to the file
with open('data.py', 'w') as f:
    f.write(repr(a))
```

ast.literal_eval

The `literal_eval` method parses and evaluates an expression for a Python datatype. Supported data types are: strings, numbers, tuples, lists, dicts, booleans, and None.

```python
with open('data.py', 'r') as f:
    inp = ast.literal_eval(f.read())
```
### CSV

#### Overview

<table>
<thead>
<tr>
<th>Data structure support</th>
<th>CSV is used to store tabular data, but unlike other serialisation formats reviewed here, it’s not suitable for (nested) objects.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardisation</td>
<td>CSV is not well standardised: neither the encoding is defined nor the separation of the cell contents (comma, semicolon etc.).</td>
</tr>
<tr>
<td>Schema IDL</td>
<td>No</td>
</tr>
<tr>
<td>Language support</td>
<td>++ The CSV format is well supported by almost every programming language. A csv module is included in the Python standard library and pandas can read a CSV file straight into a Dataframe. Even if CSV is the only format described here that is well supported by spreadsheet programs like Excel, you should see if you can import more structured Excel files directly, e.g. with pandas read_excel.</td>
</tr>
<tr>
<td>Human readability</td>
<td>+ CSV is readable especially for integer or decimal numbers with the same character length. In all other cases it will be difficult to identify the corresponding columns.</td>
</tr>
<tr>
<td>Speed</td>
<td>+ CSV is very fast to serialise and deserialise.</td>
</tr>
<tr>
<td>File size</td>
<td>++ Only Protocol Buffers (Protobuf) should be more compact.</td>
</tr>
</tbody>
</table>

#### Example

*iris.csv*

| 5.1,0.222222222,3.5,0.625,1.4,0.06779661,0.2,0.041666667,setosa |
| 4.9,0.166666667,3,0.416666667,1.4,0.06779661,0.2,0.041666667,setosa |
| 4.7,0.111111111,3.2,0.5,1.3,0.050847458,0.2,0.041666667,setosa |
| 4.6,0.083333333,3.1,0.458333333,1.5,0.084745763,0.2,0.041666667,setosa |
| 5,0.194444444,3.6,0.666666667,1.4,0.06779661,0.2,0.041666667,setosa |

See also:

- RFC 4180
## JSON

### Overview

<table>
<thead>
<tr>
<th>Data structure support</th>
<th>+--</th>
<th>JSON supports array and map or object structures and many different data types including strings, numbers, boolean, null etc., but no date formats. However, JSON does not support all data types of JavaScript: NaN and Infinity become null. Note that the JSON syntax also don’t support comments and you have to work around for example with a <strong>comment</strong> key/value pair.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardisation</td>
<td>+</td>
<td>JSON has a formal strongly typed standard (see also RFC 8259). However, JSON data also contains some pitfalls due to the ambiguity of the JSON specifications: A JSON parser MUST accept all texts that conform to the JSON grammar (RFC 7159) and An implementation may set limits on the size of texts that it accepts. An implementation may set limits on the maximum depth of nesting. An implementation may set limits on the range and precision of numbers. An implementation may set limits on the length and character contents of strings (RFC 7158 #9). Unfortunately there is neither a reference implementation nor an official test suite that would show the expected behaviour – at least JSON_Checker gives some hints.</td>
</tr>
<tr>
<td>Schema IDL</td>
<td>++</td>
<td>Partly with JSON Schema Proposal, JSON Encoding Rules (JER), Kwalify, Rx, JSON-LD or JMES-Path. After all, there are many different validators available.</td>
</tr>
<tr>
<td>Language support</td>
<td>++</td>
<td>The JSON format is very well supported by almost every programming language. The data structure of JSON closely represent objects in many languages e.g. a Python dict can be represented as JSON object, and a Python list by a JSON array.</td>
</tr>
<tr>
<td>Human readability</td>
<td>--</td>
<td>JSON is a human-readable serialisation format but it does not support comments.</td>
</tr>
<tr>
<td>Speed</td>
<td>++</td>
<td>JSON is one of the fastest human-readable formats to serialise and deserialise.</td>
</tr>
<tr>
<td>File size</td>
<td>--</td>
<td>JSON is in the medium range similar to YAML and TOML.</td>
</tr>
</tbody>
</table>

### Example

Response of the OSM-Nomination-API

```json
[
  {
    'place_id': 234847916,
    'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright',
    'osm_type': 'relation',
    'osm_id': 131761,
    'boundingbox': ['52.5200695', '52.5232601', '13.4103097', '13.4160798'],
    'lat': '52.521670650000004',
    'lon': '13.413278026558228',
    'display_name': 'Alexanderplatz, Mitte, Berlin, 10178, Deutschland',
  }
]```

(continues on next page)
'class': 'highway',
'type': 'pedestrian',
'importance': 0.6914982526373583
},


'place_id': 53256307,
'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright',
'osm_type': 'node',
'osm_id': 4389211800,
'boundingbox': ['52.5231653', '52.5232653', '13.414475', '13.414575'],
'lat': '52.5232153',
'lon': '13.414525',
'display_name': 'Alexanderplatz, Alexanderstraße, Mitte, Berlin, 10178, Deutschland',
'class': 'highway',
'type': 'bus_stop',
'importance': 0.22100000000000003,
'icon': 'https://nominatim.openstreetmap.org/images/mapicons/transport_bus_stop2.png'
},


'place_id': 90037579,
'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright',
'osm_type': 'way',
'osm_id': 23853138,
'boundingbox': ['52.5214702', '52.5217276', '13.4037885', '13.4045026'],
'lat': '52.5215991',
'lon': '13.404112295159964',
'display_name': 'Alexander Plaza, 1, Rosenstraße, Mitte, Berlin, 10178, Deutschland',
'class': 'tourism',
'type': 'hotel',
'importance': 0.11100000000000002,
'icon': 'https://nominatim.openstreetmap.org/images/mapicons/accommodation_hotel2.png'
]
Overview

<table>
<thead>
<tr>
<th>Data structure support</th>
<th>Pickle is used to store Python object structures like list or dict in a byte stream. In contrast to marshal, already serialised objects are tracked so that later references are not serialised again. Recursive objects are also possible.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardisation</td>
<td>Pickle is defined in the Python Enhancement Proposals Proposals 307, 3154 and 574.</td>
</tr>
<tr>
<td>Schema IDL</td>
<td>No</td>
</tr>
<tr>
<td>Language support</td>
<td>Python-specific</td>
</tr>
<tr>
<td>Human readability</td>
<td>Pickle is a binary serialisation format, but it can be easily read with Python.</td>
</tr>
<tr>
<td>Speed</td>
<td>The pickle format can usually be serialised and deserialised quickly by Python; see also Don’t pickle your data.</td>
</tr>
<tr>
<td>File size</td>
<td>Compact binary format, which can, however, be compressed even further, see also Data Compression and Archiving.</td>
</tr>
</tbody>
</table>

Example

1. Write

```python
import pickle

# An arbitrary collection of objects supported by pickle.
data = {
    'a': [1, 2.0, 3, 4+6j],
    'b': ('character string', b'byte string'),
    'c': {None, True, False}
}

with open('data.pickle', 'wb') as f:
    # Pickle the 'data' dictionary using the highest protocol available.
    pickle.dump(data, f, pickle.HIGHEST_PROTOCOL)
```

2. Read

```python
import pickle

with open('data.pickle', 'rb') as f:
    # The protocol version used is detected automatically, so we do not
    # have to specify it.
data = pickle.load(f)
```

See also:
pickle – Python object serialization

Documentation of the pickle module

4.2. Serialisation formats
shelve – Python object persistence

Indexed databases of pickle objects

Uwe Korn: The implications of pickling ML models

Alternatives to pickle for ML models

Ned Batchelder: Pickle’s nine flaws

Disadvantages of pickle and alternatives

Protocol Buffers (Protobuf)

Overview

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>Protobuf allows you to define data structures in *.proto files. Protobuf supports many primitive types, which can be combined into nested classes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardization</td>
<td>++</td>
<td>Protobuf is a strongly typed flexible standard.</td>
</tr>
<tr>
<td>Schema-IDL</td>
<td>++</td>
<td>Built-in IDL compiler</td>
</tr>
<tr>
<td>Language support</td>
<td>++</td>
<td>The protobuf format is well supported by many programming languages.</td>
</tr>
<tr>
<td>Human readability</td>
<td>--</td>
<td>Protobuf is not designed to be human readable.</td>
</tr>
<tr>
<td>Speed</td>
<td>++</td>
<td>Protobuf is very fast, especially in C++.</td>
</tr>
<tr>
<td>File size</td>
<td>++</td>
<td>Protobuf is the most compact format.</td>
</tr>
</tbody>
</table>

See also:

- Home
- GitHub
- Language Guide (proto3)
- gRPC

TOML

Overview

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>TOML (Tom’s Obvious, Minimal Language) supports most common including strings, integers, floats and dates, but not references like YAML does.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardization</td>
<td>++</td>
<td>TOML is a formal strongly typed standard.</td>
</tr>
<tr>
<td>Schema-IDL</td>
<td>+-</td>
<td>Partly with JSON Schema Everywhere</td>
</tr>
<tr>
<td>Language support</td>
<td>++</td>
<td>TOML is a relatively new serialization format and doesn’t have the same broad support as JSON, CSV or XML for various programming languages.</td>
</tr>
<tr>
<td>Human readability</td>
<td>++</td>
<td>One of TOML’s primary goals was to be very easy to read.</td>
</tr>
<tr>
<td>Speed</td>
<td>+-</td>
<td>TOML can be processed at medium speed.</td>
</tr>
<tr>
<td>File size</td>
<td>-</td>
<td>Only XML is less compact.</td>
</tr>
</tbody>
</table>
Example

pyproject.toml

```toml
[tool.black]
line-length = 79

[tool.isort]
atomic=true
force_grid_wrap=0
include_trailing_comma=true
lines_after_imports=2
lines_between_types=1
multi_line_output=3
not_skip="__init__.py"
use_parentheses=true

known_first_party="jupyter-tutorial"
known_third_party=["mpi4py", "numpy", "requests"]
```

See also:
- Home
- GitHub
- Wiki
- What is wrong with TOML?
- An INI critique of TOML

XML

Overview

<table>
<thead>
<tr>
<th>Data structure support</th>
<th>++</th>
<th>XML is very flexible as each element can have attributes and arbitrary child elements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardisation</td>
<td>++</td>
<td>XML is well standardised, the specification can be found at <a href="https://www.w3.org/TR/xml/">https://www.w3.org/TR/xml/</a>. XML supports both DOM style and streaming SAX style parsers.</td>
</tr>
<tr>
<td>Schema-IDL</td>
<td>++</td>
<td>XML schema, RELAX NG</td>
</tr>
<tr>
<td>Language support</td>
<td>+</td>
<td>Supported in all major languages, usually with built-in libraries.</td>
</tr>
<tr>
<td>Human readability</td>
<td>++</td>
<td>XML is a human-readable serialisation protocol. One disadvantage of XML is it’s verbosity, in particular it’s descriptive end tags.</td>
</tr>
<tr>
<td>Speed</td>
<td>+</td>
<td>XML is quite fast, although typically slower to parse than JSON.</td>
</tr>
<tr>
<td>File size</td>
<td>--</td>
<td>XML has the largest file size in comparison.</td>
</tr>
</tbody>
</table>

4.2. Serialisation formats
Example

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"></xs:schema>
```

See also:
- Home
- Specification
- Validator
- The XML FAQ

YAML

Overview

<table>
<thead>
<tr>
<th>Data structure support</th>
<th>++</th>
<th>YAML, short for YAML Ain't Markup Language, supports most common data types including strings, integers, floats and dates. YAML even supports references and external data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardisation</td>
<td>+</td>
<td>YAML is a strongly typed formal standard, but it's hard to find schema validators.</td>
</tr>
<tr>
<td>Schema-IDL</td>
<td>+-</td>
<td>Partly with Kwalify, Rx and built-in language type defs.</td>
</tr>
<tr>
<td>Language support</td>
<td>+-</td>
<td>There be libraries for the most popular languages.</td>
</tr>
<tr>
<td>Human readability</td>
<td>+</td>
<td>Basic YAML is really easy to read, however YAML's complexity can confuse a reader.</td>
</tr>
<tr>
<td>Speed</td>
<td>--</td>
<td>YAML is very slow to serialise and deserialise.</td>
</tr>
<tr>
<td>File size</td>
<td>+-</td>
<td>YAML is in the medium range similar to JSON and TOML.</td>
</tr>
</tbody>
</table>

Example

CITATION.cff

```yaml
# YAML 1.2
---
cff-version: 1.1.0
message: If you use this software, please cite it as below.
authors:
  - family-names: Druskat
given-names: Stephan
  orcid: https://orcid.org/0000-0003-4925-7248
title: "My Research Software"
version: 2.0.4
doi: 10.5281/zenodo.1234
date-released: 2017-12-18
```

See also:
Other Formats

Apache Avro  A compact and fast binary data format.

   See also:
   • Specification

BSON  Short for Binary JSON. A binary data format mainly for MongoDB

   See also:
   • Specification
   • MongoDB Extended JSON
   • bsondump

Cap’n Proto  A fast data interchange format.

   See also:
   • GitHub

JSON5  A superset of JSON by including strings with multiple lines and character escapes, hexadecimal numbers, comments etc.

   See also:
   • PyPI

HOCON  Short for Human-Optimized Config Object Notation. A JSON superset with comments, multi-line strings etc.

   See also:
   • GitHub
   • Play framework configuration file syntax and features

MessagePack  An efficient binary serialization format supported by Redis scripting.

   See also:
   • Specification
   • GitHub

SDLang  Short for Simple Declarative Language. Textually represent data in a XML-like structure.

   See also:
   • Language Guide
   • GitHub

XDR (RFC 4506)  Short for External Data Representation Standard. Useful for transferring data between different computer architectures.

4.2. Serialisation formats
4.3 Requests

Requests simplifies HTML requests compared to the Python standard library urllib.request.

4.3.1 Requests installation and sample application

Installation

The requests library is useful for communicating with REST APIs. With Spack you can provide requests in your kernel:

```
$ spack env activate python-374
$ spack install py-requests ^python@3.7.4%gcc@9.1.0
```

Alternatively, you can install requests with other package managers, e.g.

```
$ pipenv install requests
```

Example OSM Nomination API

In this example we get our data from the OpenStreetMap Nomination API. This can be reached via the URL https://nominatim.openstreetmap.org/search?...To e.g. receive information about the Berlin Congress Center in Berlin in JSON format, the URL https://nominatim.openstreetmap.org/search.php?q=Alexanderplatz+Berlin&format=json should be given, and if you want to display the corresponding map section you just have to leave out &format=json.

Then we define the base URL and the parameters. Nominatim expects at least the following two parameters

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>Address query that allows the following specifications: street, city, county, state, country and postalcode.</td>
</tr>
<tr>
<td>format</td>
<td>Format in which the data is returned. Possible values are html, xml, json, jsonv2, geojson and geocodejson.</td>
</tr>
</tbody>
</table>

The query can then be made with:

```
[1]: import requests
base_url = 'https://nominatim.openstreetmap.org/search?'
params = {
    'q': 'Alexanderplatz, Berlin',
    'format': 'json',
}
r = requests.get(base_url, params=params)
```

```
[2]: r.status_code
[2]: 200
```

```
[3]: r.json()
```
Three different locations are found, the square, a bus stop and a hotel. In order to be able to filter further, we can only display the most important location:

```python
[4]: params = {
    'q': 'Alexanderplatz, Berlin',
    'format': 'json',
    'limit': '1'
}
```

```python
r = requests.get(base_url, params=params)
r.json()
```

(continues on next page)
Clean Code

Now that we know the code works, let's turn everything into a clean and flexible function.

To ensure that the interaction was successful, we use the `raise_for_status` method of `requests`, which throws an exception if the HTTP status code isn't *200 OK*:

```
[5]: r.raise_for_status()
```

Since we don't want to exceed the load limits of the Nomination API, we will delay our requests with the `time.sleep` function:

```
[6]: from time import sleep

sleep(1)
r.json()
```

Next we declare the function itself. As arguments we need the address, the format, the limit of the objects to be returned with the default value 1 and further `kwargs` (*keyword arguments*) that are passed as parameters:

```
[7]: def nominatim_search(address, format="json", limit=1, **kwargs):
   """Thin wrapper around the Nominatim search API.
   For the list of parameters see
   https://nominatim.org/release-docs/develop/api/Search/#parameters
   """
   search_url = "https://nominatim.openstreetmap.org/search?"
   params = {"q": address, "format": format, "limit": limit, **kwargs}
   r = requests.get(search_url, params=params)
   # Raise an exception if the status is unsuccessful
   r.raise_for_status()
   sleep(1)
   return r.json()
```
Now we can try out the function, e.g. with

```
[8]: nominatim_search('Alexanderplatz, Berlin')
```

```
[8]: [{
      'place_id': 234847916,
      'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright',
      'osm_type': 'relation',
      'osm_id': 131761,
      'boundingbox': ['52.5200695', '52.5232601', '13.4103097', '13.4160798'],
      'lat': '52.521670650000004',
      'lon': '13.413278026558228',
      'display_name': 'Alexanderplatz, Mitte, Berlin, 10178, Deutschland',
      'class': 'highway',
      'type': 'pedestrian',
      'importance': 0.6914982526373583}
```

However, you can use other parameters besides `address`. You can get an overview in the Nominatim Docs.

```
[9]: nominatim_search(address=None, street='8, Marienburger Straße', city='Berlin',
                      country='Germany')
```

```
[9]: [{
      'place_id': 22277694,
      'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright',
      'osm_type': 'node',
      'osm_id': 2270572699,
      'boundingbox': ['52.5346778', '52.5347778', '13.4241714', '13.4242714'],
      'lat': '52.5347278',
      'lon': '13.4242214',
      'display_name': '8, Marienburger Straße, Kollwitzkiez, Prenzlauer Berg, Pankow, Berlin, → 10405, Deutschland',
      'class': 'place',
      'type': 'house',
      'importance': 0.42099999999999993}
```

### Caching

If the same queries are to be asked over and over again within a session, it makes sense to call up this data only once and use it again. In Python we can use `lru_cache` from Python’s standard `functools` library. `lru_cache` saves the last N requests (Least Recent Used) and as soon as the limit is exceeded, the oldest values are discarded. To use this for the `nominatim_search` method, all you have to do is define an import and a decorator:

```
[10]: from functools import lru_cache
```

```
@lru_cache(maxsize=1000)
def nominatim_search(address, format='json', limit=1, **kwargs):
    ...
```

However, `lru_cache` only saves the results during a session. If a script terminates because of a timeout or an exception, the results are lost. If the data is to be saved more permanently, tools such as joblib or python-diskcache can be used.
4.3.2 Create module

It is not very practical to start Jupyter every time and go through all the cells of the request notebook just to be able to use the functions. Instead, we should store our functions in a separate module, like in nominatim.py:

1. For this I have created a new text file in Jupyter in the same place as these notebooks, and named it nominatim.py.
2. Then I copied the imports, the method nominatim_search and its decorator lru_cache and saved the file.
3. Now we can go back to our notebook and import the code from this file and do our searches:

```
[1]: from nominatim import nominatim_search

[2]: nominatim_search('Alexanderplatz, Berlin, Germany')

[2]: [{'place_id': 234847916, 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright', 'osm_type': 'relation', 'osm_id': 131761, 'boundingbox': ['52.5200695', '52.5232601', '13.4103097', '13.4160798'], 'lat': '52.521670650000004', 'lon': '13.413278026558228', 'display_name': 'Alexanderplatz, Mitte, Berlin, 10178, Deutschland', 'class': 'highway', 'type': 'pedestrian', 'importance': 0.6914982526373583}]

[3]: nominatim_search(address=None, street='8, Marienburger Straße', city='Berlin', country='Germany')

[3]: [{'place_id': 22277694, 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright', 'osm_type': 'node', 'osm_id': 2270572699, 'boundingbox': ['52.5346778', '52.5347778', '13.4241714', '13.4242714'], 'lat': '52.5347278', 'lon': '13.4242214', 'display_name': '8, Marienburger Straße, Kollwitzkiez, Prenzlauer Berg, Pankow, Berlin, 10405, Deutschland', 'class': 'place', 'type': 'house', 'importance': 0.42099999999999993}]

The outsourcing of the notebook's code to modules makes it easier to reuse it, and also makes the notebooks more readable.

However, for the code to work, geocode.py needs to be in the same folder as a Jupyter notebook. If you want to call this module from another location, the path specification in the import would have to be changed. In this case it is better to create your own package, as described in Packing.
4.4 BeautifulSoup

1. Install:
   With *Spack* you can provide BeautifulSoup in your kernel:

   ```
   $ spack env activate python-374
   $ spack install py-beautifulsoup4 ^python@3.7.4%gcc@9.1.0
   ```
   
   Alternatively, you can install BeautifulSoup with other package managers, e.g.

   ```
   $ pipenv install beautifulsoup4
   ```

2. With `r.content` we can display the HTML of the page.
3. Next we need to decompose this string into a Python representation of the page using BeautifulSoup:

   ```python
   from bs4 import BeautifulSoup
   soup = BeautifulSoup(r.content, 'html.parser')
   ```

4. To structure the code, let’s create a new function `get_dom` (Document Object Model) that includes all of the preceding code:

   ```python
   def get_dom(url):
       r = requests.get(url)
       r.raise_for_status()
       return BeautifulSoup(r.content, 'html.parser')
   ```

The filtering out of individual elements can be done e.g. via CSS selectors. These can be determined in a website by e.g. Firefox, right-click on one of the table cells in the first column of the table. In the *Inspector* that opens you can click the element again with the right mouse button and then select *Copy → CSS Selector*. The clipboard then contains e.g. `table.wikitable:nth-child(13) > tbody:nth-child(2) > tr:nth-child(1)` or `tbody: nth-child(2) > tr: nth-child(1)`. We are now cleaning up this *CSS-Selector* because we do not want to filter for the 13th child element of table *table.wikitable* or the 2nd child element in *tbody*, but only for the 1st column within *tbody*.

Finally, with `limit=3` in this notebook, we can only display the first three results as an example:

```python
links = soup.select('table.wikitable > tbody > tr > td: nth-child(1) > a', limit=3)
print(links)
```

```python
[<a href="/wiki/Ackerstra%C3%9Fe" title="Ackerstraße">Ackerstraße</a>, <a href="/wiki/ Alexanderplatz" title="Alexanderplatz">Alexanderplatz</a>, <a href="/wiki/Almstadtstra%C3%9Fe" title="Almstadtstraße">Almstadtstraße</a>]
```

However, we don’t want the entire HTML link, just its text content:

```python
for content in links:
    print(content.text)
```
4.4.1 See also:

- Beautiful Soup Documentation

4.5 Intake

Intake makes it easy to find, explore, load, and distribute data. Therefore it is not only interesting for data scientists and engineers, but also for data providers.

See also:

- Docs
- GitHub
- Intake: Taking the Pain out of Data Access
- Intake: Parsing Data from Filenames and Paths
- Intake: Discovering and Exploring Data in a Graphical Interface
- Accessing Remote Data with a Generalized File System
- Intake: Caching Data on First Read Makes Future Analysis Faster

4.5.1 Install Intake

Requirements

Current versions of Bokeh2.0 and Panel must be available in order to use intake_gui.

Installation

Intake can be easily installed for your Jupyter kernel with:

$ pipenv install intake

Create a catalog with sample data

For the following examples we need some data sets that we create with:

$ pipenv run intake example
Creating example catalog...
  Writing us_states.yml
  Writing states_1.csv
  Writing states_2.csv

(continues on next page)
To load the catalog:

```python
>>> import intake
>>> cat = intake.open_catalog('us_states.yml')
```

### 4.5.2 Intake for data scientists

Intake makes it easy to load many different formats and types. For a complete overview, take a look at the [Plugin Directory](#) and the Intake Project Dashboard. Intake then transfers the data to common storage formats such as Pandas DataFrames, Numpy arrays or Python lists. They are then easily searchable and also accessible to distributed systems. If you are missing a plugin, you can also order one yourself, as described in [Making Drivers](#).

#### Load a data source

Hereinafter we will read two csv data records and transfer them to an intake catalog.

```python
[1]: import intake
ds = intake.open_csv('states_*.*')
print(ds)
```

Mit der `open_*`-Funktion von Intake lassen sich verschiedenen Datenquellen einlesen. Je nach Datenformat oder Dienst lassen sich unterschiedliche Argumente verwenden.

#### Configure the search path for data sources

Intake checks the Intake configuration file for `catalog_path` and the environment variable "INTAKE_PATH" for a colon-separated list of paths or semicolons in Windows to look for catalog files. When importing `intake`, all entries from all catalogs that are referenced by `intake.cat` as part of a global catalog are displayed.

#### Read data

Intake reads data in containers of various formats:

- Tables in Pandas DataFrames
- Multi-dimensional arrays in Numpy arrays
- Semi-structured data in Python lists of objects, usually dictionaries

To find out in which container format Intake holds the data, you can use the `container` attribute:

```python
[2]: ds.container
```

```python
[2]: 'dataframe'
```

In addition to `dataframe`, the result can also be `ndarray` or `python`.

```python
[3]: df = ds.read()
df.head()
```
<table>
<thead>
<tr>
<th>state</th>
<th>slug</th>
<th>code</th>
<th>nickname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>alabama</td>
<td>AL</td>
<td>Yellowhammer State</td>
</tr>
<tr>
<td>Alaska</td>
<td>alaska</td>
<td>AK</td>
<td>The Last Frontier</td>
</tr>
<tr>
<td>Arizona</td>
<td>arizona</td>
<td>AZ</td>
<td>The Grand Canyon State</td>
</tr>
<tr>
<td>Arkansas</td>
<td>arkansas</td>
<td>AR</td>
<td>The Natural State</td>
</tr>
<tr>
<td>California</td>
<td>california</td>
<td>CA</td>
<td>Golden State</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>website</th>
<th>admission_date</th>
<th>admission_number</th>
<th>capital_city</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.alabama.gov">http://www.alabama.gov</a></td>
<td>1819-12-14</td>
<td>22</td>
<td>Montgomery</td>
</tr>
<tr>
<td><a href="http://alaska.gov">http://alaska.gov</a></td>
<td>1959-01-03</td>
<td>49</td>
<td>Juneau</td>
</tr>
<tr>
<td><a href="https://az.gov">https://az.gov</a></td>
<td>1912-02-14</td>
<td>48</td>
<td>Phoenix</td>
</tr>
<tr>
<td><a href="http://www.ca.gov">http://www.ca.gov</a></td>
<td>1850-09-09</td>
<td>31</td>
<td>Sacramento</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>capital_url</th>
<th>population</th>
<th>population_rank</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.montgomeryal.gov">http://www.montgomeryal.gov</a></td>
<td>4833722</td>
<td>23</td>
</tr>
<tr>
<td><a href="http://www.juneau.org">http://www.juneau.org</a></td>
<td>735132</td>
<td>47</td>
</tr>
<tr>
<td><a href="https://www.phoenix.gov">https://www.phoenix.gov</a></td>
<td>6626624</td>
<td>15</td>
</tr>
<tr>
<td><a href="http://www.littlerock.org">http://www.littlerock.org</a></td>
<td>2959373</td>
<td>32</td>
</tr>
<tr>
<td><a href="http://www.cityofsacramento.org">http://www.cityofsacramento.org</a></td>
<td>38332521</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>constitution_url</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://alisondb.legislature.state.al.us/alison">http://alisondb.legislature.state.al.us/alison</a>...</td>
</tr>
<tr>
<td><a href="http://www.legis.state.ak.us/basis/folioproxy">http://www.legis.state.ak.us/basis/folioproxy</a>...</td>
</tr>
<tr>
<td><a href="http://www.azleg.gov/Constitution.asp">http://www.azleg.gov/Constitution.asp</a></td>
</tr>
<tr>
<td><a href="http://www.arkleg.state.ar.us/assembly/Summary">http://www.arkleg.state.ar.us/assembly/Summary</a>...</td>
</tr>
<tr>
<td><a href="http://www.leginfo.ca.gov/const-toc.html">http://www.leginfo.ca.gov/const-toc.html</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>state_flag_url</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://cdn.civil.services/us-states/flags/ala">https://cdn.civil.services/us-states/flags/ala</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/flags/ala">https://cdn.civil.services/us-states/flags/ala</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/flags/ari">https://cdn.civil.services/us-states/flags/ari</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/flags/ark">https://cdn.civil.services/us-states/flags/ark</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/flags/cal">https://cdn.civil.services/us-states/flags/cal</a>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>state_seal_url</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://cdn.civil.services/us-states/seals/ala">https://cdn.civil.services/us-states/seals/ala</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/seals/ala">https://cdn.civil.services/us-states/seals/ala</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/seals/ari">https://cdn.civil.services/us-states/seals/ari</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/seals/ark">https://cdn.civil.services/us-states/seals/ark</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/seals/cal">https://cdn.civil.services/us-states/seals/cal</a>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>map_image_url</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://cdn.civil.services/us-states/maps/alab">https://cdn.civil.services/us-states/maps/alab</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/maps/ala">https://cdn.civil.services/us-states/maps/ala</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/maps/ariz">https://cdn.civil.services/us-states/maps/ariz</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/maps/arka">https://cdn.civil.services/us-states/maps/arka</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/maps/cali">https://cdn.civil.services/us-states/maps/cali</a>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>landscape_background_url</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://cdn.civil.services/us-states/background">https://cdn.civil.services/us-states/background</a>...</td>
</tr>
<tr>
<td><a href="https://cdn.civil.services/us-states/background">https://cdn.civil.services/us-states/background</a>...</td>
</tr>
<tr>
<td>(continues on next page)</td>
</tr>
</tbody>
</table>
for chunk in ds.read_chunked(): print('Chunk: %d' % len(chunk))

Chunk: 24
Chunk: 26

ddf = ds.to_dask()
ddf.head()

state  slug code  nickname
Alabama  alabama  AL Yellowhammer State
Alaska  alaska  AK The Last Frontier
Arizona  arizona  AZ The Grand Canyon State
Arkansas  arkansas  AR The Natural State
California  california  CA Golden State

website admission_date admission_number capital_city
http://www.alabama.gov  1819-12-14  22 Montgomery
http://alaska.gov  1959-01-03  49 Juneau
https://az.gov  1912-02-14  48 Phoenix
http://www.ca.gov  1850-09-09  31 Sacramento

capital_url population population_rank
http://www.montgomeryal.gov  4833722  23
http://www.juneau.org  735132  47
https://www.phoenix.gov  6626624  15
http://www.littlerock.org  2959373  32
http://www.cityofsacramento.org  38332521  1

4.5. Intake
[6]: cat = intake.open_catalog('us_states.yml')

[7]: list(cat)

[7]: ['states']

[8]: cat.states.to_dask()['state','slug'].head()

<table>
<thead>
<tr>
<th></th>
<th>state</th>
<th>slug</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Alabama</td>
<td>alabama</td>
</tr>
<tr>
<td>1</td>
<td>Alaska</td>
<td>alaska</td>
</tr>
<tr>
<td>2</td>
<td>Arizona</td>
<td>arizona</td>
</tr>
<tr>
<td>3</td>
<td>Arkansas</td>
<td>arkansas</td>
</tr>
<tr>
<td>4</td>
<td>California</td>
<td>california</td>
</tr>
</tbody>
</table>

[9]: cat.states(csv_kwars={'header': None, 'skiprows': 1}).read().head()

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Alabama</td>
<td>alabama</td>
<td>AL</td>
<td>Yellowhammer State</td>
<td><a href="http://www.alabama.gov">http://www.alabama.gov</a></td>
</tr>
<tr>
<td>1</td>
<td>Alaska</td>
<td>alaska</td>
<td>AK</td>
<td>The Last Frontier</td>
<td><a href="http://alaska.gov">http://alaska.gov</a></td>
</tr>
<tr>
<td>2</td>
<td>Arizona</td>
<td>arizona</td>
<td>AZ</td>
<td>The Grand Canyon State</td>
<td><a href="https://az.gov">https://az.gov</a></td>
</tr>
<tr>
<td>3</td>
<td>Arkansas</td>
<td>arkansas</td>
<td>AR</td>
<td>The Natural State</td>
<td><a href="http://arkansas.gov">http://arkansas.gov</a></td>
</tr>
<tr>
<td>4</td>
<td>California</td>
<td>california</td>
<td>CA</td>
<td>Golden State</td>
<td><a href="http://www.ca.gov">http://www.ca.gov</a></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1819-12-14</td>
<td>22</td>
<td>Montgomery</td>
<td><a href="http://www.montgomeryal.gov">http://www.montgomeryal.gov</a></td>
<td>4833722</td>
</tr>
<tr>
<td>1</td>
<td>1959-01-03</td>
<td>49</td>
<td>Juneau</td>
<td><a href="http://www.juneau.org">http://www.juneau.org</a></td>
<td>735132</td>
</tr>
<tr>
<td>2</td>
<td>1912-02-14</td>
<td>48</td>
<td>Phoenix</td>
<td><a href="https://www.phoenix.gov">https://www.phoenix.gov</a></td>
<td>6626624</td>
</tr>
<tr>
<td>4</td>
<td>1850-09-09</td>
<td>31</td>
<td>Sacramento</td>
<td><a href="http://www.cityofsacramento.org">http://www.cityofsacramento.org</a></td>
<td>38332521</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td><a href="http://alisondb.legislature.state.al.us/alison">http://alisondb.legislature.state.al.us/alison</a>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td><a href="http://www.legis.state.ak.us/basis/folioproxy">http://www.legis.state.ak.us/basis/folioproxy</a>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><a href="http://www.azleg.gov/Constitution.asp">http://www.azleg.gov/Constitution.asp</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><a href="http://www.arkleg.state.ar.us/assembly/Summary">http://www.arkleg.state.ar.us/assembly/Summary</a>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><a href="http://www.leginfo.ca.gov/const-toc.html">http://www.leginfo.ca.gov/const-toc.html</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td><a href="https://cdn.civil.services/us-states/flags/ala">https://cdn.civil.services/us-states/flags/ala</a>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td><a href="https://cdn.civil.services/us-states/flags/ala">https://cdn.civil.services/us-states/flags/ala</a>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><a href="https://cdn.civil.services/us-states/flags/ari">https://cdn.civil.services/us-states/flags/ari</a>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><a href="https://cdn.civil.services/us-states/flags/ark">https://cdn.civil.services/us-states/flags/ark</a>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><a href="https://cdn.civil.services/us-states/flags/cal">https://cdn.civil.services/us-states/flags/cal</a>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td><a href="https://cdn.civil.services/us-states/seals/ala">https://cdn.civil.services/us-states/seals/ala</a>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)
4.5.3 Intake-GUI: Exploring data in a graphical user interface

Intake GUI has been re-implemented so that it can be made available not only in Jupyter notebooks, but also in other web applications. It displays the contents of all installed catalogs and enables local and remote catalogs to be selected and to be searched and selected from.

Intake supports the division of labor between data engineers who curate, manage, and deploy data, and data scientists who analyse and visualise data without having to know how it’s stored.

The Intake GUI is based on Panel, with the control panel offering a composite dashboard solution for displaying plots, images, tables, texts and widgets. Panel works both in a Jupyter notebook and in a standalone Tornado application.

From a data engineer’s point of view, this means that you can deploy the recording GUI at an endpoint and use it as
a data exploration tool for your data users. This also means that it’s easy to adapt and reorganise the GUI in order to
insert your own logo, reuse parts of it in your own applications or add new functions.
In the future, Intake-GUI should also allow the input of user parameters as well as the editing and saving of catalogs.

```python
import intake
intake.gui
```

The GUI contains three main areas:

1. a list of catalogs. The `builtin` catalog shown by default contains data records installed in the system, just like `intake.cat`.
2. a list of the sources in the currently selected catalog.
3. a description of the currently selected source.
Ad 1: Catalogs

No catalog is currently displayed in the list of catalogs. However, under the three main areas there are three buttons that can be used to add, remove, or search catalogs.

The buttons are also available through the API, e.g. for Add Catalog with:

```
[2]: intake.gui.add('./us_crime/us_crime.yaml')
```

Remote catalogs are e.g. available under

- https://s3.amazonaws.com/earth-data/UCMerced_LandUse/catalog.yml

Ad 2. Sources

Selecting a source from the list updates the descriptive text on the left side of the user interface.

This is also available via the API:

```
[3]: intake.gui.sources
```

```
[3]: [name: us_crime
    container: dataframe
    plugin: ['csv']
    →StatebyState.cfm)
    direct_access: forbid
    user_parameters: []
    metadata:
        plots:
            line_example:
                kind: line
                y: ['Robbery', 'Burglary']
                x: Year
            violin_example:
                kind: violin
                y: ['Burglary rate', 'Larceny-theft rate', 'Robbery rate', 'Violent Crime rate']
                group_label: Type of crime
                value_label: Rate per 100k
                invert: True
        args:
            urlpath: {{ CATALOG_DIR }}/data/crime.csv]
```

This consists of a list of regular Intake data source entries. To look at the first entries, we can enter the following:

```
[4]: source = intake.gui.sources[0]
source.to_dask().head()
```

```
   textasciitildeYear  Population  Violent crime total
0  1960        179323175         288460
1  1961        182992000         289390
2  1962        185771000         301510
```

(continues on next page)
### Murder and nonnegligent Manslaughter Legacy rape /1 Revised rape /2

<table>
<thead>
<tr>
<th></th>
<th>1963</th>
<th>188483000</th>
<th>316970</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8740</td>
<td>17550</td>
<td>NaN</td>
</tr>
<tr>
<td>2</td>
<td>8530</td>
<td>17220</td>
<td>NaN</td>
</tr>
<tr>
<td>3</td>
<td>8640</td>
<td>17190</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>9360</td>
<td>21420</td>
<td>NaN</td>
</tr>
</tbody>
</table>

### Robbery Aggravated assault Property crime total Burglary ...

<table>
<thead>
<tr>
<th></th>
<th>107840</th>
<th>154320</th>
<th>3095700</th>
<th>912100</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>106670</td>
<td>156760</td>
<td>3198600</td>
<td>949600</td>
</tr>
<tr>
<td>1</td>
<td>110860</td>
<td>164570</td>
<td>3450700</td>
<td>994300</td>
</tr>
<tr>
<td>2</td>
<td>116470</td>
<td>174210</td>
<td>3792500</td>
<td>1086400</td>
</tr>
<tr>
<td>3</td>
<td>130390</td>
<td>203050</td>
<td>4200400</td>
<td>1213200</td>
</tr>
</tbody>
</table>

### Violent Crime rate Murder and nonnegligent manslaughter rate

<table>
<thead>
<tr>
<th></th>
<th>160.9</th>
<th>5.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>158.1</td>
<td>4.8</td>
</tr>
<tr>
<td>1</td>
<td>162.3</td>
<td>4.6</td>
</tr>
<tr>
<td>2</td>
<td>168.2</td>
<td>4.6</td>
</tr>
<tr>
<td>3</td>
<td>190.6</td>
<td>4.9</td>
</tr>
</tbody>
</table>

### Legacy rape rate /1 Revised rape rate /2 Robbery rate

<table>
<thead>
<tr>
<th></th>
<th>9.6</th>
<th>NaN</th>
<th>60.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9.4</td>
<td>NaN</td>
<td>58.3</td>
</tr>
<tr>
<td>1</td>
<td>9.4</td>
<td>NaN</td>
<td>59.7</td>
</tr>
<tr>
<td>2</td>
<td>9.4</td>
<td>NaN</td>
<td>61.8</td>
</tr>
<tr>
<td>3</td>
<td>11.2</td>
<td>NaN</td>
<td>68.2</td>
</tr>
</tbody>
</table>

### Aggravated assault rate Property crime rate Burglary rate

<table>
<thead>
<tr>
<th></th>
<th>86.1</th>
<th>1726.3</th>
<th>508.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>85.7</td>
<td>1747.9</td>
<td>518.9</td>
</tr>
<tr>
<td>1</td>
<td>88.6</td>
<td>1857.5</td>
<td>535.2</td>
</tr>
<tr>
<td>2</td>
<td>92.4</td>
<td>2012.1</td>
<td>576.4</td>
</tr>
<tr>
<td>3</td>
<td>106.2</td>
<td>2197.5</td>
<td>634.7</td>
</tr>
</tbody>
</table>

### Larceny-theft rate Motor vehicle theft rate

<table>
<thead>
<tr>
<th></th>
<th>1034.7</th>
<th>183.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1045.4</td>
<td>183.6</td>
</tr>
<tr>
<td>1</td>
<td>1124.8</td>
<td>197.4</td>
</tr>
<tr>
<td>2</td>
<td>1219.1</td>
<td>216.6</td>
</tr>
<tr>
<td>3</td>
<td>1315.5</td>
<td>247.4</td>
</tr>
</tbody>
</table>

[5 rows x 22 columns]
Data type cannot be displayed: application/javascript, application/vnd.holoviews_load.v0+json

[5]: Row(name='Source')

[6]: intake.gui.source.description

[6]: Column(height=240, name='Description', width_policy='max')
  [0] Markdown(str, max_height=40)
  [1] Str(str, css_classes=['scrolling'], height=200, sizing_mode='stretch_width')

[7]: cat = intake.open_catalog('./us_crime/us_crime.yaml')
cat.gui

[7]: Row
  [0] Column(margin=(25, 0, 0, 0), width=50)
  [0] PNG(str, align='center')
  [1] Column(max_width=1600, name='Entries', width_policy='max')
    [0] Row
      [0] Column
        [0] Column(name='Select Data Source')
          [0] Markdown(str, max_height=40)
          [1] MultiSelect(min_width=600, options=OrderedDict([('us_crime', ...))], size=9, value=[name: us_crime container: ...], width_policy='min')
      [1] Row(name='Controls')
        [0] Toggle(name='', width=50)
        [1] Toggle(disabled=True, name='', width=50)
    [1] Column(height=240, name='Description', width_policy='max')
      [0] Markdown(str, max_height=40)
      [1] Str(str, css_classes=['scrolling'], height=200, sizing_mode='stretch_width')
        [1] Column(name='Plot', width_policy='max')

[8]: us_crime = cat.gui.sources[0]

[9]: intake.output_notebook()

    us_crime.plot.bivariate('Burglary rate', 'Property crime rate', legend=False, width=500, height=400) * \
    us_crime.plot.scatter('Burglary rate', 'Property crime rate', color='black', size=15, legend=False) + \
    us_crime.plot.table(['Burglary rate', 'Property crime rate'], width=350, height=350)

Data type cannot be displayed: application/javascript, application/vnd.holoviews_load.v0+json

Data type cannot be displayed: application/javascript, application/vnd.holoviews_load.v0+json
Ad 3. Source view

As soon as catalogs are loaded and the desired sources have been selected, they are available under the attribute `intake.gui.sources`. Each source entry has methods and can be opened as a data source like any catalog entry. For `Source: UCMerced_LandUse_by_landuse`, the entry looks like this:

```
name: UCMerced_LandUse_by_landuse
container: None
plugin: []
description: All images matching given landuse from UCMerced_LandUse/Image.
direct_access: forbid
user_parameters: [{'name': 'landuse', 'description': 'which landuse to collect', 'type': ...
storage_options:
    anon: True
    concat_dim: id
    coerce_shape: [256, 256]
```

Below the list of sources there is a series of buttons for opening up the selected data source: `Plot` opens a sub-window to display the predefined (i.e. the ones specified in yaml) plots for the selected source.

See also:

- GUI

### 4.5.4 Intake for data engineers

Intake supports data engineers with the provision of data and with the specification of the data sources, the distribution of the data, the parameterisation of the user options etc. This makes it easier for data scientists to access the data afterwards, as the possible options are already specified in the catalog.

```
[1]: import intake
    import hvplot.pandas
    intake.output_notebook()
```

Data type cannot be displayed: application/javascript, application/vnd.holoviews_load.v0+json
Intake data sets are loaded with so-called drivers, some come with the intake package, but others have to be reloaded as plug-ins. You can display the available drivers as follows:

```
[2]: list(intake.registry)
[2]: ['yaml_file_cat', 'yaml_files_cat', 'catalog', 'csv', 'intake_remote', 'ndzarr', 'numpy', 'textfiles']
```

Each of these drivers is assigned an `intake.open_*` function. It is also possible to refer to drivers by the fully qualified name (e.g. `package.submodule.DriverClass`). In the following example, however, we will focus on the `csv` driver that is included in the standard Intake installation.

In general, the first step in writing a catalog entry is to use the appropriate `open_*` function to create a `DataSource` object:

```
```

The above specification has now created a `DataSource` object, but has not yet checked whether the data can actually be accessed. To test whether the loading was really successful, the source itself can be opened (`source.discover`) or read (`source.read`):

```
[4]: source.discover()
[4]: {'datashape': None, 'dtype': {'Time': 'object', 'Arctic': 'float64', 'Antarctica': 'float64'}, 'shape': (None, 3), 'npartitions': 1, 'metadata': {}}
```

```
[5]: df = source.read()
df.head()
```

```
<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Arctic</th>
<th>Antarctica</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1990M01</td>
<td>12.72</td>
<td>3.27</td>
</tr>
<tr>
<td>1</td>
<td>1990M02</td>
<td>13.33</td>
<td>2.15</td>
</tr>
<tr>
<td>2</td>
<td>1990M03</td>
<td>13.44</td>
<td>2.71</td>
</tr>
</tbody>
</table>
```
After we have determined that the data can be loaded as desired, we want to open up the data visually:

```python
[6]: df.hvplot(kind='line', x='Time', y=['Arctic', 'Antarctica'],
             width=700, height=500)
```

Now we can load a source correctly and also receive a graphic output for opening up the data. We can now display this recipe in the YAML syntax with:

```python
[8]: print(source.yaml())
```

Finally, we can create a YAML file containing this recipe with an additional description and the tested diagram:

```bash
[10]: %%writefile sea.yaml
```

```
sources:
  sea_ice:
    args:
      description: "Polar sea ice cover"
      driver: csv
metadata:
  plots:
    basic:
      kind: line
      x: Time
      y: [Arctic, Antarctica]
      width: 700
      height: 500
```

To check that the YAML file works too, we can reload it and try to work with it:

```python
[11]: cat = intake.open_catalog('sea.yaml')
[12]: cat.sea_ice.plot.basic()
```

```
:NdOverlay [Variable]
:Curve [Time] (value)
```
The catalog appears to be functional and can now be released. The easiest way to share an Intake catalog is to put it in a place where it can be read by your target audience. In this tutorial stored in a Git repo, this can be the url of the file in the repo. All you have to share with your users is the URL of the catalog. You can try this yourself with:

```python
```

```python
[14]: cat.sea_ice.read().head()
```

<table>
<thead>
<tr>
<th>Time</th>
<th>Arctic</th>
<th>Antarctica</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1990M01</td>
<td>12.72</td>
<td>3.27</td>
</tr>
<tr>
<td>1 1990M02</td>
<td>13.33</td>
<td>2.15</td>
</tr>
<tr>
<td>2 1990M03</td>
<td>13.44</td>
<td>2.71</td>
</tr>
<tr>
<td>3 1990M04</td>
<td>12.16</td>
<td>5.10</td>
</tr>
<tr>
<td>4 1990M05</td>
<td>10.84</td>
<td>7.37</td>
</tr>
</tbody>
</table>

**Note:**

This catalog is also a DataSource instance, i.e. you can refer to it from other catalogs and thus build a hierarchy of data sources. For example, you have a master or main catalog that references several other catalogs, each with entries of a certain type and the whole thing can e.g. be searched with `Intake-GUI`. In this way, the overall data acquisition structure has a structure that makes it easier to navigate to the correct data set. You can even have separate hierarchies that reference the same data.

```python
[15]: print(cat.yaml())
```

```
sources:
  sea:
    args:
      path: https://raw.githubusercontent.com/veit/jupyter-tutorial/master/docs/gather/intake/sea.yaml
      description: '
      driver: intake.catalog.local.YAMLFileCatalog
      metadata: {}
```

## 4.6 PostgreSQL

### 4.6.1 Basic functions

**ACID compliant** ACID (A tomicity, C onsistency, I solation, D urability) is a series of properties that database transactions should fulfill to guarantee the validity of the data even in the event of a fault.

**SQL:2011** Temporal_tables [https://github.com/arkhipov/temporal_tables] also meet the SQL standard ISO/IEC 9075:2011, including:

- Time period definitions
- Valid time tables
- Transaction time tables (system-versioned tables) with time-sliced and sequenced queries

**Data types** The following data types are supported out of the box:
• primitive data types: Integer, Numeric, String, Boolean
• structured data types: Date/Time, Array, Range, UUID
• document types: JSON/JSONB, XML, key-value (Hstore)
• geometric data types: point, line, circle, polygon
• adjustments: composite, custom Types
• transactional data definition language (DDL)

Transactional DDL is implemented via write-ahead logging. Big changes are also possible, but not adding and dropping databases and tables:

```
$ psql mydb
mydb=# DROP TABLE IF EXISTS foo;
NOTICE: table "foo" does not exist
DROP TABLE
mydb=# BEGIN;
BEGIN
mydb=# CREATE TABLE foo (bar int);
CREATE TABLE
mydb=# INSERT INTO foo VALUES (1);
INSERT 0 1
mydb=# ROLLBACK;
ROLLBACK
mydb=# SELECT * FROM foo;
ERROR: relation "foo" does not exist
```

### Concurrent Index

PostgreSQL can create indexes without having to lock write access to tables.

See also:

Building Indexes Concurrently

### Extensions

PostgreSQL can easily be extended. The contrib/ directory supplied with the source code contains various extensions that are described in Appendix F. Other extensions have been developed independently, such as PostGIS or Slony-I.

### Common Table Expression

WITH Queries (Common Table Expressions) divides complex queries into simpler queries, e.g.:

```sql
WITH regional_insolation AS (
    SELECT region, SUM(amount) AS total_insolation
    FROM orders
    GROUP BY region
), top_regions AS (
    SELECT region
    FROM regional_insolation
    WHERE total_insolation > (SELECT SUM(total_insolation)/10 FROM regional_insolation)
)
```

There is also a RECURSIVE modifier that refers the WITH query to its own output. The following is an example of how to sum the numbers from 1 to 100:

```
WITH RECURSIVE t (n) AS (
    VALUE (1)
) 
```
UNION ALL
    SELECT n + 1 FROM t WHERE <100
)
SELECT sum (n) FROM t;

Multi-Version Concurrency Control (MVCC)  Multi-Version Concurrency Control allows two or more sessions to access the same data at the same time without compromising the integrity of the data.

Cross platform  PostgreSQL runs on common CPU architectures such as x86, PowerPC, Sparc, ARM, MIPS or PA-RISC. Most operating systems are also supported: Linux, Windows, FreeBSD, OpenBSD, NetBSD, Mac OS, AIX, HP/UX and Solaris.

See also:
explain.depesz.com  Web app that visualises PostgreSQL’s EXPLAIN and ANALYZE statements.

Foreign Data Wrappers (FDW)

In 2003, SQL was expanded to include SQL/MED (SQL Management of External Data). PostgreSQL 9.1 supports this read-only, 9.3 then also write. Since then, a number of Foreign Data Wrappers (FDW) have been developed for PostgreSQL.

The following is just a small selection of the best-known FDWs:

Note: Most of these wrappers are not officially supported by the PostgreSQL Global Development Group (PGDG).

Generic SQL wrappers

ODBC  Native ODBC FDW for PostgreSQL 9.5

• GitHub

Multicorn  Multicorn makes it easy to develop FDWs. For example, SQLAlchemy uses Multicorn to save your data in PostgreSQL.

• GitHub
• PGXN
• Docs

VirtDB  Native access to VirtDB (SAP ERP, Oracle RDBMS)

• GitHub
Specific SQL wrappers

**postgres_fdw**  With *postgres_fdw* data from other PostgreSQL servers can be accessed.
  
  - Git
  - PGXN
  - Docs

**Oracle**  FDW for Oracle databases
  
  - GitHub
  - PGXN
  - Docs

**MySQL**  FDW for MySQL from PostgreSQL9.3
  
  - GitHub
  - PGXN

**SQLite**  FDW for SQLite3
  
  - GitHub
  - PGXN
  - Docs

NoSQL database wrappers

**Cassandra**  FDW für Cassandra
  
  - GitHub
  - rankactive

**Neo4j**  FWD for Neo4j, which also provides a cypher function for PostgreSQL
  
  - GitHub
  - Docs

**Redis**  FDW for Redis
  
  - GitHub

**Riak**  FDW for Riak
  
  - GitHub
File wrappers

CSV  Official extension for PostgreSQL 9.1
    • Git
    • Docs

JSON  FDW for JSON files
    • GitHub
    • Example

XML  FDW for XML files
    • GitHub
    • PGXN

Geo wrappers

GDAL/OGR  FDW for the GDAL/OGR driver including databases like Oracle and SQLite as well as file formats like MapInfo, CSV, Excel, OpenOffice, OpenStreetMap PBF and XML.
    • GitHub

Geocode/GeoJSON  A collection of FDWs for PostGIS
    • GitHub

Open Street Map PBF  FDW for Open Street Map PBF
    • GitHub

Generic web wrappers

ICAL  FDW for ICAL
    • GitHub
    • Docs

IMAP  FDW for the Internet Message Access Protocol (IMAP)
    • Docs

RSS  FDQ for RSS feeds
    • Docs

See also:
    • PostgreSQL wiki
    • PGXN website
Procedural programming languages

With PostgreSQL, user-defined functions can be written in languages other than SQL and C.

There are currently four procedural languages available in the standard PostgreSQL distribution:

- PL/pgSQL
- PL/Tcl
- PL/Perl
- PL/Python

Additional procedural programming languages are available but are not included in the core distribution:

- PL/Java
- PL/Lua
- PL/R
- PL/sh
- PL/v8

See also:

External Procedural Languages

In addition, other languages can be defined, see also Writing A Procedural Language Handler.

DB-API 2.0

The Python API for database connectors is easy to use and understand. The two main concepts are:

Connection  Connection Objects allow the following methods:

- `connect(parameters...)` opens the connection to the database
- `.close()` closes the connection to the database
- `.commit()` transfers the outstanding transaction to the database
- `.rollback()` This method is optional as not all databases allow transactions to be rolled back.
- `.cursor()` Return of a new cursor object via the connection.

Example:

```python
import driver

conn = driver.connect(database='example',
                      host='localhost',
                      port=5432)

try:
    # create the cursor
    # use the cursor
except Exception:
    conn.rollback()
else:
    conn.commit()
    conn.close()
```
**Cursor**  
Cursor objects are used to manage the context of a `.fetch*()` method.

Curators that are created in the same connection are not isolated from one another.

There are two attributes for cursor objects:

- `.description` contains the following seven elements:
  1. name
  2. type_code
  3. display_size
  4. internal_size
  5. precision
  6. scale
  7. null_ok

The first two elements (name and type_code) are mandatory, the other five are optional and are set to `None` if no meaningful values can be specified.

- `.rowcount` indicates the number of lines that the last call of `.execute*()` with SELECT, UPDATE or INSERT resulted in.

Example:

```python
cursor = conn.cursor()
cursor.execute(""
    SELECT column1, column2
    FROM tableA
""
)
for column1, column2 in cursor.fetchall():
    print(column1, column2)
```

See also:

- PEP 249 – Python Database API Specification v2.0

**Psycopg**

Psycopg is a PostgreSQL adapter based on the C library for PostgreSQL `libpq`. Among other things, it offers:

- DB API 2.0 compatibility
- Multithreading with thread safety
- Connections pooling to be able to use a cache of existing database connections for queries.
- Asynchronous and Coroutines support
- Adaptation of the Python types in SQL
Install

With Spack you can provide psycopg2 in your kernel, e.g. with

```
$ spack env activate python-374
$ spack install py-psycopg2 ^python@3.7.4
```

Object-relational mapping

«Object-relational mapping (...) in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages.»

In the simplest case, classes are mapped to tables, with each object corresponding to a table row and each attribute to a table column.

There are essentially three different methods of mapping inheritance hierarchies:

- **Single Table** One table is created for each inheritance hierarchy, with all attributes of the base class and all classes derived from it being stored in a common table.
- **Joined Table or Class Table** A table is created for each subclass and a further table for each subclass derived from it.
- **Table per Class or Concrete Table** The attributes of the abstract base class are included in the tables for the specific subclasses. However, it is not possible to determine instances of different classes with one query.

SQLAlchemy

SQLAlchemy is a Python-SQL-Toolkit and object-relational mapper.

SQLAlchemy is known for its ORM, whereby it provides different patterns for object-relational mapping, whereby classes can be mapped to the database in different ways. The object model and the database schema are cleanly decoupled from the start.

SQLAlchemy differs fundamentally from other ORMs, as SQL and details of the object relation are not abstracted away: all processes are represented as a collection of individual tools.

Database connection

```python
from sqlalchemy import create_engine
engine = create_engine('postgresql:///example', echo=True)
```

Data model

```python
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship

Base = declarative_base()

class Address(Base):
    (continues on next page)
```

---

1. Wikipedia: relational mapping
__tablename__ = 'address'

    id = Column(Integer, primary_key=True)
    street = Column(String)
    zipcode = Column(String)
    country = Column(String, nullable=False)

class Contact(Base):
    __tablename__ = 'contact'

    id = Column(Integer, primary_key=True)

    firstname = Column(String, nullable=False)
    lastname = Column(String, nullable=False)
    email = Column(String, nullable=False)
    address_id = Column(Integer, ForeignKey(Address.id), nullable=False)
    address = relationship('Address')

Create tables

Base.metadata.create_all(engine)

Create Session

session = Session(engine)
address = Address(street='Birnbaumweg 10', zipcode='79115', country='Germany')

contact = Contact(
    firstname='Veit', lastname='Schiele',
    email='veit@cusy.io',
    address=address
)

session.add(contact)
session.commit()

Read

contact =
session.query(Contact).filter_by(email='veit@cusy.io').first()
print(contact.firstname)

contacts = session.query(Contact).all()
for contact in contacts:
    print(contact.firstname)

contacts =
session.query(Contact).filter_by(email='veit@cusy.io').all()
for contact in contacts:
    print(contact.firstname)

Update

contact = session.query(Contact) \n    .filter_by(email='veit@cusy.io').first()
contact.email = 'info@veit-schiele.de'
session.add(contact)
session.commit()

Delete

contact = session.query(Contact) \n    .filter_by(email='info@veit-schiele.de').first()
session.delete(contact)
session.commit()

Alembic

Alembic is based on SQLAlchemy and serves as a database migration tool with the following functions:
• ALTER statements to a database to change the structure of tables and other constructs
• System for creating migration scripts. Optionally, the sequence of steps for the downgrade can also be specified.
• The scripts are executed in a specific order.

Create migration environment

The Migration Environment is a directory that is specific to a particular application. It is created with the Alembic ini command and then managed along with the application’s source code.

$ cd myrproject
$ alembic init alembic
Creating directory /path/to/myproject/alembic...done
Creating directory /path/to/myproject/alembic/versions...done
Generating /path/to/myproject/alembic.ini...done
Generating /path/to/myproject/alembic/env.py...done
Generating /path/to/myproject/alembic/README...done
Generating /path/to/myproject/alembic/script.py.mako...done
Please edit configuration/connection/logging settings in /
'/path/to/myproject/alembic.ini' before proceeding.

The structure of such a migration environment can e.g. look like this:
**Templates**

Alembic includes a number of templates that can be displayed with list:

```bash
$ alembic list_templates
Available templates:

generic - Generic single-database configuration.
multidb - Rudimentary multi-database configuration.
pylons - Configuration that reads from a Pylons project environment.
```

Templates are used via the 'init' command, e.g.:

```
alembic init --template pylons ./scripts
```

**Configure ini file**

The file created with the generic template looks like this:

```ini
# A generic, single database configuration.

[alembic]
# path to migration scripts
script_location = alembic

# template used to generate migration files
file_template = %%(rev)s_%%(slug)s

# timezone to use when rendering the date
# within the migration file as well as the filename.
# string value is passed to dateutil.tz.gettz()
# leave blank for localtime
timezone =

# max length of characters to apply to the
# "slug" field
truncatSlug_length = 40

# set to 'true' to run the environment during
```

(continues on next page)
# the 'revision' command, regardless of autogenerate
# revision_environment = false

# set to 'true' to allow .pyc and .pyo files without
# a source .py file to be detected as revisions in the
# versions/ directory
# sourceless = false

# version location specification; this defaults
# to alembic/versions. When using multiple version
# directories, initial revisions must be specified with --version-path
# version_locations = %%(here)s/bar %%(here)s/bat alembic/versions

# the output encoding used when revision files
# are written from script.py.mako
# output_encoding = utf-8

sqlalchemy.url = driver://user:pass@localhost/dbname

# Logging configuration
[loggers]
keys = root,sqlalchemy,alembic

[handlers]
keys = console

[formatters]
keys = generic

[logger_root]
level = WARN
handlers = console
qualname =

[logger_sqlalchemy]
level = WARN
handlers =
qualname = sqlalchemy.engine

[logger_alembic]
level = INFO
handlers =
qualname = alembic

[handler_console]
class = StreamHandler
args = (sys.stderr,)
level = NOTSET
formatter = generic

[formatter_generic]
format = %(levelname)-5.5s [%(name)s] %(message)s
Jupyter Tutorial, Release 0.8.0

(continued from previous page)

datefmt = %H:%M:%S
%(here)s Replacement variable for creating absolute paths
file_template This is the naming scheme used to generate new migration files. The available variables include:
%%(rev)s Revision ID
%%(slug)s Abbreviated revision message
%%(year)d, %%(month).2d, %%(day).2d, %%(hour).2d, %%(minute).2d, %%(second).2d Creation time
Create a migration script
A new revision can be created with:
$ alembic revision -m "create account table"
Generating /path/to/yourproject/alembic/versions/1975ea83b712_create_account_table.py...
˓→done
Then the file 1975ea83b712_create_account_table.py looks like this:
"""create account table
Revision ID: 1975ea83b712
Revises:
Create Date: 2018-12-08 11:40:27.089406
"""
# revision identifiers, used by Alembic.
revision = '1975ea83b712'
down_revision = None
branch_labels = None
from alembic import op
import sqlalchemy as sa
def upgrade():
pass
def downgrade():
pass
down_revision Variable that tells Alembic in which order the migrations should be carried out, e.g.:
# revision identifiers, used by Alembic.
revision = 'ae1027a6acf'
down_revision = '1975ea83b712'
upgrade, downgrade e.g.:

184

Chapter 4. Read, persist and provide data


def upgrade():
    op.create_table('account',
        sa.Column('id', sa.Integer, primary_key=True),
        sa.Column('name', sa.String(50), nullable=False),
        sa.Column('description', sa.Unicode(200)),
    )

def downgrade():
    op.drop_table('account')

create_table() and drop_table() are Alembic directives. You can get an overview of all Alembic directives in the Operation Reference.

Run migration

First migration:

```
$ alembic upgrade head
INFO [alembic.context] Will assume transactional DDL.
INFO [alembic.context] Running upgrade None -> 1975ea83b712
```

We can also refer directly to revision numbers:

```
$ alembic upgrade ae1
```

Relative migrations can also be initiated:

```
$ alembic upgrade +2
```

or:

```
$ alembic downgrade -1
```

or:

```
$ alembic upgrade ae10+2
```

Display Information

Current version:

```
$ alembic current
INFO [alembic.context] Will assume transactional DDL.
Current revision for postgresql://scott:XXXXX@localhost/test: 1975ea83b712 ->
  ae1027a6acf (head), Add a column
```

History:
$ alembic history --verbose

Rev: ae1027a6acf (head)
Parent: 1975ea83b712
Path: /path/to/yourproject/alembic/versions/ae1027a6acf_add_a_column.py

add a column

Revision ID: ae1027a6acf
Revises: 1975ea83b712
Create Date: 2014-11-20 13:02:54.849677

Rev: 1975ea83b712
Parent: <base>
Path: /path/to/yourproject/alembic/versions/1975ea83b712_add_account_table.py

create account table

Revision ID: 1975ea83b712
Revises: 
Create Date: 2014-11-20 13:02:46.257104

The history can also be displayed more specifically:

$ alembic history -r1975ea:ae1027

or:

$ alembic history -r-3:current

or:

$ alembic history -r1975ea:

See also:
Auto Generating Migrations

**ipython-sql**

**ipython-sql** introduces the %sql or %%sql magics for iPython and Jupyter notebooks.

**Installation**

You can easily install ipython-sql in your Jupyter kernel with:

$ pipenv install ipython-sql
First steps

1. First, ipython-sql is activated in your notebook with
   
   ```
   In [1]: %load_ext sql
   ```

2. The SQLAlchemy URL is used to connect to the database:
   
   ```
   In [2]: %sql postgresql://
   ```

3. Then you can create a table, e.g.:
   
   ```
   In [3]: %sql postgresql://
       ....: CREATE TABLE accounts (login, name, email)
       ....: INSERT INTO accounts VALUES ('veit', 'Veit Schiele', veit@example.org);
   ```

4. You can query the contents of the accounts table with
   
   ```
   In [4]: result = %sql select * from accounts
   ```

Configuration

Query results are loaded as a list, so very large amounts of data can occupy memory. Usually there is no automatic limit, but with Autolimit you can limit the amount of results.

**Note:** displaylimit only limits the amount of results displayed, but not the amount of memory required.

With `%config SqlMagic` you can display the current configuration:

```
In [4]: %config SqlMagic
SqlMagic options
______________
SqlMagic.autocommit=<Bool>
   Current: True
   Set autocommit mode
SqlMagic.autolimit=<Int>
   Current: 0
   Automatically limit the size of the returned result sets
SqlMagic.autopandas=<Bool>
   Current: False
   Return Pandas DataFrames instead of regular result sets
...
```

**Note:** If autopandas is set to True, displaylimit is not applied. In this case, the max_rows option of pandas can be used as described in the pandas documentation.
Pandas

If pandas is installed, the DataFrame method can be used:

```python
In [5]: result = %sql SELECT * FROM accounts
In [6]: dataframe = result.DataFrame()
In [7]: %sql --persist dataframe
In [8]: %sql SELECT * FROM dataframe;
```

--persist  Argument with the name of a DataFrame object, creates a table name in the database from this.
--append  Argument to add rows with this name to an existing table.

PostgreSQL features

Meta-commands from `psql` can also be used in `ipython-sql`:

- `-l, --connections`  lists all active connections
- `-x, --close <session-name>`  close named connection
- `-c, --creator <creator-function>`  specifies the creator function for a new connection
- `-s, --section <section-name>`  specifies section of `dsn_file` to be used in a connection
- `-p, --persist`  creates a table in the database from a named DataFrame
- --append  similar to --persist, but the contents are appended to the table
- `-a, --connection_arguments <"{connection arguments}">`  specifies a dict of connection arguments to be passed to the SQL driver
- `-f, --file <path>`  executes SQL from the file under this path

See also:

- `pgspecial`

**Warning:** Since `ipython-sql` processes -- options such as `--persist`, and at the same time accepts `--` as a SQL comment, the parser has to make some assumptions: for example, `--persist is great` in the first line is processed as an argument and not as a comment.

PostGIS

PostGIS is an extension for PostgreSQL that includes geographic objects and functions. The extension implements i.a. the Simple Feature Access specification of the Open Geospatial Consortium. Although PostgreSQL already supports geometry types, these are insufficient for geographic tasks. Therefore, PostGIS creates its own data types that are better suited for geographic tasks. The following geometry types are supported:

- OpenGIS with well-known text and well-known binary
- Extended Well-Known Text and Extended Well-Known Binary also with height information and/or measured values
- SQL/MM with Circularstring, Compoundcurve, Curvepolygon, Multicurve and Multisurface
GEOS, on the other hand, contains the numerous spatial functions and operators for geographic data. Finally, pgRouting contains routing functions based on PostGIS. In the OpenStreetMap project, PostGIS is used to render maps with Mapnik.

**Install PostGIS**

For Ubuntu 20.04 and 18.04 as well as Debian 10 you can simply install PostGIS with:

```
$ sudo apt install postgis postgresql-12-postgis-3
```

Then you can activate PostGIS.

1. Switch to the PostgreSQL user:

```
$ sudo -i -u postgres
```

2. Create test user and database:

```
$ createuser postgis
$ createdb postgis_db -O postgis
```

3. Establish a connection to the database:

```
$ psql -d postgis_db
```

4. Activate the PostGIS extension in the database:

```
ppostgis_db=# CREATE EXTENSION postgis;
```

5. Check that PostGIS is working:

```
postgis_db=# SELECT PostGIS_version();
   postgres_version
---------------------
   2.5 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
(1 row)
```

**See also:**

- PostGIS Installation

**Optimising PostgreSQL for GIS database objects**

In the standard installation, PostgreSQL is configured very cautiously so that it can run on as many systems as possible. However, GIS database objects are large compared to text data. Therefore, PostgreSQL should be configured to work better with these objects. To do this, we configure the `/etc/postgresql/9.3/main/postgresql.conf` file as follows:

1. `shared_buffer` should be changed to approx. 75% of the total working memory, but never fall below 128 kB:

```
shared_buffers = 768MB
```
2. `work_mem` should be increased to at least 16MB:

```
work_mem = 16MB
```

3. `maintenance_work_mem` should be increased to 128MB:

```
maintenance_work_mem = 128MB
```

4. `checkpoint_segments` should be set to 6:

```
checkpoint_segments = 6
```

5. Finally, `random_page_cost` should be set to 2.0.

```
random_page_cost = 2.0
```

PostgreSQL should be restarted for the changes to take effect:

```
$ sudo service postgresql restart
```

### Loading geospatial data

Now let’s load some geospatial data into our database so that we can familiarise ourselves with the tools and processes used to retrieve that data.

Natural Earth provides a great source of basic data for the whole world on various scales. And the best thing is that this data is in the public domain:

1. Download the data

```
$ mkdir nedata
$ cd !$
cd nedata
    cultural/ne_110m_admin_0_countries.zip
```

2. Unzip the file

```
$ sudo apt install unzip
$ unzip ne_110m_admin_0_countries.zip
Archive:  ne_110m_admin_0_countries.zip
    inflating:  ne_110m_admin_0_countries.README.html
    extracting:  ne_110m_admin_0_countries.VERSION.txt
    extracting:  ne_110m_admin_0_countries.cpg
    inflating:  ne_110m_admin_0_countries.dbf
    inflating:  ne_110m_admin_0_countries.prj
    inflating:  ne_110m_admin_0_countries.shp
    inflating:  ne_110m_admin_0_countries.shx
```

3. Load into our `postgis_db` database

   The files `.dbf`, `.prj`, `.shp` and `.shp` form a so-called ShapeFile, a popular geospatial data format that is used by GIS software. To load this into our database, we also need GDAL, the Geospatial Data Abstraction Library. When we install GDAL we also get OGR, OpenGIS Simple Features Reference Implementation, a vector data translation library that we can use to translate the shapefile into data.
1. GDAL can be easily installed with the package manager:

   $ sudo apt install gdal-bin

2. Then we switch to the postgresql user:

   $ sudo -i -u postgres

3. Now we convert the shapefile with ogr2ogr and import it into our database:

   $ ogr2ogr -f PostgreSQL PG:dbname=postgis_db -progress \
   -nlt PROMOTE_TO_MULTI \
   /home/veit/nedata/ne_110m_admin_0_countries.shp

   -f PostgreSQL indicates that the target is a PostgreSQL database
   PG:dbname=postgis_db specifies the PostgreSQL database name. In addition to the name, other options can also be specified, in general:
   PG:"dbname='db_ename' host='addr' port='5432' user='x' password='y'"
   -progress outputs a progress bar
   -nlt PROMOTE_TO_MULTI indicates that all object types should be loaded into the database as multipolygons
   /home/veit/nedata/ne_110m_admin_0_countries.shp specifies the path to the input file

   See also:
   • ogr2ogr

4. Check the import with ogrinfo

   $ ogrinfo -so PG:dbname=postgis_db ne_110m_admin_0_countries
   Output
   INFO: Open of `PG:dbname=postgis_db'
   using driver `PostgreSQL' successful.
   Layer name: ne_110m_admin_0_countries
   Geometry: Multi Polygon
   Feature Count: 177
   ...

5. Alternatively, we can also list individual tables:

   $ psql -d postgis_db
   postgis_db=# \dt
   List of relations
              Schema | Name                          | Type | Owner
              -------------+---------------------------------+-------+----------
        public | ne_110m_admin_0_countries | table | postgres
        public | spatial_ref_sys           | table | postgres
   (2 rows)

6. Finally, we can log out of the database with

   $ psql -q -c "
close all;
"
psql> \q

See also:

- PostGIS Reference

Database security

Database permissions

The PostgreSQL login via superuser postgres should only ever be allowed via Unix domain sockets and via localhost. Access with peer authentication in the pg_hba.conf, however, can be granted:

```
# TYPE DATABASE USER ADDRESS METHOD
local  all postgres peer
host   all all 10.23.42.1/24 scram-sha-256
```

The database should be created by the database administrator and then configured in such a way that not everyone (PUBLIC) can connect to it:

```
CREATE DATABASE myapp;
REVOKE ALL ON myapp FROM PUBLIC;
```

This means that only the superuser can connect to the myapp database.

Save passwords

Passwords should never be in plain text, e.g. also not be saved in an .env file. When saving and transmitting passwords, this should always be salted. For PostgreSQL there is the extension pgcrypto, which can be easily activated with

```
CREATE EXTENSION pgcrypto;
```

For this reason, secure passwords should be assigned when they are created, which can then get saved e.g. in Vault or similar:

```
CREATE ROLE myapp_users;
CREATE ROLE myapp_reader IN ROLE myapp_users LOGIN PASSWORD '...';
CREATE ROLE myapp_writer IN ROLE myapp_users LOGIN PASSWORD '...';
```

Then users with the role myapp_users first get CONNECT rights and then myapp_reader read rights and myapp_writer write rights:

```
GRANT CONNECT ON DATABASE to myapp_users;
GRANT SELECT ON diagnosis_key TO myapp_reader;
GRANT INSERT ON diagnosis_key TO myapp_writer;
```

The user myapp_reader can, however, read all data at once. This is also a point of attack that is better cut by a function:

```
CREATE OR REPLACE FUNCTION get_key_data(in_id UUID)
RETURNS JSONB
AS $query$
SELECT key_data FROM diagnosis_key WHERE id = in_id;
$query$
LANGUAGE sql SECURITY DEFINER SET search_path = :schema, pg_temp;
```
Then the function `myapp_owner` is assigned, the authorisations for `myapp_reader` and `myapp_writer` are revoked and finally the execution of the function `myapp_reader` is allowed:

```sql
ALTER FUNCTION get_key_data(UUID) OWNER TO myapp_owner;
REVOKE ALL ON FUNCTION get_key_data(UUID) FROM PUBLIC;
GRANT EXECUTE ON FUNCTION get_key_data(UUID) TO myapp_reader;
```

This means that `myapp_reader` can only read a single data record.

**id**

The `id` shouldn’t be written as `serial`, `bigserial` or similar. Counting numbers could be easily guessed by attackers. Therefore the UUIDv4 data type is much more suitable. In PostgreSQL you can generate UUIDv4 with the `uuid-ossp` extension or for PostgreSQL9.4 also the `pgcrypto` extension:

```sql
CREATE EXTENSION "uuid-ossp";
CREATE TABLE diagnosis_key (
    id uuid primary key default uuid_generate_v4() NOT NULL,
    ...
);
```

or

```sql
CREATE EXTENSION "pgcrypto";
CREATE TABLE diagnosis_key (
    id uuid primary key default gen_random_uuid() NOT NULL,
    ...
);
```

**Time stamp**

Occasionally, the date and time are stored as `bigint`, i.e. as a number, even though there is also a `TIMESTAMP` data type. This would have the advantage that you can easily count on them, for example:

```sql
SELECT age(submission_timestamp);
SELECT submission_timestamp - '1 day'::interval;
```

In addition, the data could be deleted after a certain period of time, e.g. after thirty days with:

```sql
DELETE FROM diagnosis_key WHERE age(submission_timestamp) > 30;
```

Deletion can be accelerated if a separate partition is created for each day with the PostgreSQL extension `pg_partman`.

See also:

- Veil2 – Relational Security for Postgres
- PostgreSQL Secure Monitoring (Posemo)
**PostgreSQL performance**

You shouldn’t start with *MVCC – Multiversion Concurrency Control* if you want to optimise your PostgreSQL database: many improvements can be made much easier since neither transaction logs nor large Linux kernel page sizes are likely to be responsible. Usually we start with two metrics that can very well indicate the performance of your databases:

**Cache and index hit rate**

**Cache hit ratio** Percentage of time that data can be served from RAM instead of hard disk space. For a web app with many small requests, I recommend about 99%.

```sql
SELECT
    'index hit rate' AS name,
    (sum(idx_blks_hit)) / nullif(sum(idx_blks_hit + idx_blks_read),0) AS ratio
FROM pg_statio_user_indexes
UNION ALL
SELECT
    'table hit rate' AS name,
    sum(heap_blks_hit) / nullif(sum(heap_blks_hit) + sum(heap_blks_read),0) AS ratio
FROM pg_statio_user_tables;
```

If the cache hit rate is too low, you can simply increase the memory.

**Index hit ratio** Frequency of use of the indices.

```sql
SELECT relname,
    CASE idx_scan
        WHEN 0 THEN 'Insufficient data'
        ELSE (100 * idx_scan / (seq_scan + idx_scan))::text
    END percent_of_times_index_used,
    n_live_tup rows_in_table
FROM pg_stat_user_tables
ORDER BY n_live_tup DESC;
```

Typically, we shouldn’t have more than 10,000 records in a table and the percentage of the index used should be greater than 90%.

In our example, we see that the account table is missing relevant indices, as an index is only used in 11% of the queries. The activity table is also missing some suitable indices, but it also has a lot of records, so it might make sense to split it into several tables.
Clean up unused indices

Unused indices lead to a slower throughput when writing the data sets without making queries faster.

```
SELECT
    schemaname || '.' || relname AS table,
    indexrelname AS index,
    pg_size_pretty(pg_relation_size(i.indexrelid)) AS index_size,
    idx_scan as index_scans
FROM pg_stat_user_indexes ui
JOIN pg_index i ON ui.indexrelid = i.indexrelid
WHERE NOT indisunique AND idx_scan < 50 AND pg_relation_size(relid) > 5 * 8192
ORDER BY pg_relation_size(i.indexrelid) / nullif(idx_scan, 0) DESC NULLS FIRST,
pg_relation_size(i.indexrelid) DESC;
```

Indices that are not used can simply be removed. On the other hand the decision becomes more difficult for indices that are only used very rarely: here a trade-off must be made between the write and the query speed.

Clean up unused data

Although PostgreSQL can hold a wide variety of data, it is not always useful to do so. Tables such as messages, logs and events have a good chance of taking up most of the memory without directly benefiting the database application: if this data is rather for monitoring or error analysis, it should be stored outside the database and rotated regularly.

Analyse query performance with `pg_stat_statements`

`pg_stat_statements` records queries and keeps a number of statistics on them. Thus, at regular intervals, we check which queries are the slowest on average and which put the greatest load on the system:

```
SELECT
    (total_time / 1000 / 60) as total_minutes,
    (total_time/calls) as average_time,
    query
FROM pg_stat_statements
ORDER BY 1 DESC
LIMIT 50;
```

```
<table>
<thead>
<tr>
<th>total_time</th>
<th>avg_time</th>
<th>query</th>
</tr>
</thead>
<tbody>
<tr>
<td>295.761165833319</td>
<td>10.1374053278061</td>
<td>SELECT id FROM account WHERE email LIKE ?</td>
</tr>
<tr>
<td>219.138564283326</td>
<td>80.24530822355305</td>
<td>SELECT * FROM account WHERE user_id = ? AND current = True</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Typical response times should be ~1ms and in a few cases ~4-5ms. To start optimising performance, we usually weigh the total time against the average time, so in the above example we would probably start with the second line as we see the greater potential for savings here. To get a more accurate idea of the query, we analyse it more closely with:

```
EXPLAIN ANALYZE
SELECT *
FROM account
```
WHERE user_id = 123
AND current = True

Aggregate (cost=4690.88..4690.88 rows=1 width=0) (actual time=519.288..519.289 rows=1
loops=1)
  -> Nested Loop (cost=0.00..4690.66 rows=433 width=0) (actual time=15.302..519.076
rows=213 loops=1)
    -> Index Scan using idx_account_userid on account (cost=0.00..232.52 rows=23
width=4) (actual time=10.143..62.822 rows=1 loops=8)
        Index Cond: (user_id = 123)
        Filter: current
        Rows Removed by Filter: 14
Total runtime: 219.428 ms
(1 rows)

So we see that although an index is used, 15 different rows are retrieved from it, of which 14 are then discarded. To optimise this, we would create a conditional or a composite index. In the first case current = true would have to be met, in the second case a composite index would be created with both values. A conditional index is usually more useful with a small set of values, while the composite index is more beneficial with larger sets of values. In our example, a conditional index clearly makes more sense. We can create this with:

```
CREATE INDEX CONCURRENTLY idx_account_userid_current ON account(user_id) WHERE current = True;
```

Now the query plan should also improve:

```
Aggregate (cost=4690.88..4690.88 rows=1 width=0) (actual time=519.288..519.289 rows=1
loops=1)
  -> Index Scan using idx_account_userid_current on account (cost=0.00..232.52 rows=23
width=4) (actual time=10.143..62.822 rows=1 loops=8)
        Index Cond: ((user_id = 123) AND (current = True))
Total runtime: .728 ms
(1 rows)
```
pgMonitor

pgMonitor is an environment to visualise the health and performance of a PostgreSQL cluster. It combines a suite of tools to facilitate the collection of important metrics, including:

- number of connections
- Database size
- Replication lag
- Transaction wraparound
- Extra space taken up by your tables and indexes
- CPU, memory, I/O and uptime

It combines multiple open-source software packages to create a robust PostgreSQL monitoring environment, including:

- PostgreSQL Exporter an open-source data export to Prometheus that supports collecting metrics from any PostgreSQL server 9.1.
- Prometheus an open-source metrics collector that is highly customisable.
- Grafana an open-source data visualiser that allows you to generate many different kinds of charts and graphs.

Installation and configuration

Installation and configuration instructions for each package are provided:

1. PostgreSQL Exporter
2. Prometheus
3. Grafana

4.7 NoSQL databases

So far there is no uniform definition of NoSQL, but most NoSQL database systems usually have the following in common:

- no relational data model
- distributed and horizontal scalability
- no or weak schema restrictions
- simple API
- no ACID, but Eventual consistency or BASE as the consistency model

NoSQL databases can be divided into
### 4.7.1 Key-value database systems

Key-value databases, also known as key value stores, store *key/value pairs*.

#### Database systems

Key/value database systems are e.g. Riak, Cassandra, Redis and MongoDB.

<table>
<thead>
<tr>
<th>Home</th>
<th>Riak</th>
<th>Cassandra</th>
<th>Redis</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub</td>
<td>basho/riak</td>
<td>apache/cassandra</td>
<td>redis/redis</td>
<td>monogodb/mongo</td>
</tr>
<tr>
<td>Docs</td>
<td>docs.riak.com</td>
<td>cassandra.apache.org/doc/</td>
<td>redis.io/documentation</td>
<td>docs.mongodb.com</td>
</tr>
<tr>
<td>Application areas</td>
<td>Session storage, Log data, Sensor data, CMS</td>
<td>Georedundancy, high writing speed, democratic peer-to-peer (P2P) architecture, data with a defined lifetime</td>
<td>Session Cache, Full Page Cache (FPC), Queues, Pub/Sub</td>
<td>IoT, Mobile apps, CMS, simple geospatial data, …</td>
</tr>
<tr>
<td>Development language</td>
<td>Erlang</td>
<td>Java</td>
<td>ANST C</td>
<td>C++</td>
</tr>
<tr>
<td>Licenses</td>
<td>Apache License 2.0</td>
<td>Apache License 2.0</td>
<td>BSD-3-Clause License</td>
<td>Server Side Public License</td>
</tr>
<tr>
<td>Data model</td>
<td>Essentially <em>Key/value pair</em></td>
<td><em>Column Family</em> correspond to tables, keyspaces to databases; no logical structure, no scheme</td>
<td>Keys are stored as strings, values as strings, hashes, lists, sets and sorted sets</td>
<td>Flexible scheme with denormalised model</td>
</tr>
<tr>
<td>Query language</td>
<td>Keyfilter, <em>MapReduce</em>, Link walking, no ad hoc queries possible</td>
<td>Cassandra Query Language (CQL)</td>
<td></td>
<td>jQuery, <em>MapReduce</em></td>
</tr>
<tr>
<td>Transactions, concurrency</td>
<td><em>ACID</em></td>
<td><em>Eventual Consistency</em></td>
<td>in-memory, asynchronous on disc with <em>Append Only File Mode</em></td>
<td>Two-phase locking (2PL)</td>
</tr>
<tr>
<td>Remarks</td>
<td>See also Scylla, a Cassandra-compatible reimplementation in C.</td>
<td>See also KeyDB, a fork from Redis with multithreading.</td>
<td></td>
<td><em>BSON</em> with a maximum document size of 16 MB.</td>
</tr>
</tbody>
</table>
### 4.7.2 Column-oriented database systems

Column-oriented databases, also known as wide column stores, store data from several entries together with a time stamp in columns. Columns with similar or related content can be combined in a *Column family*.

#### Database systems

Examples of column-oriented database systems are *Cassandra*, *Hypertable* and *HBase*.

<table>
<thead>
<tr>
<th>Home</th>
<th>Cassandra</th>
<th>Hypertable</th>
<th>HBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub</td>
<td>apache/cassandra</td>
<td>vicaya/hypertable</td>
<td>apache/hbase</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application areas</th>
<th>Cassandra</th>
<th>Hypertable</th>
<th>HBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Georedundancy, high writing speed, democratic peer-to-peer (P2P) architecture, data with a defined lifetime</td>
<td>Hypertable’s Bigtable design solves horizontal scaling problems through a distributed storage system for structured data.</td>
<td>IoT, fraud detection, recommendation engines</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Development language</th>
<th>Java</th>
<th>C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Licenses</td>
<td>Apache License 2.0</td>
<td>GPL-3.0 License</td>
<td>Apache-2.0 License</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data model</th>
<th>Column Family correspond to tables, Keyspaces databases; no logical structure, no scheme</th>
<th>Associative arrays</th>
<th>Tables divided into regions</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Query language</th>
<th>Cassandra Query Language (CQL)</th>
<th>Hypertable Query Language (HQL)</th>
<th>Java Client API, Thrift/REST API</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Transactions, concurrency</th>
<th>Eventual Consistency</th>
<th>MVCC – Multiversion Concurrency Control</th>
<th>ACID per line, MVCC – Multiversion Concurrency Control</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Replication, scaling</th>
<th>SimpleStrategy, NetworkTopologyStrategy and OldNetworkTopologyStrategy</th>
<th>File system level replication</th>
<th>Master-Slave Replication</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Remarks</th>
<th>is based on distributed file systems such as Apache Hadoop, DFS or GlusterFS</th>
</tr>
</thead>
</table>

---

**4.7. NoSQL databases**
4.7.3 Document-oriented database systems

A document in this context is a structured compilation of certain data. The data of a document is stored as a *Key/value pair*, whereby the value can also be a list or an array.

**Database systems**

Document-oriented database systems are, for example, MongoDB, CouchDB, Riak, OrientDB and ArangoDB.
<table>
<thead>
<tr>
<th><strong>Home</strong></th>
<th><strong>MongoDB</strong></th>
<th><strong>CouchDB</strong></th>
<th><strong>Riak</strong></th>
<th><strong>OrientDB</strong></th>
<th><strong>ArangoDB</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub</td>
<td>mongodbc/mongo</td>
<td>apache/couchdb</td>
<td>basho/riak</td>
<td>orientdb/orientdb</td>
<td>arangodb/arangodb</td>
</tr>
<tr>
<td>Docs</td>
<td>docs.mongodb.com</td>
<td>docs.couchdb.org</td>
<td>docs.riak.com</td>
<td><a href="http://www.orientdb.com">www.orientdb.com</a></td>
<td>arangodb.com/documentation/</td>
</tr>
<tr>
<td>Application areas</td>
<td>IoT, Mobile apps, CMS, simple geospatial data, ...</td>
<td>Mobile, CRM, CMS, ...</td>
<td>Session storage, Log data, Sensor data, CMS</td>
<td>Master data management, social networks, Time Series, Key Value, Chat, traffic management</td>
<td>Fraud Detection, IoT, identity management, e-commerce, network, logistics, CMS</td>
</tr>
<tr>
<td>Development language</td>
<td>C++</td>
<td>Erlang</td>
<td>Erlang</td>
<td>Java</td>
<td>C++, JavaScript</td>
</tr>
<tr>
<td>Licenses</td>
<td>Server Side Public License</td>
<td>Apache License 2.0</td>
<td>Apache License 2.0</td>
<td>Apache License 2.0</td>
<td>Apache License 2.0</td>
</tr>
<tr>
<td>Data model</td>
<td>Flexible scheme with denormalised model</td>
<td>Flexible scheme</td>
<td>Essentially Key/Value pair</td>
<td>Multi-Model</td>
<td>Multi-model: documents, graphs and Key/value pair</td>
</tr>
<tr>
<td>Query language</td>
<td>jQuery, MapReduce</td>
<td>REST, MapReduce</td>
<td>Key filter, MapReduce, link walking, no ad-hoc queries possible</td>
<td>Extended SQL, Gremlin</td>
<td>ArangoDB Query Language (AQL)</td>
</tr>
<tr>
<td>Transactions, concurrency</td>
<td>Two-phase locking (2PL)</td>
<td><strong>ACID</strong></td>
<td><strong>ACID</strong></td>
<td>ACID, <strong>MVCC</strong> – Multiversion Concurrency Control</td>
<td></td>
</tr>
<tr>
<td>Replication, scaling</td>
<td>Master-Slave replication, Auto-Sharding</td>
<td>Master-master replication</td>
<td>Multi-master replication</td>
<td>Multi-Master-Replikation, Sharding</td>
<td>Master-slave replication, sharding</td>
</tr>
<tr>
<td>Remarks</td>
<td>BSON with a maximum document size of 16 MB.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.7.4 Graph database systems

Graph databases specialise in networked information and the simplest and most efficient possible *Graph traversal*.

**Graph model**

A graph consists of a number of nodes and edges. Graphs are used to represent a variety of problems through nodes, edges and their relationships, for example in navigation systems in which the paths are stored in the form of graphs.

**Graph traversal**

Graph traversal is mostly used to find nodes. There are different algorithms for such search queries in a graph, which can be roughly divided into

- **Breadth-first search, BFS and depth-first search, DFS**
  
  The breadth-first search begins with all neighboring nodes of the start node. In the next step, the neighbors of the neighbors are then searched. The path length increases with each iteration.

  The depth-first search follows a path until a node with no outgoing edges is found. The path is then traced back to a node that has further outgoing edges. The search will then continue there.

- **Algorithmic traversal**
  
  Examples of algorithmic traversal are
  
  – Hamiltonian path (traveling salesman)
  
  – Eulerian path
  
  – Dijkstra’s algorithm

- **Randomised traversal**
  
  The graph is not run through according to a certain scheme, but the next node is selected at random. This allows a search result to be presented much faster, especially with large graphs, but this is not always the best.

**Database systems**

Typical graph databases are Neo4j, OrientDB InfiniteGraph and ArangoDB.
<table>
<thead>
<tr>
<th><strong>Home</strong></th>
<th>Neo4j</th>
<th>OrientDB</th>
<th>InfiniteGraph</th>
<th>ArangoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GitHub</strong></td>
<td>neo4j/neo4j</td>
<td>orienttechnologies/orientdb</td>
<td>InfiniteGraph Tutorials</td>
<td>arangodb/arangodb</td>
</tr>
<tr>
<td><strong>Docs</strong></td>
<td>neo4j.com/docs/</td>
<td>orientdb.org/docs/</td>
<td>arangodb.com/documentation/</td>
<td></td>
</tr>
<tr>
<td><strong>Application areas</strong></td>
<td>CMS, social networks, GIS systems, ERP, ...</td>
<td>Master data management, social networks, time series, key value, traffic management</td>
<td>Extension of Objectivity/DB installations</td>
<td>Fraud Detection, IoT, identity management, e-commerce, network, logistics, CMS</td>
</tr>
<tr>
<td><strong>Development language</strong></td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
<td>C++, JavaScript</td>
</tr>
<tr>
<td><strong>Licenses</strong></td>
<td>AGPL and commercially</td>
<td>Apache License 2.0</td>
<td>commercially</td>
<td>Apache License 2.0</td>
</tr>
<tr>
<td><strong>Data model</strong></td>
<td>Property graph model (PGM)</td>
<td>Multi-Model</td>
<td>Property graph model (PGM)</td>
<td>Multi-model: documents, graphs and Key/Value pair</td>
</tr>
<tr>
<td><strong>Query language</strong></td>
<td>REST, Cypher, Gremlin</td>
<td>Extended SQL, Cypher, Gremlin</td>
<td>Traverser API, PQL</td>
<td>ArangoDB Query Language (AQL)</td>
</tr>
<tr>
<td><strong>Transactions, concurrency</strong></td>
<td>Two-phase locking (2PL)</td>
<td>ACID</td>
<td>ACID</td>
<td>ACID, MVCC – Multiversion Concurrency Control</td>
</tr>
<tr>
<td><strong>Replication, scaling</strong></td>
<td>Master-slave with master failover</td>
<td>Multi-master replication, Sharding</td>
<td>Objectivity/DB, no Graph partitioning</td>
<td>Master-slave replication, sharding</td>
</tr>
<tr>
<td><strong>Remarks</strong></td>
<td>InfiniteGraph is a graph database on top of the Object database systems Objectivity/DB, whereby the objects are connected by edges. Multiple and bidirectional edges are also allowed here. Iterators correspond to the Graph traversal.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See also:

- Apache TinkerPop Home
- TinkerPop Documentation
- github.com/apache/tinkerpop
- Practical Gremlin – An Apache TinkerPop Tutorial
- gremlinpython

4.7. NoSQL databases 203
4.7.5 Object database systems

Many programming languages suggest object-oriented programming, so storing these objects seems natural. It therefore makes sense to design the entire process from implementation to storage uniformly and simply. In detail, the advantages are:

**Natural modeling and representation of problems**  Problems can be modeled in ways that are very close to the human way of thinking.

**Clearer, more readable and more understandable**  The data and the functions operating on them are combined into one unit, making the programs clearer, more readable and easier to understand.

**Modular and reusable**  Program parts can be easily and flexibly reused.

**Expandable**  Programs can be easily expanded and adapted to changed requirements.

**Object-relational impedance mismatch**

Object-oriented programming and relational data storage are problematic for various reasons. Inheritance is an important concept in OOP for implementing complex models. In the relational paradigm, however, there is nothing like it. Object-relational mappers, ORM, such as *SQLAlchemy*, were developed to convert corresponding class hierarchies into a relational model. In principle there are two different approaches for an ORM, whereby in both cases a table is created for a class:

**Vertical partitioning**  The table only contains the attributes of the corresponding class and a foreign key for the table of the superclass. An entry is then created for each object in the table belonging to the class and in the tables of all superclasses. When accessing the tables, joins must be used, which can lead to significant performance losses in complex models.

**Horizontal partitioning**  Each table contains the attributes of the associated class and all superclasses. If the superclass is changed, however, the tables of all derived classes must also be updated.

Basically, when combining OOP and relational data management, two data models must always be created. This makes this architecture significantly more complex, more error-prone and more time-consuming to maintain.

**Database systems**

Examples of object database systems are ZODB and Objectivity/DB.

<table>
<thead>
<tr>
<th>Home</th>
<th>ZODB</th>
<th>Objectivity/DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub</td>
<td>zopefoundation/ZODB</td>
<td></td>
</tr>
<tr>
<td>Application areas</td>
<td>Plone, Pyramid, BTrees, volatile data</td>
<td>IoT, telecommunications, network technology</td>
</tr>
<tr>
<td>Development language</td>
<td>Python</td>
<td>Java</td>
</tr>
<tr>
<td>Licenses</td>
<td>Zope Public License (ZPL) 2.1</td>
<td>commercially</td>
</tr>
<tr>
<td>Data model</td>
<td>PersistentList, PersistentMapping, BTree</td>
<td>Objects, References, Relationships, Indexes, Trees and Collections</td>
</tr>
<tr>
<td>Query language</td>
<td></td>
<td>Objectivity/DB predicate query language</td>
</tr>
<tr>
<td>Transactions, concurrency</td>
<td>ACID</td>
<td>ACID</td>
</tr>
<tr>
<td>Replication, scaling</td>
<td>ZODB Replication Services (ZRS)</td>
<td>Quorum based synchronous replication</td>
</tr>
</tbody>
</table>

Remarks
4.7.6 XML database systems

XML databases are able to validate XML documents against an XML schema or a DTD. In addition, they support at least XPATH, XQuery and XSLT.

Database systems

Examples of XML database systems are eXist and MonetDB.

<table>
<thead>
<tr>
<th>Home</th>
<th>GitHub</th>
<th>Docs</th>
<th>Application areas</th>
<th>Development language</th>
<th>Licenses</th>
<th>Data model</th>
<th>Query language</th>
<th>Transactions, concurrency</th>
<th>Replication, scaling</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>eXist</td>
<td>eXist-db/exist</td>
<td>exist-db.org/exist/apps/doc/documentation</td>
<td>CMS</td>
<td>Java</td>
<td>LGPL-2.1 License</td>
<td>XML</td>
<td>XQuery, XPATH</td>
<td>Optimistic Concurrency</td>
<td>Master-slave replication</td>
<td>With R, analyses can be carried out directly at the database level.</td>
</tr>
<tr>
<td>MonetDB</td>
<td>MonetDB/MonetDB</td>
<td><a href="http://www.monetdb.org/Documentation">www.monetdb.org/Documentation</a></td>
<td>CMS, Date-Warehouse, Data mining</td>
<td>C</td>
<td>Mozilla Public License 2.0</td>
<td>XML, column-oriented data structure</td>
<td>SQL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BaseX</td>
<td>BaseXdb/baseX</td>
<td>docs.baseX.org</td>
<td>CMS</td>
<td>Java</td>
<td>BSD-3-Clause License</td>
<td>XML, geographic data</td>
<td>XQuery, XPATH</td>
<td>ACID, XQuery Locks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Major concepts and technologies of NoSQL databases are

- **MapReduce**
- **CAP theorem**
- **Eventual consistency and BASE**
- **Consistent hash function**
- **MVCC – Multiversion Concurrency Control**
- **Vector clock**
- **Paxos**
4.8 gRPC

gRPC is a modern, open source, high-performance remote procedure call (RPC) framework. By default, gRPC uses Protocol Buffers (Protobuf) as the Interface Definition Language (IDL) for describing both the service interface and the structure of the payload messages. In gRPC, a client application can directly call a method on a server application on a different machine as if it were a local object, making it easier for you to create distributed applications and services. As in many RPC systems, gRPC is based on the idea of defining a service, specifying the methods that can be called remotely with their parameters and return types. The server implements the interface and runs a gRPC server to handle client calls; the client has a stub that provides the same methods as the server.

The following are the main design principles of gRPC:

- gRPC can be created on all common development platforms and in many different languages.
- It is designed to work on devices with low CPU and memory capabilities, such as Android\(^1\) and iOS devices, MicroPython boards and browsers\(^2\).
- It is licensed under Apache License 2.0 and uses open standards such as HTTP/2 and Quick UDP Internet Connections (QUIC).
- gRPC is interoperable and can therefore also be used in the LoRaWan (Long Range Wide Area Network), for example.
- The individual layers can be developed independently of each other. For example, the transport layer (OSI layer 4) can be developed independently of the application layer (OSI layer 7).
- gRPC supports various serialisation formats, including Protocol Buffers (Protobuf), JSON\(^4\), XML and Thrift.
- Asynchronous and synchronous (blocking) processing are supported in most languages.
- Streaming of messages in a single RPC call is supported.
- gRPC allows protocol extensions for security, health checks, load balancing, failover, etc.

---

\(^1\) gRPC in Android Java
\(^2\) gRPC-Web is Generally Available
\(^3\) gRPC-Web Client Runtime Library
\(^4\) gRPC + JSON
Starting with an interface definition in a `.proto` file, gRPC provides Protocol Compiler plugins that generate Client- and Server-side APIs. Both synchronous and asynchronous communication is supported in most languages. gRPC also supports streaming of messages in a single RPC call. The gRPC protocol abstractly specifies the communication between clients and servers:

1. First the stream is started by the client with a mandatory **Call Header**
   
   1. followed by optional **Initial-Metadata**
   
   2. followed by optional **Payload Messages**.

   The contents of **Call Header** and **Initial Metadata** are sent as HTTP/2 headers compressed with HPACK.

2. The server answers with an optional **Initial-Metadata**

   1. followed by **Payload Messages**

   2. and terminated with mandatory **Status** and optional **Status-Metadata**.

   Payload Messages are serialised into a byte stream fragmented into HTTP/2 frames. **Status** and **Trailing-Metadata** are sent as HTTP/2 trailing headers.

Unlike Fastapi, however, the gRPC API cannot simply be tested on the command line with cURL. If necessary, you can use `grpcurl`. This requires that the gRPC server supports the GRPC Server Reflection Protocol. Usually **Reflection** should only be available in the development phase. Then you can call grpcurl, e.g. with:

```
$ grpcurl localhost:9111 list
```

**See also:**

- Home
- GitHub
- gRPC Blog
4.8.1 gRPC-Example

By default, gRPC uses Protocol Buffers (Protobuf) for serialising data, although it also works with other data formats such as JSON.

Define the data structure

The first step when working with protocol buffers is to define the structure for the data you want to serialise in a .proto file. Protocol buffer data is structured as messages, where each message is a small logical record of information containing a series of name-value pairs called fields. Here’s a simple example accounts.proto:

```protobuf
syntax = "proto3";

message Account {
  uint32 account_id = 1;
  string account_name = 2;
}
```

**Warning:** You shouldn’t simply use uint32 for user or group IDs, as these would be far too easy to guess. You can use an RFC4122-compliant implementation for this purpose. You can find a corresponding protobuf configuration in rfc4122.proto.

After you have defined your data structure, you use the protocol buffer compiler protoc to generate descriptors in your preferred languages. These provide simple accessors for each field, as well as methods to serialise the whole structure. For example, if your language is Python, running the compiler on the example above will generate declarators you can then use in your application to populate, serialise, and retrieve protocol buffer messages.

Define the gRPC service

gRPC services are also defined in the .proto files, with RPC method parameters and return types specified as protocol buffer messages:

```protobuf
message CreateAccountRequest {
  string account_name = 1;
}

message CreateAccountResult {
  Account account = 1;
}

message GetAccountsRequest {
  repeated Account account = 1;
}

message GetAccountsResult {
  Account account = 1;
}
```
### Generate the gRPC Code

```
$ pipenv install grpcio grpcio-tools
$ pipenv run python -m grpc_tools.protoc --python_out=. --grpc_python_out=. accounts._proto
```

This generates two files:

- **accounts_pb2.py** contains classes for the messages defined in accounts.proto.
- **accounts_pb2_grpc.py** contains the defined classes AccountsStub for calling RPCs, AccountsServicer for the API definition of the service and a function add_AccountsServicer_to_server for the server.

### Create server

For this we write the file accounts_server.py:

```python
from concurrent import futures
import logging
import grpc
import accounts_pb2_grpc as accounts_service
import accounts_pb2 as accounts_messages

class AccountsService(accounts_service.AccountsServicer):
    def CreateAccount(self, request, context):
        metadata = dict(context.invocation_metadata())
        print(metadata)
        account = accounts_messages.Account(
            account_name=request.account_name, account_id=1
        )
        return accounts_messages.CreateAccountResult(account=account)

    def GetAccounts(self, request, context):
        for account in request.account:
            account = accounts_messages.Account(
                account_name=account.account_name,
                account_id=account.account_id,
            )
            yield accounts_messages.GetAccountsResult(account=account)

    def serve():
        server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
        accounts_service.add_AccountsServicer_to_server(AccountsService(), server)
        server.add_insecure_port("[::]:8081")
        server.start()
        server.wait_for_termination()
```

(continues on next page)
logging.basicConfig()
serve()

Create client

For this we create accounts_client.py:

```python
import logging
import grpc
import accounts_pb2_grpc as accounts_service
import accounts_pb2 as accounts_messages

def run():
    channel = grpc.insecure_channel("localhost:8081")
    stub = accounts_service.AccountsStub(channel)
    response = stub.CreateAccount(
        accounts_messages.CreateAccountRequest(account_name="tom"),
    )
    print("Account created:", response.account.account_name)

if __name__ == "__main__":
    logging.basicConfig()
    run()
```

Run client and server

1. Starting the server:

   ```sh
   $ pipenv run python accounts_server.py
   ```

2. Starting the client from another terminal:

   ```sh
   $ pipenv run python accounts_client.py
   Account created: tom
   ```

4.8.2 Test gRPC

pytest-grpc

gRPC can be tested automatically with pytest-grpc.

1. First, we install

   ```sh
   $ pipenv install pytest-grpc
   Installing pytest-grpc...
   ```
Adding pytest-grpc to Pipfile's [packages]...
✓ Installation Succeeded
...

+ Then we create a Test Fixture for our gRPC-Example with:

```python
import pytest
from accounts_pb2 import CreateAccountRequest, GetAccountsRequest

@ pytest.fixture(scope="module")
def grpc_add_to_server():
    from accounts_pb2_grpc import add_AccountsServicer_to_server
    return add_AccountsServicer_to_server

@ pytest.fixture(scope="module")
def grpc_servicer():
    from accounts_server import AccountsService
    return AccountsService()

@ pytest.fixture(scope="module")
def grpc_stub(grpc_channel):
    from accounts_pb2_grpc import AccountsStub
    return AccountsStub(grpc_channel)
```

See also:

- pytest fixtures

1. Afterwards we can write tests, e.g.:

```python
def test_create_account(grpc_stub):
    value = "test-data"
    nl = "\n"
    request = CreateAccountRequest(account_name=value)
    response = grpc_stub.CreateAccount(request)

    assert 
    f"{response.account}"
    == f"account_id: 1
    account_name: "test-data"\n"

def test_get_accounts(grpc_stub):
    request = GetAccountsRequest()
    response = accounts_server.GetAccounts(request)
    assert response.name == f"test-{request.name}"
```

4.8. gRPC
2. Authentication can also be tested, e.g. with:

```python
from pathlib import Path
import grpc

@pytest.fixture(scope="module")
def grpc_server(_grpc_server, grpc_addr, my_ssl_key_path, my_ssl_cert_path):
    ""
    Overwrites default `grpc_server` fixture with ssl credentials
    ""

    credentials = grpc.ssl_server_credentials(
        [(my_ssl_key_path.read_bytes(), my_ssl_cert_path.read_bytes())]
    )

    _grpc_server.add_secure_port(grpc_addr, server_credentials=credentials)
    _grpc_server.start()
    yield _grpc_server
    _grpc_server.stop(grace=None)

@ pytest.fixture(scope="module")
def my_channel_ssl_credentials(my_ssl_cert_path):
    # If we're using self-signed certificate it's necessarily to pass root -> certificate to channel
    return grpc.ssl_channel_credentials(
        root_certificates=my_ssl_cert_path.read_bytes()
    )

@ pytest.fixture(scope="module")
def grpc_channel(my_channel_ssl_credentials, create_channel):
    ""
    Overwrites default `grpc_channel` fixture with ssl credentials
    ""

    with create_channel(my_channel_ssl_credentials) as channel:
        yield channel

@ pytest.fixture(scope="module")
def grpc_authorized_channel(my_channel_ssl_credentials, create_channel):
    ""
    Channel with authorization header passed
    ""

    grpc_channel_credentials = grpc.access_token_call_credentials("some_token")
    composite_credentials = grpc.composite_channel_credentials(
        my_channel_ssl_credentials, grpc_channel_credentials
    )

    with create_channel(composite_credentials) as channel:
        yield channel

@ pytest.fixture(scope="module")
def my_authorized_stub(grpc_stub_cls, grpc_channel):
    ""
```
(continues on next page)
3. Afterwards we can test against a real gRPC server with:

$ pipenv run pytest --fixtures tests/

or directly against the Python code:

$ pipenv run pytest --fixtures tests/ --grpc-fake-server

platform darwin -- Python 3.7.3, pytest-6.2.2, py-1.10.0, pluggy-0.13.1
rootdir: /Users/veit/cusy/trn/jupyter-tutorial/docs/data/grpc
plugins: grpc-0.8.0
collected 2 items

tests/test_accounts.py .F [100%]
...

See also:

- GitHub
- Example

Wireshark

Wireshark is an open source tool for analysing network protocols. In the following, we will show you how to use the gRPC and Protobuf dissectors. They make it easier for you to decode gRPC messages that are serialised in Protobuf or JSON format. You can also use them to analyse server, client and bidirectional gRPC streaming.

Note: Usually, Wireshark can only analyse gRPC messages in plain text. For dissecting a TLS session, Wireshark needs the secret key, the export of which is currently only supported by Go gRPC\(^1\).

See also:

- Analyzing gRPC messages using Wireshark

4.9 FastAPI

FastAPI is a web framework for building APIs with Python 3.6+ based type hints.

Key features are:

- very high performance thanks to pydantic for the data part and Starlette for the web part.
- fast and easy to code
- validation for most Python data types, including

---

\(^1\) How to Export TLS Master keys of gRPC
– JSON objects (dict)
– JSON array (list)
– string (str), defining min and max length
– numbers (int, float) with min and max values, etc.
– URLs
– email with python-email-validator
– UUID
– … and others

• robust, production-ready code with automatic interactive documentation
• based on the open standards for APIs: OpenAPI formerly known as Swagger) and JSON Schema

See also:

• Home
• GitHub

4.9.1 Installation

Requirements

```
$ pipenv install fastapi
Adding fastapi to Pipfile's [packages]...
✓ Installation Succeeded
Locking [dev-packages] dependencies...
✓ Success!
Locking [packages] dependencies...
✓ Success!
...
```

Optional requirements

For production you also need an ASGI server, such as uvicorn:

```
$ pipenv install uvicorn
Adding uvicorn to Pipfile's [packages]...
✓ Installation Succeeded
Locking [dev-packages] dependencies...
✓ Success!
Locking [packages] dependencies...
✓ Success!
Updated Pipfile.lock (051f02)!
...
```

Pydantic can use the optional dependencies

ujjson for faster JSON parsing.
email_validator for email validation.
Starlette can use the optional dependencies

**requests** if you want to use the TestClient.

**aiofiles** if you want to use FileResponse or StaticFiles.

**jinja2** if you want to use the default template configuration.

**python-multipart** if you want to support form parsing, with request.form().

**itsdangerous** required for SessionMiddleware support.

**pyyaml** for Starlette’s SchemaGenerator support.

**graphene** for GraphQLApp support.

**ujson** if you want to use UJSONResponse.

**orjson** if you want to use ORJSONResponse.

They can be installed, e.g. with:

```
$ pipenv install fastapi[ujson]
```

Alternatively you can install all of these with:

```
$ pipenv install fastapi[all]
```

### 4.9.2 Example

#### 1. Create

Create a file main.py with:

```python
from typing import Optional
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}"),
def read_item(item_id: int, q: Optional[str] = None):
    return {"item_id": item_id, "q": q}
```

#### 2. Run

Run the server with:

```
$ pipenv run uvicorn main:app --reload
```

INFO:   Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:   Started reloader process [89155] using statetreload
INFO:   Started server process [89164]
INFO:   Waiting for application startup.
INFO:   Application startup complete.

### 4.9. FastAPI
3. Check

Open your browser at http://127.0.0.1:8000/ and you will see:

You will also get an interactive API documentation provided by Swagger UI at http://127.0.0.1:8000/docs:
You will also get an alternative automatic documentation provided by ReDoc at http://127.0.0.1:8000/redoc:

4. Update

Now we modify the file main.py to receive a body from a PUT request:

```python
from typing import Optional
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Item(BaseModel):
    name: str
    price: float
    is_offer: Optional[bool] = None

@app.get("/")
def read_root():
    return {'Hello': 'World'}

@app.get("/items/{item_id}"),
def read_item(item_id: int, q: Optional[str] = None):
    return {'item_id': item_id, 'q': q}

@app.post("/items/{item_id}"),
def update_item(item_id: int, item: Item):
    return {'item_name': item.name, 'item_id': item_id}
```

The server should reload the file automatically because we added --reload to the uvicorn command. Also the interactive API documentation will show the new body with PUT. If you click on the button Try it out you will fill in the parameter for item_id. Then click on the Execute button and your browser will send the parameter to the API and show them on the screen, e.g. as response body:

```json
{
    "item_name": "string",
    "item_id": 1234
}
```
4.9. FastAPI
Read Root

Responses

- 200 Successfull Response

RESPONSE SCHEMA: application/json

any

Response samples

null

Read Item

PATH PARAMETERS

- item_id
  
  integer (item-id)

QUERY PARAMETERS

- q
  
  string (q)

Responses

- 200 Successfull Response

RESPONSE SCHEMA: application/json

any

- 404 Validation Error

RESPONSE SCHEMA: application/json

- detail
  
  Array of objects (Detail)
4.10 Glossary

ACID  ACID is an acronym for Atomicity Consistency Isolation Durability. They are a prerequisite for the reliability of database transactions.

Atomicity  A transaction is a series of database operations that are either carried out completely or not at all.

Consistency  Transaction that leaves a consistent state after completion. The integrity conditions defined in the database schema are checked before the transaction is completed.

Isolation  Concurrent transactions must not influence each other. This is usually achieved with Locking, which restricts the concurrency.

Durability  After a successful transaction, data must be permanently stored in the database and can be secured, for example, by writing a transaction log.

BASE  BASE is an acronym for Basically Available, Soft State, Eventually Consistent and originated as the opposite of ACID.

A very optimistic concept of consistency is used that does not require Locking. Locks are problematic in several ways, since access is not possible as long as data records are locked by other transactions. In addition, the agreement to set a lock is already very complex.

Data consistency is seen as a state that can be achieved at some point. This is the idea of Eventual Consistency.

With BASE, competing access is avoided through MVCC – Multiversion Concurrency Control However, there is a wide range of solutions for the various distributed database systems:

- Causal Consistency
  is comparable to the consistency in ACID.
- Read Your Writes
- Session Consistency
- Monotonic Read Consistency
- Monotonic Write Consistency

CAP theorem  CAP is an acronym for Consistency, Availability and Partition Tolerance. The findings of the CAP theorem play a central role in the selection of a distributed database system.

The CAP theorem states that in distributed systems the three requirements of consistency, availability and failure tolerance are not fully compatible and only a maximum of two out of three can be achieved. Therefore it must be decided individually for each application whether a CA, CP or AP application should be implemented.

Cassandra  Cassandra is a Column-oriented database systems, and was originally developed by Facebook to optimise searches in email. Today it is further developed under the umbrella of the Apache Software Foundation.

Cassandra’s data model has neither a logical structure nor a schema. For the modeling it is recommended «First write your queries then model your data». Then usually a Column Family is created for each expected request. The data is denormalised, but each column family responds to a specific type of query.

In Cassandra, the consistency can be specified for each request. This allows specific requests to be very consistent while others sacrifice consistency for speed. There are, for example, the following four levels for write consistency:

- ANY  ensures that the data is stored in at least one node.
- ONE  ensures that the data is stored in the commit log of at least one replica.
- QUORUM  ensures that the data is stored in a quorum of replicas.
- ALL  ensures that the data is saved on all replicas.
Cassandra provides two different APIs: Thrift and CQL (Cassandra Query Language).

**Column Family** Column families correspond to tables in relational databases. They group columns with the same or similar content, e.g.

```python
profile = {
    cusy: {
        name: "Cusy GmbH",
        email: "info@cusy.io",
        website: "cusy.io"
    },
    veit: {
        name: "Veit Schiele",
        email: "veit.schiele@cusy.io"
    }
}
```

**Consistent hash function** Consistent hash functions minimise the number of reallocations, since not all keys have to be reallocated when a change occurs, only the size of a hash table is changed.

**Consistency** The state of a database is said to be consistent if the stored data meets all requirements for **Semantic integrity**.

**CouchDB** CouchDB an acronym for Cluster of unreliable commodity hardware Data Base. This is a **Document-oriented database systems**.

**Eventual Consistency** «Consistency as a state transition that is reached at some point.»

The term was developed for **BASE** as an alternative to **ACID**.

**Graph traversal** Graph traversal is mostly used to find nodes. There are different algorithms for such search queries in a graph, which can be roughly divided into

- **Breadth-first search, BFS and depth-first search, DFS**
  
  The breadth-first search begins with all neighboring nodes of the start node. In the next step, the neighbors of the neighbors are then searched. The path length increases with each iteration.
  
  The depth-first search follows a path until a node with no outgoing edges is found. The path is then traced back to a node that has further outgoing edges. The search will then continue there.

- **Algorithmic traversal**
  
  Examples of algorithmic traversal are
  
  - Hamiltonian path (traveling salesman)
  - Eulerian path
  - Dijkstra’s algorithm

- **Randomised traversal**
  
  The graph is not run through according to a certain scheme, but the next node is selected at random. This allows a search result to be presented much faster, especially with large graphs, but this is not always the best.

**Graph model** A graph consists of a number of nodes and edges. Graphs are used to represent a variety of problems through nodes, edges and their relationships, for example in navigation systems in which the paths are stored in the form of graphs.
Graph partitioning With graph partitioning, graphs are divided into smaller subgraphs. However, there is no mathematically exact method to minimise the number of intersected edges, but only a few heuristic algorithms, e.g. clustering algorithms, which combine strongly networked subgraphs to abstract nodes.

One speaks of overlapping partitioning in the case of graphs that cannot be completely divided and exist in several subgraphs.

HBase HBase is a Column-oriented database systems, which is based on distributed file systems and is designed for real-time access to large databases.

Hypertable Hypertable is a Column-oriented database systems and is based on distributed file systems. The data model is that of a multi-dimensional table that can be searched using keys. The first dimension is the so-called row key, the second is the Column family, the third dimension is the column qualifier and the fourth dimension is time.

Key/value pair A value is always assigned to a specific key, which can consist of a structured or arbitrary character string. These keys can be divided into namespaces and databases. In addition to strings, the values can also contain lists, sets or hashes.

Locking Locking is the term used to describe the blocking of data for concurrent transactions.

There are different lock procedures, depending on the type of access:

Optimistic concurrency Optimistic concurrency, also called optimistic locking, assumes that there are few write accesses to the database and read accesses do not trigger a lock. In the event of changes, a check is first made to determine whether the time stamp has remained unchanged since the data was read.

Pessimistic locking Pessimistic locking assumes a lot of write accesses to the database. Read access is therefore also blocked. The data is only released again when the changes have been saved.

Two-phase locking (2PL) The two-phase locking protocol distinguishes between two phases of transactions:

1. The growth phase in which locks can only be set but not released.
2. The shrinkage phase, in which locks can only be released but not requested.

The two-phase lock protocol knows three lock states:

SLOCK, shared lock or read lock is set with read access to data

XLOCK, exclusive lock or write lock is set with write access to data

UNLOCK removes the locks SLOCK and XLOCK.

MapReduce MapReduce is a framework introduced by Google Inc. in 2004, which is used for the concurrent computations of enormous amounts of data on computer clusters. It was inspired by the map and reduce functions, which are often used in functional programming, even if the semantics deviate slightly from them.

MongoDB MongoDB is a schema-free Document-oriented database systems, that manages documents in BSON format.

MVCC – Multiversion Concurrency Control MVCC controls concurrent accesses to data records (read, insert, change, delete) by different, unchangeable versions of these data records. The various versions are arranged in a chronological order, with each version referring to its previous version. MVCC has developed into a central basic technology for NoSQL databases in particular, which makes it possible to coordinate competing accesses even without locking data records.

Paxos Paxos is a family of protocols for building consensus on a network of unreliable or fallible processors.

Property graph model (PGM) Nodes and edges consist of objects with properties embedded in them. Not only a value (label) is stored in an edge or a node, but a Key/value pair.

Riak In essence, Riak is a decentralised Key/value pair with a flexible MapReduce engine.

Redis Redis is a Key-value database systems, that usually stores all data in RAM.
Semantic integrity  Semantic integrity is always given when the entries are correct and consistent. Then we talk of consistent data. If this is not the case, the data is inconsistent. In SQL, the semantic integrity can be checked with TRIGGER and CONSTRAINT

Vector clock  A vector clock is a software component used to assign unique time stamps to messages. It allows a causal order to be assigned to the events in distributed systems on the basis of a time stamp and, in particular, to determine the concurrency of events.

XPATH  XPATH processes the tree structure of an XML document and generates extracts from XML documents. In order to receive complete XML documents as a result, these must be created with XQuery or XSLT, for example. XPATH is not a complete query language as it is limited to selections and extractions. XPATH has been part of XQuery since version 1.1 and from version 2.0 onwards, XPATH is extended by XQuery.

XQuery  XQuery stands for XML Query Language and is mainly a functional language in which nested expressions can also be evaluated during a query.

XSLT  XSLT is an acronym for Extensible Stylesheet Language Transformation. It can be used to transform XML documents.
In this chapter we want to give you a practical overview of libraries and methods for data cleansing and validation with Python.

See also:
- Data Cleaning 101 by Katharine Jarmul
- Great Expectations

### 5.1 Deduplicate data

In this notebook, we deduplicate data using the Dedupe library, which uses a flat neural network to learn from a small training session.

In addition, the same developers have created parserator with which you can extract text functions and train your own text extraction.

#### 5.1.1 1. Imports

```python
[1]: import pandas as pd
    import dedupe
    import os

```

#### 5.1.2 2. Check data quality

```python
[3]: customers.head()
```

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>job</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Patricia Schaefer</td>
<td>Programmer, systems</td>
</tr>
<tr>
<td>1</td>
<td>Olivie Dubois</td>
<td>Ingénieur recherche et développement en agroal...</td>
</tr>
<tr>
<td>2</td>
<td>Mary Davies-Kirk</td>
<td>Public affairs consultant</td>
</tr>
<tr>
<td>3</td>
<td>Mirosława Eckbauer</td>
<td>Dispensing optician</td>
</tr>
<tr>
<td>4</td>
<td>Richard Bauer</td>
<td>Accountant, chartered certified</td>
</tr>
</tbody>
</table>

(continues on next page)
5.1.3 3. Configure Dedupe

Now we define the fields to be taken into account during deduplication and create a new deduper object:

```python
variables = [
    {'field': 'name', 'type': 'String'},
    {'field': 'job', 'type': 'String'},
    {'field': 'company', 'type': 'String'},
    {'field': 'street_address', 'type': 'String'},
    {'field': 'city', 'type': 'String'},
    {'field': 'state', 'type': 'String', 'has_missing': True},
    {'field': 'email', 'type': 'String', 'has_missing': True},
    {'field': 'user_name', 'type': 'String'},
]```
deduper = dedupe.Dedupe(variables)

[7]: deduper
    <dedupe.api.Dedupe at 0x12736ed30>

[8]: customers.shape
    (2080, 8)

5.1.4 4. Create training data

[9]: deduper.prepare_training(customers.T.to_dict())

```
INFO:dedupe.training:Final predicate set:
INFO:dedupe.training:(SimplePredicate: (alphaNumericPredicate, email), SimplePredicate: α→(wholeFieldPredicate, company))
```

5.1.5 5. Active learning

If Dedupe finds a pair of records, you will be asked to mark it as a duplicate. You can do this using the y, n and u buttons to mark duplicates. Push f when you’re done.

[10]: dedupe.console_label(deduper)

```
name : Frédérique Lejeune-Daniel
job : Technicien chimiste
company : Schmitt
street_address : chemin Denise Ferrand
city : Saint CharlotteVille
state : IE
email : jchretien@costa.com
user_name : joseph60

name : Frédérique Lejeune-Daniel
job : Tecce cse
company : Sctmitt
street_address : chemin Denise Ferrand
city : Saint ChalotteVille
state : IE
email : jchretien@costacom
user_name : joseph60

0/10 positive, 0/10 negative
Do these records refer to the same thing?
(y)es / (n)o / (u)nsure / (f)inished

y
INFO:dedupe.training:Final predicate set:
INFO:dedupe.training:(SimplePredicate: (alphaNumericPredicate, email), SimplePredicate:
  → (wholeFieldPredicate, company))
INFO:dedupe.training:(SimplePredicate: (alphaNumericPredicate, user_name),
  → SimplePredicate: (wholeFieldPredicate, email))

name : Monique Marty
job : Maroquinier
company : Arnaud
street_address : 70, rue de Carre
city : ChevallierBourg
state : EC
e-mail : frederiquerichard@cohen.com
user_name : marquessebastien

1/10 positive, 0/10 negative
Do these records refer to the same thing?
(Yes / No / Unsure / Finished / Previous)
y
INFO:dedupe.training:Final predicate set:
INFO:dedupe.training:(SimplePredicate: (alphaNumericPredicate, email), SimplePredicate:
  → (wholeFieldPredicate, company))
INFO:dedupe.training:(SimplePredicate: (alphaNumericPredicate, user_name),
  → SimplePredicate: (wholeFieldPredicate, email))

name : Ing. Marian Heidrich MBA.
job : Civil engineer, consulting
company : Johann Heuser AG
street_address : Lilija-Ortmann-Straße 54
city : Husum
state : Hamburg
e-mail : truebconcetta@googlemail.com
user_name : marie78

2/10 positive, 0/10 negative
Do these records refer to the same thing?
(Yes / No / Unsure / Finished / Previous)
f

Finished labeling
INFO:dedupe.training:Final predicate set:
import os

deduper = Dedupe(training_file/url, ...)

def main():
    print('reading labeled examples from ', training_file)
    deduper.prepare_training(customers.T.to_dict(), f)

    if os.path.exists(training_file):
        print('reading labeled examples from ', training_file)
        deduper.prepare_training(customers.T.to_dict(), f)
    else:
        deduper.prepare_training(customers.T.to_dict())

    if os.path.exists(training_file):
        deduper.prepare_training(customers.T.to_dict())
    else:
        deduper.prepare_training(customers.T.to_dict())

    # Make sure predicate set is stable before training.
    deduper.prepare_training()  

    # The code gets more complicated when performing active learning.
    if os.path.exists(settings_file):
        print('reading labeled examples from ', training_file)
        deduper.prepare_training()  

    # Save the weights and active learning settings.
    with open(training_file, 'w') as tf:
        deduper.write_training(tf)

    if os.path.exists(settings_file):
        print('reading labeled examples from ', training_file)
        deduper.prepare_training()  

    # Save the weights and active learning settings.
    if os.path.exists(settings_file):
        deduper.write_training(tf)

    # When you're done, save your training data:
    with open(training_file, 'w') as tf:
        deduper.write_training(tf)

    # Also save your weights and predicates. If settings_file already exists, training and active learning will be skipped.

When you're done, save your training data:

with open(training_file, 'w') as tf:
    deduper.write_training(tf)

Also save your weights and predicates. If settings_file already exists, training and active learning will be skipped in the next run:

settings_file = 'csv_example_learned_settings'
if os.path.exists(settings_file):

(continues on next page)
5.2 String matching

In this notebook we use the popular string matching library fuzzywuzzy. For more information on the different methods available and their differences, see the blog post FuzzyWuzzy: Fuzzy String Matching in Python.

5.2.1 1. Import

```python
from fuzzywuzzy import fuzz, process
```

5.2.2 2. Beispiel

```python
berlin = ['Berlin, Germany',
          'Berlin, Deutschland',
          'Berlin',
          'Berlin, DE']
```

5.2.3 String similarity

The match of the first two strings and seems low: 'Berlin, Germany' and 'Berlin, Deutschland':

```python
fuzz.ratio(berlin[0], berlin[1])
```

```python
65
```

5.2.4 Partial string similarity

Inconsistent substrings are a common problem. To get around this, fuzzywuzzy uses a heuristic called best partial.

```python
fuzz.partial_ratio(berlin[0], berlin[1])
```

```python
60
```
5.2.5 Token sorting

With token sorting, the relevant character sequence is provided with a token, the tokens are sorted alphabetically and then reassembled into a character sequence, for example:

```python
[5]: fuzz.ratio(berlin[1], berlin[2])
  48
[6]: fuzz.token_set_ratio(berlin[1], berlin[2])
  100
```

5.2.6 Additional Information

```python
[7]: fuzz.ratio?
```

5.2.7 Extract from a list

```python
[8]: choices = ['Germany',
               'Deutschland',
               'France',
               'United Kingdom',
               'Great Britain',
               'United States']
[9]: process.extract('DE', choices, limit=2)
   [('Deutschland', 90), ('Germany', 45)]
[10]: process.extract('Vereinigtes Königreich', choices)
    [('United Kingdom', 51), ('United States', 41), ('Germany', 39), ('Great Britain', 35), ('Deutschland', 31)]
[11]: process.extractOne('frankreich', choices)
    ('France', 62)
[12]: process.extractOne('U.S.', choices)
    ('United States', 86)
```
5.2.8 Known ports

FuzzyWuzzy is also ported to other languages. Here are some known ports:

- Java: xpresso
- Java: xdrop fuzzywuzzy
- Rust: fuzzyrusty
- JavaScript: fuzzball.js
- C++: tmplt fuzzywuzzy
- C#: FuzzySharp
- Go: go-fuzzywuzzy
- Pascal: FuzzyWuzzy.pas
- Kotlin: FuzzyWuzzy-Kotlin
- R: fuzzywuzzyR

5.3 Managing missing data with pandas

This notebook introduces a few ways to manage nulls using panda’s DataFrames. Further information can be found in the documentation of the panda: Working with missing data.

5.3.1 See also:

- Missing data cookbook

```python
[1]: import pandas as pd
from numpy import random


5.3.2 Check the data

```python
[3]: df.head()

```
```
```
```
5.3. Managing missing data with pandas

5.3.3 Remove all null values (including the note n/a)

df = pd.read_csv('https://raw.githubusercontent.com/kjam/data-cleaning-101/master/data/iot_example_with_nulls.csv', na_values=['n/a'])

Test if we can use dropna

[6]: df.shape
[6]: (146397, 7)

[7]: df.dropna().shape
[7]: (46116, 7)

[8]: df.dropna(how='all', axis=1).shape
[8]: (146397, 7)
Find all columns in which all data is present

```python
[9]: my_columns = list(df.columns)

[10]: my_columns
['timestamp', 'username', 'temperature', 'heartrate', 'build', 'latest', 'note']

[11]: list(df.dropna(thresh=int(df.shape[0] * .9), axis=1).columns)
['timestamp', 'username', 'heartrate']
```

Finding all columns that are missing data

```python
[12]: missing_info = list(df.columns[df.isnull().any()])

[13]: missing_info
['temperature', 'build', 'latest', 'note']

[14]: for col in missing_info:
    num_missing = df[df[col].isnull() == True].shape[0]
    print('number missing for column {}: {}'.format(col, num_missing))

  number missing for column temperature: 32357
  number missing for column build: 32350
  number missing for column latest: 32298
  number missing for column note: 48704

[15]: for col in missing_info:
    percent_missing = df[df[col].isnull() == True].shape[0] / df.shape[0]
    print('percent missing for column {}: {}'.format(col, percent_missing))

  percent missing for column temperature: 0.2210228870810195
  percent missing for column build: 0.22097447352063226
  percent missing for column latest: 0.22061927498514314
  percent missing for column note: 0.332684412931959
5.3.4 Replace missing data with majority values

```
[16]: df.note.value_counts()

                     wake       16496
                       user       16416
                     interval     16274
                       sleep       16226
                       update       16213
                       test       16068
Name: note, dtype: int64
```

```
[17]: df.build.value_counts().head()

       b1d3b3a7-6639-9b0b-9b4c-22a976563f74    1
         43b11996-707a-0522-23d5-19d17b1f41e6    1
        ee8339c4-cbab-8164-a17e-2efb4f80dc18    1
       012ba321-84f3-83e6-7d63-b344674bd40c    1
       aacd60a6-100c-ac70-8322-13b5909604d9    1
Name: build, dtype: int64
```

```
[18]: df.latest.value_counts()

        0.0       75735
         1.0       38364
Name: latest, dtype: int64
```

```
[19]: df.latest = df.latest.fillna(0)
```

**Example for the missing temperature values**

```
[20]: df.username.value_counts().head()

        esmith         45
            zsmith         43
            vsmith         41
            ysmith         40
            jsmith         37
Name: username, dtype: int64
```

```
[21]: df = df.set_index('timestamp')
```

```
[22]: df.head()

           username  temperature   heartrate
  timestamp
2017-01-01T12:00:23  michaelsmith       12.0       67
2017-01-01T12:01:09     kharrison        6.0       78
2017-01-01T12:01:34   smithadam        5.0       89
2017-01-01T12:02:09  eddierodriguez     28.0       76
2017-01-01T12:02:36    kenneth94       29.0       62
```

(continues on next page)
5.4 Scikit Learn preprocessing

[1]: from sklearn import preprocessing
from sklearn.impute import SimpleImputer
import numpy as np
import pandas as pd
from datetime import datetime


5.4.1 Checking the data quality

[3]: hvac.dtypes

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>object</td>
</tr>
<tr>
<td>Time</td>
<td>object</td>
</tr>
<tr>
<td>TargetTemp</td>
<td>float64</td>
</tr>
<tr>
<td>ActualTemp</td>
<td>int64</td>
</tr>
<tr>
<td>System</td>
<td>int64</td>
</tr>
<tr>
<td>SystemAge</td>
<td>float64</td>
</tr>
<tr>
<td>BuildingID</td>
<td>int64</td>
</tr>
<tr>
<td>10</td>
<td>float64</td>
</tr>
</tbody>
</table>

dtype: object

[4]: hvac.shape

(8000, 8)

[5]: hvac.head()

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>TargetTemp</th>
<th>ActualTemp</th>
<th>System</th>
<th>SystemAge</th>
<th>BuildingID</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/1/13</td>
<td>0:00:01</td>
<td>66.0</td>
<td>58</td>
<td>13</td>
<td>20.0</td>
<td>4</td>
<td>NaN</td>
</tr>
<tr>
<td>6/2/13</td>
<td>1:00:01</td>
<td>NaN</td>
<td>68</td>
<td>3</td>
<td>20.0</td>
<td>17</td>
<td>NaN</td>
</tr>
<tr>
<td>6/3/13</td>
<td>2:00:01</td>
<td>70.0</td>
<td>73</td>
<td>17</td>
<td>20.0</td>
<td>18</td>
<td>NaN</td>
</tr>
<tr>
<td>6/4/13</td>
<td>3:00:01</td>
<td>67.0</td>
<td>63</td>
<td>2</td>
<td>NaN</td>
<td>15</td>
<td>NaN</td>
</tr>
<tr>
<td>6/5/13</td>
<td>4:00:01</td>
<td>68.0</td>
<td>74</td>
<td>16</td>
<td>9.0</td>
<td>3</td>
<td>NaN</td>
</tr>
</tbody>
</table>
### 5.4.2 Assign the mean to missing values

```python
[6]: imp = SimpleImputer(missing_values=np.nan, 
    strategy='mean')

[7]: hvac_numeric = hvac[['TargetTemp', 'SystemAge']]

[8]: imp = imp.fit(hvac_numeric.loc[:10])

[9]: transformed = imp.fit_transform(hvac_numeric)

[10]: transformed
```

```
array([[66., 20.],
       [67.50773481, 20.],
       [70., 20.],
       ...
       [67.50773481, 4.],
       [65., 23.],
       [66., 21.]])
```

```python
[11]: hvac['TargetTemp'], hvac['SystemAge'] = transformed[:,0], transformed[:,1]

[12]: hvac.head()
```

```
<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>TargetTemp</th>
<th>ActualTemp</th>
<th>System</th>
<th>SystemAge</th>
<th>BuildingID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6/1/13</td>
<td>0:00:01</td>
<td>66.000000</td>
<td>58</td>
<td>13</td>
<td>20.000000</td>
</tr>
<tr>
<td>1</td>
<td>6/2/13</td>
<td>1:00:01</td>
<td>67.507735</td>
<td>68</td>
<td>3</td>
<td>20.000000</td>
</tr>
<tr>
<td>2</td>
<td>6/3/13</td>
<td>2:00:01</td>
<td>70.000000</td>
<td>73</td>
<td>17</td>
<td>20.000000</td>
</tr>
<tr>
<td>3</td>
<td>6/4/13</td>
<td>3:00:01</td>
<td>67.000000</td>
<td>63</td>
<td>2</td>
<td>15.386643</td>
</tr>
<tr>
<td>4</td>
<td>6/5/13</td>
<td>4:00:01</td>
<td>68.000000</td>
<td>74</td>
<td>16</td>
<td>9.000000</td>
</tr>
</tbody>
</table>
```

### 5.4.3 Scaling temperature values

```python
[13]: hvac['ScaledTemp'] = preprocessing.scale(hvac['ActualTemp'])

[14]: hvac['ScaledTemp'].head()
```

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1.293272</td>
</tr>
<tr>
<td>1</td>
<td>0.048732</td>
</tr>
<tr>
<td>2</td>
<td>0.719733</td>
</tr>
<tr>
<td>3</td>
<td>-0.622270</td>
</tr>
<tr>
<td>4</td>
<td>0.853934</td>
</tr>
</tbody>
</table>
```

Name: ScaledTemp, dtype: float64
5.4.4 Scaling with the MinMaxScaler

[15]: min_max_scaler = preprocessing.MinMaxScaler()

[16]: temp_minmax = min_max_scaler.fit_transform(hvac[['ActualTemp']])

[17]: temp_minmax

array([[0.12],
       [0.52],
       [0.72],
       ...
       [0.56],
       [0.32],
       [0.44]])

5.5 Dask pipeline

5.5.1 Example: Tracking the International Space Station with Dask

In this notebook we will be using two APIs:

1. Google Maps Geocoder
2. Open Notify API for ISS location

We will use them to keep track of the ISS location and next lead time in relation to a list of cities. To create our diagrams and intelligently parallelise data, we use Dask, especially Dask Delayed.

1. Imports

[1]:
import requests
import logging
import sys
import numpy as np
from time import sleep
from datetime import datetime
from math import radians
from dask import delayed
from operator import itemgetter
from sklearn.neighbors import DistanceMetric
2. Logger

[2]: logger = logging.getLogger()
    logger.setLevel(logging.INFO)

3. Latitude and longitude pairs from a list of cities

see also Location APIs

[3]:

```python
def get_lat_long(address):
    resp = requests.get(
        'https://eu1.locationiq.org/v1/search.php',
        params={'key': '92e7ba84cf3465',
                'q': address,
                'format': 'json'}
    )
    if resp.status_code != 200:
        print('There was a problem with your request!')
        print(resp.content)
    return
    data = resp.json()[0]
    return {
        'name': data.get('display_name'),
        'lat': float(data.get('lat')),
        'long': float(data.get('lon')),
    }
```

[4]:

```python
get_lat_long('Berlin, Germany')
```

[4]: {'name': 'Berlin, 10117, Germany', 'lat': 52.5170365, 'long': 13.3888599}

[5]:

```python
locations = []
for city in ['Seattle, Washington',
             'Miami, Florida',
             'Berlin, Germany',
             'Singapore',
             'Wellington, New Zealand',
             'Beirut, Lebanon',
             'Beijing, China',
             'Nairobi, Kenya',
             'Cape Town, South Africa',
             'Buenos Aires, Argentina']:
    locations.append(get_lat_long(city))
sleep(2)
```

[6]:

```python
locations
```

[6]: [{'name': 'Seattle, King County, Washington, USA',
      'lat': 47.6038321,
      'long': -122.3300624},
     {'name': 'Miami, Miami-Dade County, Florida, USA',
      'lat': 25.7742658,
      'long': -80.1735732},
     {'name': 'Buenos Aires, Capital Federal, Argentina',
      'lat': -34.5838716,
      'long': -58.4776596},
     {'name': 'Wellington, Wellington City, New Zealand',
      'lat': -41.2819312,
      'long': 174.7820882},
     {'name': 'Beijing, Beijing, China',
      'lat': 39.904226,
      'long': 116.4074807},
     {'name': 'Nairobi, Nairobi, Kenya',
      'lat': -1.28333333, 'long': 36.87333333},
     {'name': 'Cape Town, Western Cape, South Africa',
      'lat': -33.928953, 'long': 18.4243312},
     {'name': 'Buenos Aires, Argentina',
      'lat': -34.5838716, 'long': -58.4776596},
     {'name': 'Wellington, New Zealand',
      'lat': -41.2819312, 'long': 174.7820882},
     {'name': 'Beijing, China',
      'lat': 39.904226, 'long': 116.4074807},
     {'name': 'Nairobi, Kenya',
      'lat': -1.28333333, 'long': 36.87333333},
     {'name': 'Cape Town, South Africa',
      'lat': -33.928953, 'long': 18.4243312}]

(continues on next page)
{'name': 'Berlin, 10117, Germany', 'lat': 52.5170365, 'long': 13.3888599},
{'name': 'Singapore, Singapore, Central, Singapore',
 'lat': 1.340863,
 'long': 103.830391822121},
{'name': 'Wellington, Wellington City, Wellington, 6011, New Zealand',
 'lat': -41.2887953,
 'long': 174.7772114},
{'name': 'Beirut, Ras Beirut, Beirut Governorate, Lebanon',
 'lat': 33.8959203,
 'long': 35.47843},
{'name': 'Beijing, Dongcheng District, Beijing, 100010, China',
 'lat': 39.906217,
 'long': 116.3912757},
{'name': 'Nairobi, Kenya', 'lat': -1.2832533, 'long': 36.8172449},
{'name': 'Cape Town, City of Cape Town, Western Cape, 8001, South Africa',
 'lat': -33.928992,
 'long': 18.417396},
{'name': 'Autonomous City of Buenos Aires, Comuna 6, Autonomous City of Buenos Aires, Argentina',
 'lat': -34.6075682,
 'long': -58.4370894}]

5.5.2 Retrieve ISS data and determine lead times for cities

[7]: def get_spaceship_location():
    resp = requests.get('http://api.open-notify.org/iss-now.json')
    location = resp.json()['iss_position']
    return { 'lat': float(location.get('latitude')),
             'long': float(location.get('longitude'))}

[8]: def great_circle_dist(lon1, lat1, lon2, lat2):
    "Found on SO: http://stackoverflow.com/a/41858332/380442"
    dist = DistanceMetric.get_metric('haversine')
    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])
    X = [[lat1, lon1], [lat2, lon2]]
    kms = 6367
    return (kms * dist.pairwise(X)).max()

[9]: def iss_dist_from_loc(issloc, loc):
    distance = great_circle_dist(issloc.get('long'),
                                issloc.get('lat'),
                                loc.get('long'), loc.get('lat'))
    logging.info('ISS is ~%d km from %s', int(distance), loc.get('name'))
    return distance

[10]: def iss_pass_near_loc(loc):
    resp = requests.get('http://api.open-notify.org/iss-pass.json',
    (continues on next page)
params={'lat': loc.get('lat'),
        'lon': loc.get('long'))

data = resp.json().get('response')[0]

td = datetime.fromtimestamp(data.get('risetime')) - datetime.now()

m, s = divmod(int(td.total_seconds()), 60)
h, m = divmod(m, 60)

logging.info('ISS will pass near %s in %02d:%02d:%02d',loc.get('name'), h, m, s)

return td.total_seconds()

[11]: iss_dist_from_loc(get_spaceship_location(), locations[4])
    INFO:root:ISS is ~13325km from Wellington, Wellington City, Wellington, 6011, New Zealand

[11]: 13325.326019595097

[12]: iss_pass_near_loc(locations[4])
    INFO:root:ISS will pass near Wellington, Wellington City, Wellington, 6011, New Zealand
        → in 02:32:28

[12]: 9148.867879

5.5.3 5. Create a delayed pipeline

[13]: output = []

    for loc in locations:
        issloc = delayed(get_spaceship_location)()
        dist = delayed(iss_dist_from_loc)(issloc, loc)
        output.append((loc.get('name'), dist))

    closest = delayed(lambda x: sorted(x,
                              key=itemgetter(1))[0])(output)

[14]: closest
[14]: Delayed('lambda-7c277b48-05c2-46ec-b511-4b7740cf187d')

5.5.4 6. Show DAG

[15]: closest.visualize()
5.5.5 7. compute()

```python
closest.compute()
```

```
INFO:root:ISS is ~8893km from Singapore, Singapore, Central, Singapore
INFO:root:ISS is ~9226km from Autonomous City of Buenos Aires, Comuna 6, Autonomous City...
INFO:root:ISS is ~14460km from Wellington, Wellington City, Wellington, 6011, New Zealand
INFO:root:ISS is ~3694km from Cape Town, City of Cape Town, Western Cape, 8001, South...
INFO:root:ISS is ~1443km from Nairobi, Kenya
INFO:root:ISS is ~13891km from Seattle, King County, Washington, USA
INFO:root:ISS is ~6033km from Berlin, 10117, Germany
INFO:root:ISS is ~11454km from Miami, Miami-Dade County, Florida, USA
INFO:root:ISS is ~10292km from Beijing, Dongcheng District, Beijing, 100010, China
```

```
('Nairobi, Kenya', 1443.415385986981)
```

5.6 Data validation with voluptuous (schema definitions)

In this notebook we use Voluptuous, to define schemas for our data. We can then use the schema checker at various points in our cleanup to make sure we meet the criteria. Finally, we can use schema checking exceptions to mark, set aside, or remove impure or invalid data.

5.6.1 1. Imports

```python
import logging
import pandas as pd
from datetime import datetime
from voluptuous import Schema, Required, Range, All, ALLOW_EXTRA
from voluptuous.error import MultipleInvalid, Invalid
```
5.6.2 2. Logger

```python
[3]: logger = logging.getLogger(0)
logger.setLevel(logging.WARNING)
```

5.6.3 3. Read sample data

```python
```

5.6.4 4. Examine data

```python
[5]: sales.head()
```

```
   Unnamed : 0  timestamp       city  store_id  sale_number
0         0 2018-09-10 05:00:45  Williamburgh 6     1530
1         1 2018-09-12 10:01:27     Ibarraberg 1     2744
2         2 2018-09-13 12:01:48    Sarachester 2     1908
3         3 2018-09-14 20:02:19  Caldwellbury 14      771
4         4 2018-09-16 01:03:21    Erikanland 11     1571

   sale_amount  associate
0      1167.0     Gary Lee
1       258.0    Daniel Davis
2       266.0    Michael Roth
3     -108.0  Michaela Stewart
4     -372.0     Mark Taylor
```

```python
[6]: sales.dtypes
```

```
Unnamed: 0   int64
timestamp    object
city         object
store_id     int64
sale_number  int64
sale_amount  float64
associate    object
dtype: object
```

5.6.5 5. Define the scheme

```python
[7]: schema = Schema({
    Required('sale_amount'): All(float,
                                   Range(min=2.50, max=1450.99)),
}, extra=ALLOW_EXTRA)
```

```python
[8]: error_count = 0
for s_id, sale in sales.T.to_dict().items():
```

(continues on next page)
try:
schema(sale)
except MultipleInvalid as e:
    logging.warning('issue with sale: %s (%s) - %s', s_id, sale['sale_amount'], e)
error_count += 1

WARNING:root:issue with sale: 3 (-108.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 4 (-372.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 5 (-399.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 6 (-304.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 7 (-295.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 10 (-89.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 13 (-303.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 15 (-432.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 19 (-177.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 20 (-154.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 22 (-130.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 23 (1487.0) - value must be at most 1450.99 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 25 (-145.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 28 (1471.0) - value must be at most 1450.99 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 31 (-259.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 38 (-241.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 40 (-4.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 41 (1581.0) - value must be at most 1450.99 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 45 (1529.0) - value must be at most 1450.99 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 46 (-238.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 48 (-284.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 51 (-164.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']
WARNING:root:issue with sale: 55 (-184.0) - value must be at least 2.5 for dictionary
  → value @ data['sale_amount']

(continues on next page)
WARNING:root:issue with sale: 56 (-304.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 59 (1579.0) - value must be at most 1450.99 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 60 (-455.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 63 (1551.0) - value must be at most 1450.99 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 65 (-397.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 69 (-400.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 70 (1482.0) - value must be at most 1450.99 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 71 (-321.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 74 (-47.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 76 (-68.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 86 (1454.0) - value must be at most 1450.99 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 101 (-213.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 103 (-144.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 104 (-265.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 107 (-349.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 111 (-78.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 112 (-310.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 116 (1570.0) - value must be at most 1450.99 for dictionary
    → dictionary value @ data['sale_amount']
WARNING:root:issue with sale: 120 (1490.0) - value must be at most 1450.99 for dictionary
    → dictionary value @ data['sale_amount']
WARNING:root:issue with sale: 123 (-179.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 124 (-391.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 129 (1504.0) - value must be at most 1450.99 for dictionary
    → dictionary value @ data['sale_amount']
WARNING:root:issue with sale: 130 (-91.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 132 (-372.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
WARNING:root:issue with sale: 141 (1512.0) - value must be at most 1450.99 for dictionary
    → dictionary value @ data['sale_amount']
WARNING:root:issue with sale: 142 (-449.0) - value must be at least 2.5 for dictionary
    → value @ data['sale_amount']
However, we don’t currently know whether

- we have an incorrectly defined scheme
- negative values may be returned or incorrectly marked
• higher values are combined purchases or special sales

## 5.6.6 6. Add a custom validation

```python
[11]: def ValidDate(fmt='%Y-%m-%d %H:%M:%S'):
    return lambda v: datetime.strptime(v, fmt)

[12]: schema = Schema({
    Required('timestamp'): All(ValidDate()),
}, extra=ALLOW_EXTRA)

[13]: error_count = 0
    for s_id, sale in sales.T.to_dict().items():
        try:
            schema(sale)
        except MultipleInvalid as e:
            logging.warning('issue with sale: %s (%s) - %s',
                             s_id, sale['timestamp'], e)
            error_count += 1

[14]: error_count
[14]: 0
```

## 5.6.7 7. Valid date structures are not yet valid dates

```python
[15]: def ValidDate(fmt='%Y-%m-%d %H:%M:%S'):
    def validation_func(v):
        try:
            assert datetime.strptime(v, fmt) <= datetime.now()
        except AssertionError:
            raise Invalid('date is in the future! %s' % v)
    return validation_func

[16]: schema = Schema({
    Required('timestamp'): All(ValidDate()),
}, extra=ALLOW_EXTRA)

[17]: error_count = 0
    for s_id, sale in sales.T.to_dict().items():
        try:
            schema(sale)
        except MultipleInvalid as e:
            logging.warning('issue with sale: %s (%s) - %s',
                             s_id, sale['timestamp'], e)
            error_count += 1

[18]: error_count
```
5.7 Pandas DataFrame validation with Engarde

Engarde is no longer under active development. Take a look at the Bulwark tool for similar functionality.

In this notebook we check pandas.DataFrame objects with the engarde library. With it you can write decorators for functions as well as use built-in functions to test your DataFrame with certain validation rules or definitions.

5.7.1 1. Imports

```python
import pandas as pd
import engarde.decorators as ed
from datetime import datetime
```

5.7.2 2. Read data

```python
```

5.7.3 3. Check data

```python
sales.head()
```

<table>
<thead>
<tr>
<th>timestamp</th>
<th>city</th>
<th>store_id</th>
<th>sale_number</th>
<th>sale_amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-09-15T06:17:10</td>
<td>Alexandrabury</td>
<td>18</td>
<td>1043.0</td>
<td>15.0</td>
</tr>
<tr>
<td>2017-09-11T16:30</td>
<td>East Jesusport</td>
<td>2</td>
<td>1729.0</td>
<td>396.0</td>
</tr>
<tr>
<td>2017-07-12T15:00:18</td>
<td>New Dougsmouth</td>
<td>13</td>
<td>2028.0</td>
<td>-78.0</td>
</tr>
<tr>
<td>2017-07-29T13:04:55</td>
<td>West Carriemouth</td>
<td>19</td>
<td>1245.0</td>
<td>1149.0</td>
</tr>
<tr>
<td>2017-11-07T21:35:58</td>
<td>Port Timothy</td>
<td>1</td>
<td>2365.0</td>
<td>724.0</td>
</tr>
</tbody>
</table>

associate
0  Stacey Daniels
1  Haley Pitts
2  Carlos French
3  Jeffrey Ford
4  Christopher West

```python
sales.dtypes
```

<table>
<thead>
<tr>
<th></th>
<th>dtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>object</td>
</tr>
<tr>
<td>city</td>
<td>object</td>
</tr>
<tr>
<td>store_id</td>
<td>int64</td>
</tr>
<tr>
<td>sale_number</td>
<td>float64</td>
</tr>
<tr>
<td>sale_amount</td>
<td>float64</td>
</tr>
<tr>
<td>associate</td>
<td>object</td>
</tr>
</tbody>
</table>
5.7.4 Check data types

Engarde lets us keep track of data types. So in a first function we should check our expected results.

```python
[5]: new_dtypes = {
    'timestamp': object,
    'city': object,
    'store_id': int,
    'sale_number': float,
    'sale_amount': float,
    'associate': object
}
```

```python
[6]: @ed.has_dtypes(new_dtypes)
@ed.is_shape((None, 6))
@ed.shape((None, 6))
def update_dtypes(sales):
    sales.timestamp = sales.timestamp.map(lambda x: datetime.strptime(x, '%Y-%m-%dT%H:%M:%S').date())
    return sales
```

```python
[7]: sales = update_dtypes(sales)
```

```python
[8]: sales.timestamp.iloc[0]
[8]: datetime.date(2017, 9, 15)
```

5.7.5 5. Remove insufficient data

To remove data of poor quality, we first remove duplicates and missing entries.

```python
[9]: @ed.has_dtypes(new_dtypes)
@ed.is_shape((None, 6))
@ed.none_missing()
def remove_poor_quality_data(sales):
    sales = sales.drop_duplicates()
    sales = sales.dropna(subset=['sale_amount', 'store_id',
                               'sale_number',
                               'city', 'associate'])
    return sales
```

```python
[10]: sales = remove_poor_quality_data(sales)
```

```python
[11]: sales.isnull().any()
[11]: timestamp False
city False
store_id False
sale_number False
sale_amount False
associate False
type: bool
```

5.7. Pandas DataFrame validation with Engarde
final_types = new_dtypes.copy()
final_types.update({
    'store_total': float,
    'associate_total': object,
    'city_total': float
})

@ed.has_dtypes(final_types)
@ed.none_missing()
def calculate_store_sales(sales):
    sales['store_total'] = sales.groupby('store_id').transform(sum)['sale_amount']
    sales['associate_total'] = sales.groupby('associate').transform(sum)['sale_amount']
    sales['city_total'] = sales.groupby('city')[
        'sale_amount'].transform(sum)
    return sales

sales.head()

5.8 Pandas DataFrame validation with Bulwark

Bulwark is a package for property-based testing of pandas dataframes. The project was heavily influenced by the no-longer-supported Engarde library.
5.8.1 Installation

```
$ pipenv install bulwark
Installing bulwark...
Adding bulwark to Pipfile's [packages]...
✓ Installation Succeeded
Locking [dev-packages] dependencies...
✓ Success!
Updated Pipfile.lock (0d075a)!
```

5.8.2 Usage

**Checks**

Bulwark comes with checks for many of the common assumptions.

```
import bulwark.checks as ck
df.pipe(ck.has_no_nans())
```

**Decorators**

For each check in check.py bulwark.decorators creates decorators, e.g.:

```
import bulwark.decorators as dc
@dc.IsShape((-1, 10))
@dc.IsMonotonic(strict=True)
@dc.HasNoNans()
def compute(df):
    # complex operations to determine result
    ...
    return result_df
```

**CustomCheck**

You can also create your own custom function, e.g.:

```
import bulwark.checks as ck
import bulwark.decorators as dc
import numpy as np
import pandas as pd

def len_longer_than(df, l):
    if len(df) <= l:
        raise AssertionError("df is not as long as expected.")
    return df

@dc.CustomCheck(len_longer_than, 10, enabled=False)
```

(continues on next page)
def append_a_df(df, df2):
    return df.append(df2, ignore_index=True)

def = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
df2 = pd.DataFrame({"a": [1, np.nan, 3, 4], "b": [4, 5, 6, 7]})
append_a_df(df, df2)  # doesn't fail because the check is disabled

MultiCheck

With the built-in MultiCheck you can run multiple tests and see all the errors at once, e.g.:

```python
@dc.MultiCheck(checks={ck.has_no_nans: {"columns": None},
    len_longer_than: {"l": 6},
    warn=False)
def append_a_df(df, df2):
    return df.append(df2, ignore_index=True)

def = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
df2 = pd.DataFrame({"a": [1, np.nan, 3, 4], "b": [4, 5, 6, 7]})
append_a_df(df, df2)
```

Note: When you use MultiCheck, there's no need to use CustomCheck – just feed in the function.

5.9 TDDA: Test-Driven Data Analysis

In this notebook we will take a closer look at the Python TDDA library that takes data inputs (such as NumPy arrays or Pandas DataFrames) and creates a series of constraints around them. You can then save your constraints (JSON output) and test new data based on the observed constraints.

5.9.1 1. Imports

```python
[1]: import pandas as pd
    import numpy as np
    from tdda.constraints.pdconstraints import discover_constraints, \
        verify_df

[2]: df = pd.read_csv('https://raw.githubusercontent.com/kjam/data-cleaning-101/master/data/\n    iot_example.csv')
```
5.9.2 2. Check data

```python
[3]: df.sample(10)
```

```bash
39897 2017-01-17T10:34:58 starknicholas 9 63
56184 2017-01-23T22:48:32 hendersonsteven 11 70
79080 2017-02-02T03:07:30 wardtimothy 7 79
59493 2017-01-25T06:39:51 xevans 27 74
143746 2017-02-27T22:40:25 aaron53 18 83
39847 2017-01-17T10:05:42 amynichols 20 64
20457 2017-01-09T16:02:10 jonessarah 22 84
123208 2017-02-19T17:19:59 jperkins 15 70
80161 2017-02-02T13:27:04 allenjones 6 72
145244 2017-02-28T13:00:35 davidreese 19 72
```

```python
[4]: df.dtypes
```

```bash
timestamp object
username object
temperature int64
heartrate int64
build object
latest int64
note object
dtype: object
```

5.9.3 3. Create a constraint object with `discover_constraints`

```python
[5]: constraints = discover_constraints(df)
```

```python
[6]: constraints
```

```bash
<tdda.constraints.base.DatasetConstraints at 0x1156b82e8>
```

```python
[7]: constraints.fields
```

```bash
Fields([('timestamp', <tdda.constraints.base.FieldConstraints at 0x1156b8518>),
          ('username', <tdda.constraints.base.FieldConstraints at 0x1156b8710>),
          ('temperature',
```

(continues on next page)
5.9.4 4. Writing the constraints to a file

[8]: with open('..../data/ignore-iot_constraints.tdda', 'w') as f:
    f.write(constraints.to_json())

[9]: cat ..../data/ignore-iot_constraints.tdda

```json
{
    "creation_metadata": {
        "local_time": "2020-07-06 14:14:58",
        "utc_time": "2020-07-06 12:12:58",
        "creator": "TDDA 1.0.31",
        "host": "eve.local",
        "user": "veit",
        "n_records": 146397,
        "n_selected": 146397
    },
    "fields": {
        "timestamp": {
            "type": "string",
            "min_length": 19,
            "max_length": 19,
            "max_nulls": 0,
            "no_duplicates": true
        },
        "username": {
            "type": "string",
            "min_length": 3,
            "max_length": 21,
            "max_nulls": 0
        },
        "temperature": {
            "type": "int",
            "min": 5,
            "max": 29,
            "sign": "positive",
            "max_nulls": 0
        },
        "heartrate": {
            "type": "int",
            "min": 60,
            "max": 89,
            "sign": "positive",
            "max_nulls": 0
        }
    }
}```
5.9.5 5. Checking dataframes with verify_df


[11]: v = verify_df(new_df, '../data/ignore-iot_constraints.tdda')
5.10 Hypothesis: property based testing

In this notebook we use property based tests to find problems in our code. Hypothesis is a library similar to Haskell’s Quickcheck. Later we will get to know it more closely together with other test libraries: Hypothesis. Hypothesis can also provide mock objects and tests for Numpy data types.

5.10.1 1. Imports

[1]: from hypothesis import given, assume
from hypothesis.strategies import tuples, integers, emails
import re

5.10.2 2. Find range

[2]: def calculate_range(tuple_obj):
    return max(tuple_obj) - min(tuple_obj)

5.10.3 3. Test with strategies and given

[3]: @given(tuples(integers(), integers(), integers()))
def test_calculate_range(tup):
    result = calculate_range(tup)
    assert isinstance(result, int)
    assert result > 0

[4]: test_calculate_range()

Falsifying example: test_calculate_range(
    tup=(0, 0, 0),
)

-----------------------------------
AssertionError Traceback (most recent call last)
<ipython-input-4-870cdf3c4b4e> in <module>
----> 1 test_calculate_range()

<ipython-input-3-452e470cf7c7> in test_calculate_range()
 1 @given(tuples(integers(), integers(), integers()))
----> 2 def test_calculate_range(tup):
 3     result = calculate_range(tup)
 4     assert isinstance(result, int)
 5     assert result > 0

[... skipping hidden 1 frame]

<ipython-input-3-452e470cf7c7> in test_calculate_range(tup)
 3     result = calculate_range(tup)
 4     assert isinstance(result, int)
----> 5     assert result > 0

(continues on next page)
AssertionError:

5.10.4 3. Correct the test with $\geq$

[5]: `@given(tuples(integers(), integers()))`
def test_calculate_range(tup):
    result = calculate_range(tup)
    assert isinstance(result, int)
    assert result $\geq$ 0

[6]: test_calculate_range()

5.10.5 4. Check against regular expressions

[7]: `def parse_email(email):
    result = re.match('(?P<username>[\w]+).(?P<domain>[\w.]+)',
                      email).groups()`
    return result

[8]: `@given(emails())`
def test_parse_email(email):
    result = parse_email(email)
    #print(result)
    assert len(result) == 2
    assert '.' in result[1]

[9]: test_parse_email()

Falsifying example: test_parse_email(
    email='/@A.ac',
)

---------------------------------------------------------------------------
AttributeError Traceback (most recent call last)
<ipython-input-9-e5ad7429fbd5> in <module>
    1 test_parse_email()
<ipython-input-8-a21f35eebf98> in test_parse_email(email)
     1 @given(emails())
----> 2 def test_parse_email(email):
     3     result = parse_email(email)
     4     #print(result)
     5     assert len(result) == 2

[... skipping hidden 1 frame]

<ipython-input-8-a21f35eebf98> in test_parse_email(email)
     1 @given(emails())

(continues on next page)
```python
def test_parse_email(email):
    result = parse_email(email)
    #print(result)
    assert len(result) == 2

<ipython-input-7-7c9f64d6175d> in parse_email(email)
    1 def parse_email(email):
    2     result = re.match('\(?P<username>\w+\).\(?P<domain>\[\w\.]\)+',
    ---> 3     email).groups()
    4     return result

AttributeError: 'NoneType' object has no attribute 'groups'
```

```
def parse_email(email):
    result = re.match('(?P<username>\\..\-~#$\&\%\{\}\+\\\^`=\*')+(?P<domain>\[\w\.\-]+)', email).groups()
    return result

[11]: test_parse_email()

    Hint:

    On the website regex101 you can first try out your regular expressions.
```
VISUALISE DATA

We have outsourced the visualisation of data to a separate tutorial: PyViz Tutorial.
Refactoring is understood as the improvement of the source code, whereby the results should remain unchanged. A distinction is made between two main goals:

### 7.1 Check and improve code quality and complexity

Before you start refactoring, you should measure the complexity of your code. In the following, I would like to introduce you to some tools and concepts that check the complexity of your code and simplify the maintenance and care of Python packages and other source code. Often, together with Git pre-commit hooks, the code quality can also be checked and improved automatically.

**See also:**

- PyCQA Meta Documentation
- github.com/PyCQA

#### 7.1.1 Checker

**flake8**

flake8 ensures that most of your code follows PEP 8. However, automatic formatting, for example with Black, is even more convenient. In addition **flake8** also checks for unused imports.

**Installation**

```
$ spack env activate python-374
$ spack install py-flake8 ^python@3.7.4
```
Check

$ flake8 path/to/your/code

Configuration

flake8 can be configured for tox in the tox.ini file of a package, e.g.:

```ini
[tox]
envlist = py37, py38, flake8, docs

[testenv:flake8]
basepython = python
deps =
    flake8
    flake8-isort
commands =
    flake8 src tests setup.py conftest.py docs/conf.py
```

See also:

- Configuring flake8
- flake8 error/violation codes
- pycodestyle error codes

check-manifest

cHECK-MANIFEST is a tool with which you can quickly check whether the file Manifest.in for Python packages is complete.

Installation

$ pipenv install check-manifest

Check

$ cd /path/to/MANIFEST.in
$ pipenv run check-manifest

... or for an automatic update

$ pipenv run check-manifest -uv
listing source files under version control: 6 files and directories
building an sdist: check-manifest-0.7.tar.gz: 4 files and directories
lists of files in version control and sdist do not match!
missing from sdist:
    tests.py
(continues on next page)
suggested MANIFEST.in rules:
   include *.py
   include tox.ini

updating MANIFEST.in

$ cat MANIFEST.in
include *.rst

# added by check_manifest.py
include *.py
include tox.ini

Configuration

You can configure check-manifest so that certain file patterns are ignored by creating a section [tool.
check-manifest] in your pyproject.toml file or a section [check-manifest] in your setup.cfg or tox.ini
file, for example:

[tool.check-manifest]
ignore = [".travis.yml"]

# setup.cfg or tox.ini
[check-manifest]
ignore =
   .travis.yml

check-manifest knows the following options:

ignore A list of filename patterns that are ignored by check-manifest. Use this option if you want to keep files in
your version control system that shouldn’t be in your source distributions. The standard list is:

PKG-INFO
   *.egg-info
   *.egg-info / *
setup.cfg
   .hgtags
   .hgsigs
   .hgignore
   .gitignore
   .bztignore
   .gitattributes
   .github / *
   .travis.yml
   Jenkinsfile
   *.mo

ignore-default-rules If true, your ignore entries replace the standard list instead of completing it.

ignore-bad-ideas A list of filename patterns that will be ignored by checking the generated files. This allows you
to keep generated files in your version control system, even if this is usually a bad idea.

7.1. Check and improve code quality and complexity
Integration with version control

With Git pre-commit hooks, check-manifest can be part of your Git workflow. To do this, add the following to your .pre-commit-config.yaml file:

```yaml
repos:
- 
  repo: https://github.com/mgedmin/check-manifest
  rev: "0.39"

hooks:
- 
  id: check-manifest
```

Mypy

With Mypy you can do a static type check.

See also:

- Home
- GitHub
- Docs
- PyPI

Installation

Mypy requires Python3.5. Then it can be installed, e.g. with:

```
$ pipenv install mypy
```

Check

Then you can check it, e.g. with:

```
$ pipenv run mypy myprogram.py
```

Note: Although Mypy needs to be installed with Python3, it can also parse Python2 code, e.g. with:

```
$ pipenv run mypy --py2 myprogram.py
```
Pytype

Pytype is a static analysis tool that derives types from your Python code without the need for type annotations. However, it can also enforce type annotations that are in the code. Although annotations are optional for Pytype, they are checked and applied if they are present. The type annotations generated by Pytype are stored in standalone .pyi files, which can be merged back into Python using merge-pyi. Finally, it flags common errors such as misspelled attribute names or function calls and much more, even across file boundaries.

See also:
  • Home
  • GitHub
  • PyPI
  • User guide
  • FAQ

Requirements

  • All common Linux distributions are supported
  • MacOS 10.7 and Xcode 8
  • Windows with WSL. In addition, the following libraries must be installed:

  
  ```bash
  $ sudo apt install build-essential python3-dev libpython3-dev
  ```

Installation

Pytype can be easily installed with

```bash
$ pipenv install pytype
```

The installation can then be checked with

```bash
$ pipenv run pytype file_or_directory
```

Configuration

For a Python package, you can set up Pytype by creating a pytype.cfg file with

```bash
$ pipenv run pytype --generate-config pytype.cfg
```

This then starts with e.g.

```bash
# NOTE: All relative paths are relative to the location of this file.

[pytype]

# Space-separated list of files or directories to exclude.
exclude =
```

(continues on next page)
Now you can customise the configuration file according to your requirements.

**Additional scripts**

- **annotate-ast** in-progress type annotator for ASTs
- **merge-pyi** Merge type information from a .pyi file into a Python file
- **pytd-tool** parser for .pyi files
- **pytype-single** debugging tool for pytype developers that analyses a single python file assuming that .pyi files have already been generated for all dependencies
- **pyxref** cross-references generator

**Wily**

The *Zen of Python*\(^1\) emphasises complexity reduction in many ways:

- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.

Wily is a command line tool for checking the complexity of Python code in tests and applications. For this purpose, Wily uses the following metrics:

**Cyclomatic complexity** measures the complexity of code by the number of linearly independent paths in the control flow graph.

The Software Engineering Institute at Carnegie Mellon University distinguishes the following four levels of risk\(^2\):

<table>
<thead>
<tr>
<th>Cyclomatic complexity</th>
<th>Risk assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–10</td>
<td>Simple programme without much risk</td>
</tr>
<tr>
<td>11–20</td>
<td>moderate risk</td>
</tr>
<tr>
<td>21–50</td>
<td>complex, high-risk programme</td>
</tr>
<tr>
<td>&gt; 50</td>
<td>untestable programme with very high risk</td>
</tr>
</tbody>
</table>

Halstead complexity measures

- Statically analysing procedure that calculates the difficulty of the programme, the effort and the implementation time from the number of operators and operands.

**Maintainability Index** is based on the cyclomatic complexity, the Halstead complexity measures and the number of lines of code\(^3\):

---

1. PEP 20 – The Zen of Python
3. Using Metrics to Evaluate Software System Maintainability
## Installation

Wily can be easily installed with

```
$ pipenv install wily
```

You can then check the installation with

```
$ pipenv run wily --help
```

**Usage:** wily [OPTIONS] COMMAND [ARGS]...

```
Version: 1.19.0
```

```
Inspect and search through the complexity of your source code. To get started, run setup:
```

```
$ wily setup ...
```

## Configuration

A `wily.cfg` file can be created in the project directory with the list of available operators:

```
[wily]
# list of operators, choose from cyclomatic, maintainability, mccabe and raw
operators = cyclomatic,raw
# archiver to use, defaults to git
archiver = git
# path to analyse, defaults to .
path = /path/to/target
# max revisions to archive, defaults to 50
max_revisions = 20
```

Python code in .ipynb files is also usually recognised automatically. However, you may be able to disable this for a Jupyter notebook with

```
ipython_support = false
```

or for individual cells with
Use

... as a command line tool

1. Building a cache with the statistics of the project

   Note: Wily assumes that your project folder is a Git repository. However, Wily does not create a cache if the working directory is dirty.

   $ pipenv run wily build

2. Show metric

   $ pipenv run wily report

   This outputs both the metric and the delta to the previous revision.

3. Show ranking

   $ pipenv run wily rank

   This shows the ranking of all files in a directory or a single file based on the specified metric, if present in .wily/.

4. Show graph

   $ pipenv run wily graph

   This displays a graph in the default browser.

5. Show build directory information

   $ pipenv run wily index

6. List the metrics available in the Wily operators

   $ pipenv run wily list-metrics

... as pre-commit hook

You can also use Wily as a pre-commit hook. To do this, you would have to add the following to the pre-commit-config.yaml configuration file, for example:

repos:
  - repo: local
    hooks:
      - id: wily
        name: wily
        entry: wily diff
        verbose: true

(continues on next page)
... in a CI/CD pipeline

Usually Wily compares the complexity with the previous revision. However, you can also specify other references, e.g. HEAD^1 with

```bash
$ pipenv run wily build src/
$ pipenv run wily diff src/ -r HEAD^1
```

PyRe

PyRe (Python Reliability) analyses the structural reliability of Python code and summarises it in a report. Currently, however, only first-order randomness methods such as Crude Monte Carlo Simulation and Importance Sampling are supported.

See also:
- Docs
- GitHub

Installation

```bash
$ pipenv install git+git://github.com/hackl/pyre.git
```

Reliability analysis

A FORM (first-order reliability method) analysis can lead to the following result, for example:

```
==============================================
RESULTS FROM RUNNING FORM RELIABILITY ANALYSIS
Number of iterations: 17
Reliability index beta: 1.75397614074
Failure probability: 0.039717297753
Number of calls to the limit-state function: 164
==============================================
```
Pysa

The Python Static Analyzer Pysa performs taint analysis to identify potential security problems. Pysa traces data streams from their origin to their endpoint and identifies vulnerable code.

See also:

- What Is Taint Analysis and Why Should I Care?
- How Pysa works
- Running Pysa
- Pysa Tutorial

Configuration

Pysa uses two file types for configuration:

- a `taint.config` file in JSON format, in which sources, sinks, features and rules are defined.

```json
{
    "comment": "UserControlled, Test, Demo sources are predefined. Same for Demo, Test and RemoteCodeExecution sinks",
    "sources": [],
    "sinks": [],
    "features": [],
    "rules": []
}
```

- files with the extension `.pysa` in a directory configured with `taint_models_path` in your `.pyre_configuration` file.

You can find practical examples in the Pyre repository.

Use

Pysa can be called, e.g. with

```
$ pipenv run pyre analyze --save-results-to ./
```

The `--save-results-to` option stores detailed results in `.taint-output.json`.

Pysa postprozessor

Installation

```
$ pipenv install fb-sapp
```
Use

1. Parsing the JSON file, e.g. with

   `$ pipenv run sapp --database-name sapp.db analyze ./taint-output.json`

   The results are stored in the local SQLite file `sapp.db`.

2. Exploring the problems with

   `$ pipenv run sapp --database-name sapp.db explore`

   This starts an IPython interface connected to the SQLite database:
   - `issues` lists all issues
   - `issue 1` selects the first issue
   - `trace` shows the data flow from source to sink
   - `n` jumps to the next call
   - `list` shows the source code of the call
   - `jump 1` jumps to the first call and shows the source code

   Further commands can be found in Commands.

### 7.1.2 Formatter

**Black**

Black formats your code in a nice and deterministic format.

**See also:**

Was lesbaren Code auszeichnet, ist gut beschrieben im Trey Hunners Blog-Post Craft Your Python Like Poetry.

**Installation**

```
$ pipenv install black
```

**Check**

Then you can check the installation with

```
$ pipenv run black /path/to/your/source/file
```
Integration

With jupyter-black you can already use Black in your Jupyter notebooks.

See also:
Integration into other editors such as PyCharm, Wing IDE or Vim is also possible, see Editor integration

Configuration

In contrast to Black’s standard 88-character formatting, however, I prefer a line length of 79 characters.
For this you can enter the following in `pyproject.toml`:

```
[tool.black]
line-length = 79
```

See also:
You can get more information about the configuration of Black in the Toml file in `pyproject.toml`.

isort

isort formats your import statements in separate and sorted blocks.

Installation

```
$ pipenv install isort
```

Configuration

isort can be configured e.g. in the `pyproject.toml` file:

```
[tool.isort]
atomic=true
force_grid_wrap=0
include_trailing_comma=true
lines_after_imports=2
lines_between_types=1
multi_line_output=3
not_skip="__init__.py"
use_parentheses=true

known_first_party="jupyter-tutorial"
known_third_party=["mpi4py", "numpy", "requests"]
```

In order to recognise third-party packages for your project imports, you can either install your project together with isort or use seed-isort-config.

Note: With isort 5 you can use profiles. This simplifies the configuration of isort in order to continue to play with Black in the future:
prettier

prettier offers automatic formatters for other file types, including TypeScript, JSON, Vue, YAML, TOML and XML.

### Installation

```
$ npm install prettier --save-dev --save-exact
```

### Configuration

```
$ npx prettier --write path/to/my/file.js
```

#### Pre-commit hook for prettier

### Installation

```
$ npm install pretty-quick husky --save-dev
```

### Configuration

In the `package.json` file you can configure the pre-commit hook as follows:

```
{ "husky": { "hooks": { "pre-commit": "pretty-quick --staged" } } }
```

See also:

- Prettier docs

### 7.1.3 Refactoring

#### Code-Smells and Anti-Patterns

See also:

- Effective Python by Brett Slatkin
Functions that should be objects

In addition to object-oriented programming, Python also supports procedural programming using functions and inheritable classes. Both paradigms should, however, be applied to the appropriate problems.

Typical symptoms of functional code that should be converted to classes are

- similar arguments across functions
- high number of distinct Halstead operands
- mix of mutable and immutable functions

For example, three functions with ambiguous usage can be reorganised so that `load_image()` is replaced by `__init__`, `crop()` becomes a class method, and `get_thumbnail()` a property:

Objects that should be functions

Sometimes, however, object-oriented code should also be better broken down into functions, e.g. if a class contains only one other method apart from `__init__()` or only static methods.

Note: You do not have to search for such classes manually, but there is a pylint rule for it:

```bash
$ pipenv run pylint --disable=all --enable-R0903 requests
************* Module requests.auth
requests/auth.py:72:0: R0903: Too few public methods (1/2) (too-few-public-methods)
requests/auth.py:100:0: R0903: Too few public methods (1/2) (too-few-public-methods)
************* Module requests.models
requests/models.py:60:0: R0903: Too few public methods (1/2) (too-few-public-methods)
-----------------------------------
Your code has been rated at 9.99/10
```

This shows us that two classes with only one public method have been defined in `auth.py`, in lines 72ff. and 100ff. Also in `models.py` there is a class with only one public method from line 60.

Nested code

«Flat is better than nested.»
– Tim Peters, Zen of Python

Nested code makes it difficult to read and understand. You need to understand and remember the conditions as you go through the nestings. Objectively, the cyclomatic complexity increases as the number of code branches increases.

You can reduce nested methods with multiple nested if statements by replacing levels with methods that return `False` if necessary. Then you can use `.count()` to check if the number of errors is > 0.

Another possibility is to use list comprehensions. This way the code

```python
results = []
for item in iterable:
    if item == match:
        results.append(item)
```
can be replaced by

```python
results = [item for item in iterable if item == match]
```

**Note:** The `itertools` of the Python standard library are often also good for reducing the nesting depth by creating functions to create iterators from data structures. You can also filter with `itertools`, e.g. with `filterfalse`.

---

**Query tools for complex dicts**

JMESPath, glom, asq and flupy can significantly simplify the query of dicts in Python.

**Reduce code with dataclasses and attrs**

`dataclasses` were introduced in Python 3.7 and there is also a backport for Python 3.6. They are meant to simplify the definition of classes that are mainly created to store values and can then be accessed via attribute search. Some examples are `collection.namedtuple`, `typing.NamedTuple`, Recipes to Records\(^1\) and Nested Dicts\(^2\). Data classes save you from writing and managing these methods.

**See also:**

- PEP 557 – Data Classes

`attrs` is a Python package that has been around much longer than `dataclasses`, is more comprehensive and can also be used with older versions of Python.

---

**Rope**

**Installation**

Rope can be easily installed with

```
$ pipenv install rope
```

**Use**

Now we first import the `Project` type and instantiate it with the path to the project:

```
[1]: from rope.base.project import Project
proj = Project('requests')
```

This creates a project folder named `.ropeproject` in our project.

```
[2]: [f.name for f in proj.get_files()]
```

---

\(^1\) Records (Python recipe)

\(^2\) Dot-style nested lookups over dictionary based data structures (Python recipe)

---

7.1. Check and improve code quality and complexity
The `proj` variable can execute a number of commands such as `get_files` and `get_file`. In the following example we use this to assign the variable `api` to the file `api.py`.

```
[3]: !cp requests/api.py requests/api_v1.py

[4]: api = proj.get_file('api.py')

[5]: from rope.refactor.rename import Rename
    change = Rename(proj, api).get_changes('api_v1')
    proj.do(change)

[6]: !cd requests && git status
    On branch master
    Changes not staged for commit:
      (use "git add <file>..." to update what will be committed)
      (use "git restore <file>..." to discard changes in working directory)
        modified:   __init__.py

    Untracked files:
      (use "git add <file>..." to include in what will be committed)
        .ropeproject/
        api_v1.py

    Changes not staged for commit (use "git add" and/or "git commit -a")
```

```
[7]: !cd requests && git diff __init__.py
    diff --git a/__init__.py b/__init__.py
    index f8f9429..502e33a 100644
    --- a/__init__.py
    (continues on next page)
```
With `proj.do(change)`, the file `requests/__init__.py` has been changed to import from `new_api` instead of `api`. Rope can be used not only for renaming files, but also in various other cases; see also Rope Refactorings.

### 7.2 Performance measurement and optimisation

#### 7.2.1 Performance measurement

However, once you have worked with your code, it can be useful to examine its efficiency more closely. The *IPython Profiler* or *scalene* can be used for this.

**IPython Profiler**

IPython provides access to a wide range of functions to measure times and create profiles. The following magic IPython commands are explained here:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%time</td>
<td>Time to execute a single statement</td>
</tr>
<tr>
<td>%timeit</td>
<td>Average time it took to execute a single statement repeatedly</td>
</tr>
<tr>
<td>%prun</td>
<td>Run code with the profiler</td>
</tr>
<tr>
<td>%mprun</td>
<td>Run code with the line-by-line profiler</td>
</tr>
<tr>
<td>%memit</td>
<td>Measure the memory usage of a single statement</td>
</tr>
<tr>
<td>%mprun</td>
<td>Executes the code with the line-by-line memory profiler</td>
</tr>
</tbody>
</table>

The last four commands are not contained in IPython itself, but in the modules `line_profiler` and `memory_profiler`.

**%timeit and %time**

We saw the `%timeit` line and `%%timeit` cell magic in the introduction of the magic functions in IPython magic commands. They can be used to measure the timing of the repeated execution of code snippets:

```
[1]: %timeit sum(range(100))
5.7 μs ± 1.39 μs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Note that `%timeit` executes the execution multiple times in a loop. If the number of loops is not specified with `-n`, `%timeit` automatically adjusts the number so that sufficient measurement accuracy is achieved.
Sometimes repeating an operation is not the best option, e.g. when we have a list that we want to sort. Here we may be misled by repeated surgery. Sorting a presorted list is much faster than sorting an unsorted list, so repeating it distorts the result:

```python
import random
L = [random.random() for i in range(100000)]
%timeit L.sort()
```

The slowest run took 4.20 times longer than the fastest. This could mean that an intermediate result is being cached.

17.3 ms ± 9.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

Then the `timeit` function might be a better choice. `timeit` should also be the better choice for long-running commands, when short system-related delays are unlikely to affect the result:

```python
import random
L = [random.random() for i in range(100000)]
%time L.sort()
```

CPU times: user 29.9 ms, sys: 259 µs, total: 30.2 ms
Wall time: 182 ms

Sorting an already sorted list:

```python
%time L.sort()
```

CPU times: user 2.28 ms, sys: 30 µs, total: 2.31 ms
Wall time: 3 ms

Note how much faster the pre-sorted list is to be sorted, but also note how much longer the timing with `timeit` compared to `timeit`, even for the pre-sorted list. This is due to the fact that `timeit` is doing some clever things to keep system calls from interfering with the timing. This prevents, for example, the garbage collection of Python objects that are no longer used and that could otherwise affect the time measurement. Because of this, the `timeit` results are usually noticeably faster than the `time` results.

### Profiling for scripts: `%prun`

A program is made up of many individual instructions, and sometimes it is more important to measure those instructions in context than to measure them yourself. Python includes a built-in Code-Profiler. However, IPython offers a much more convenient way to use this profiler in the form of the magic function `%prun`.

As an example, let’s define a simple function that does some calculations:

```python
def sum_of_lists(N):
    total = 0
    for i in range(5):
        L = [j ^ (j >> i) for j in range(N)]
```
In the notebook the output looks something like this:

| 14 function calls in 9.597 seconds |

<table>
<thead>
<tr>
<th>Ordered by: internal time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncalls</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

The result is a table that shows the execution time for each function call, sorted by total time. In this case, most of the time is consumed with list comprehension within `sum_of_lists`. This gives us clues as to where we could improve the efficiency of the algorithm.

### Profiling line by line: `%prun`

Profiling with `%prun` is useful, but sometimes a line-by-line profile report is more insightful. This isn’t built into Python or IPython, but there is a package available, line_profiler, that enables this. This can be provided in your kernel with

```bash
$ spack env activate python-374
$ spack install py-line-profiler ^python@3.7.4%gcc@9.1.0
```

Alternatively, you can install line_profiler with other package managers, e.g.

```bash
$ pipenv install line_profiler
```

If you are using Python 3.7.x and get the error message `error: command 'clang' failed with exit status 1`, the only thing left for now is to install Cython together with the resources from the Git repository:

```bash
$ pipenv install Cython git+https://github.com/rkern/line_profiler.git#egg=line_profiler
```

Now you can load IPython with the `line_profiler` extension:

```
[8]: %load_ext line_profiler
```

The `%prun` command profiles each function line by line. In this case, you must explicitly specify which functions are of interest for creating the profile:

---

7.2. Performance measurement and optimisation
The result looks something like this:

```
Timer unit: 1e-06 s
Total time: 0.015145 s
File: <ipython-input-6-f105717832a2>
Function: sum_of_lists at line 1
```

<table>
<thead>
<tr>
<th>Line #</th>
<th>Hits</th>
<th>Time</th>
<th>Per Hit</th>
<th>% Time</th>
<th>Line Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>total = 0</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>11.0</td>
<td>1.8</td>
<td>0.1</td>
<td>for i in range(5):</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>14804.0</td>
<td>2960.8</td>
<td>97.7</td>
<td>L = [j ^ (j &gt;&gt; i) for j in␣</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-range(N)]</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>329.0</td>
<td>65.8</td>
<td>2.2</td>
<td>total += sum(L)</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>return total</td>
</tr>
</tbody>
</table>
```

The time is given in microseconds and we can see which line the function spends most of its time on. We may then be able to modify the script in such a way that the efficiency of the function can be increased.

More information about `%lprun` and the available options can be found in the IPython help function `%lprun?`.

**Create a storage profile: `%memit` and `%mprun`**

Another aspect of profiling is the amount of memory that an operation uses. This can be evaluated with another IPython extension, the `memory_profiler`. This can be provided in your kernel with

```
$ spack env activate python-374
$ spack install py-memory-profiler ^python@3.7.4%gcc@9.1.0
```

Alternatively you can install `memory_profiler` with other package managers, e.g.

```
$ pipenv install memory_profiler
```

For a line-by-line description of memory usage, we can use the `%mprun` magic. Unfortunately, this magic only works for functions that are defined in separate modules and not for the notebook itself. So we first use the `%file` magic to create a simple module called `mprun_demo.py` that contains our `sum_of_lists` function.

```
[%file mprun_demo.py]
from memory_profiler import profile
@profile
```

We see that this feature occupies approximately 100 MB of memory.

For a line-by-line description of memory usage, we can use the `%mprun` magic. Unfortunately, this magic only works for functions that are defined in separate modules and not for the notebook itself. So we first use the `%file` magic to create a simple module called `mprun_demo.py` that contains our `sum_of_lists` function.
def my_func():
    a = [1] * (10 ** 6)
    del b
    return a

Writing mprun_demo.py

from mprun_demo import my_func
%mprun -f my_func my_func()

Here the Increment column shows how much each row affects the total memory consumption: Note that when we calculate b we need about 160 MB of additional memory; however, this is not released again by deleting b.

More information about %memit, %mprun and their options can be found in the IPython help with %memit?.

pyheatmagic

pyheatmagic is an extension that allows the IPython magic command %heat to display Python code as a heatmap with Py-Heat.

It can be easily installed in the kernel with

$ pipenv install py-heat-magic Installing py-heat-magic...
Loading the extension in IPython

[14]: %load_ext heat

Display the heat map

[15]: %heat
def powfun(a, b):
    """Method to raise a to power b using pow() function."""
    return pow(a, b)

def powop(a, b):
    """Method to raise a to power b using ** operator."""
    return a ** b

def powmodexp(a, b):
    """Method to raise a to power b using modular exponentiation."""
    base = a
    res = 1
    while b > 0:
        if b & 1:
            res *= base
            base *= base
        b >>= 1
    return res

def main():
    """Test function."""
    a, b = 2377757, 773
    pow_function = powfun(a, b)
    pow_operator = powop(a, b)
    pow_modular_exponentiation = powmodexp(a, b)

    if __name__ == '__main__':
        main()
Alternatively, the heatmap can also be saved as a file, e.g. with

```bash
%%heat -o pow-heatmap.png
```

**See also:**

- Penn Machine Learning Benchmarks

---

7.2. Performance measurement and optimisation

---

![Code snippet](image)
scalene

scalene creates profiles for CPU and memory very quickly. The overhead is usually very low at 10-20%.

See also:
- GitHub
- PyPI
- scalene-paper.pdf

Installation

Linux and WSL

```bash
$ pipenv install scalene
```

MacOS

```bash
$ brew install --HEAD emeryberger/scalene/libscalene
```

Use

```bash
$ pipenv run scalene test/testme.py
```

You can display all available options with

```bash
$ pipenv run scalene --help
```

```
scalene --help
             [--profile-interval PROFILE_INTERVAL] [--cpu-only]
             [--profile-all] [--profile-only PROFILE_ONLY]
             [--use-virtual-time]
             [--cpu-percent-threshold CPU_PERCENT_THRESHOLD]
             [--cpu-sampling-rate CPU_SAMPLING_RATE]
             [--malloc-threshold MALLOC_THRESHOLD]

Scalene: a high-precision CPU and memory profiler.
         https://github.com/emeryberger/scalene

% scalene yourprogram.py
```

(continues on next page)
7.2.2 Parallel programming

Three examples of Threading, Multiprocessing and Async <> asyncio-example illustrate the rules and best practices for parallel programming.

Introduction to concurrency

When developing code, there can often be trade-offs between different implementations. However, at the beginning of the development of an algorithm, it is usually counterproductive to worry about the efficiency of the code.

«We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.»¹

Martelli’s model of scalability

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Single thread and single process</td>
</tr>
<tr>
<td>2–8</td>
<td>Multiple threads and multiple processes</td>
</tr>
<tr>
<td>&gt;8</td>
<td>Distributed processing</td>
</tr>
</tbody>
</table>

Martelli’s observation: In the course of time, the second category becomes more and more insignificant: individual cores are becoming more and more powerful and large data sets are getting bigger and bigger.

¹ Donald Knuth, founder of Literate Programming, in Computer Programming as an Art (1974)
Global Interpreter Lock (GiL)

CPython has a lock on its internally shared global state. As a result, no more than one thread can run at the same time. The GiL is not a big problem for I/O-heavy applications; however, using threading will slow down CPU-heavy applications. Accordingly, multi-processing is exciting for us to get more CPU cycles.

Literate programming and Martelli’s model of scalability determined the design decisions on Python’s performance for a long time. Little has changed in this assessment to this day: Contrary to intuitive expectations, more CPUs and threads in Python initially lead to less efficient applications. However, the Gilectomy project, which was supposed to replace the GiL, also encountered another problem: the Python C API exposes too many implementation details. With this, however, performance improvements would quickly lead to incompatible changes, which then seem unacceptable, especially in a language as popular as Python. Nevertheless, there are already some solutions:

- Numba is a JIT compiler that translates mainly scientific Python and NumPy code into fast machine code.
- PyPy with a more universal JIT compiler, but which has to emulate existing C extension like NumPy, which is really inefficient.

Faster CPython

At PyCon US in May 2021, Guido van Rossum presented Faster CPython, a project that aims to double the speed of Python 3.11. The cooperation with the other Python core developers is regulated in PEP 659 – Specializing Adaptive Interpreter. There is also an open issue tracker and various tools for collecting bytecode statistics. CPU-intensive Python code in particular is likely to benefit from the changes; code already written in C, I/O-heavy processes and multithreaded code, on the other hand, are unlikely to benefit.
Multithreading, multiprocessing and asynchronous communication

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Multithreading</th>
<th>Multiprocessing</th>
<th>asyncio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separation</td>
<td>Threads share one state. However, sharing a state can lead to race conditions, i.e. the result of an operation can depend on the timing of certain individual operations.</td>
<td>The processes are independent of each other. If they are to communicate with each other, interprocess communication (IPC), object pickling and other overhead is necessary.</td>
<td>With \texttt{run_coroutine_threadsafe()}, asyncio objects can also be used by other threads. Almost all asyncio objects are not thread-safe.</td>
</tr>
<tr>
<td>Switching</td>
<td>Threads change preemptively, i.e. no explicit code needs to be added to cause a change of tasks. However, such a change is possible at any time; accordingly, critical areas must be protected with lock.</td>
<td>As soon as you get a process assigned, significant progress should be made. So you should not make too many roundtrips back and forth.</td>
<td>asyncio switches cooperatively, i.e. \texttt{yield} or \texttt{await} must be explicitly specified to cause a switch. You can therefore keep the effort to these changes very low.</td>
</tr>
<tr>
<td>Tooling</td>
<td>Threads require very little tooling: Lock and Queue. Locks are difficult to understand in non-trivial examples. For complex applications, it is therefore better to use atomic message queues or asyncio.</td>
<td>Simple tooling with \texttt{map} and \texttt{imap_unordered} among others, to test individual processes in a single thread before switching to multiprocessing. If IPC or object pickling is used, the tooling becomes more complex.</td>
<td>At least for complex systems, asyncio leads to the goal more easy than multithreading locks. However asyncio requires a large set of tools: futures, Event Loops and non-blocking versions of almost everything.</td>
</tr>
<tr>
<td>Performance</td>
<td>Multithreading produces the best results for IO-heavy tasks. The performance limit for threads is one CPU minus task switches and synchronisation overheads.</td>
<td>The processes can be distributed to several CPUs and should therefore be used for CPU-heavy tasks. However, additional effort may be required and synchronisation of the processes.</td>
<td>Calling a poor Python function takes more overhead than requesting a generator or awaitable – i.e., asyncio can utilise the CPU efficiently. For CPU-intensive tasks, however, multiprocessing is more suitable.</td>
</tr>
</tbody>
</table>

Summary

There is no one ideal implementation of concurrency – each of the approaches presented next has specific advantages and disadvantages. So before you decide which approach to follow, you should analyse the performance problems carefully and then choose the most suitable solution. In our projects, we often use several approaches, depending on the part for which the performance is to be optimised.
Threading example

Updating and displaying a counter:

```
[1]: counter = 0

print('Starting up')
for i in range(10):
    counter += 1
    print('The count is %d' % counter)
print('Finishing up')
```

Start with code that is clear, simple, and top-down. It’s easy to develop and incrementally testable.

**Note:** Test and debug your application before starting threading. Threading never makes debugging easier.

Convert to functions

The next step is to create reusable code as a function:

```
[2]: counter = 0

def worker():
    'My job is to increment the counter and print the current count'
    global counter

    counter += 1
    print('The count is %d' % counter)

print('Starting up')
for i in range(10):
    worker()
print('Finishing up')
```

(continues on next page)
The count is 6
The count is 7
The count is 8
The count is 9
The count is 10
Finishing up

Multi-Threading

Now some worker threads can be started:

```
[3]: import threading

counter = 0

def worker():
    'My job is to increment the counter and print the current count'
    global counter

    counter += 1
    print('The count is %d' % counter)

print('Starting up')
for i in range(10):
    threading.Thread(target=worker).start()
print('Finishing up')
```

Starting up
The count is 1
The count is 2
The count is 3
The count is 4
The count is 5
The count is 6
The count is 7
The count is 8
The count is 9
The count is 10
Finishing up
Test

A simple test run leads to the same result.

Detection of race conditions

Note: Tests cannot prove the absence of errors. Many interesting race conditions do not show up in test environments.

Fuzzing

Fuzzing is a technique to improve the detection of race conditions:

```
[4]: import threading, time, random

FUZZ = True

def fuzz():
    if FUZZ:
        time.sleep(random.random())

counter = 0

def worker():
    'My job is to increment the counter and print the current count'
    global counter

    fuzz()
    oldcnt = counter
    fuzz()
    counter = oldcnt + 1
    fuzz()
    print('The count is %d' % counter, end='')
    fuzz()

print('Starting up')
    fuzz()
for i in range(10):
    threading.Thread(target=worker).start()
    fuzz()
print('Finishing up')
    fuzz()

Starting up
The count is 1
The count is 2
The count is 2
The count is 3
The count is 4
The count is 4
The count is 4
Finishing up
```

This technique is limited to relatively small blocks of code and is imperfect in that it still cannot prove the absence of errors. Nevertheless, fuzzed tests can reveal race conditions.
Careful threading with queues

The following rules must be observed:

1. All shared resources should be executed in exactly one thread. All communication with this thread should be done with only one atomic message queue – usually with the `queue` module, email or message queues such as RabbitMQ or ZeroMQ.

Resources that require this technology include global variables, user inputs, output devices, files, sockets, etc.

2. One category of sequencing problems is to ensure that step A is performed before step B. The solution is to run them both on the same thread, with all the actions happening in sequence.

3. To implement a barrier that waits for all parallel threads to complete, just join all threads with `join()`.

4. You cannot wait for daemon threads to complete (they are infinite loops); instead you should execute `join()` on the queue itself, so that the tasks are only merged when all tasks in the queue have been completed.

5. You can use global variables to communicate between functions, but only within a single-threaded program. In a multi-thread program, however, you cannot use global variables because they are mutable. Then the better solution is `threading.local()`, since it is global in a thread, but not beyond.

6. Never try to terminate a thread from the outside: you never know if that thread is holding a lock. Therefore, Python does not provide a direct thread termination mechanism. However, if you try to do this with ctypes, this is a recipe for deadlock.

Now, if we apply these rules, our code looks like this:

```python
[5]: import threading, queue

counter = 0

counter_queue = queue.Queue()

def counter_manager():
    'I have EXCLUSIVE rights to update the counter variable'
    global counter

    while True:
        increment = counter_queue.get()
        counter += increment
        print_queue.put(['The count is %d' % counter, ])
        counter_queue.task_done()

t = threading.Thread(target=counter_manager)
t.daemon = True
t.start()

del t

print_queue = queue.Queue()

def print_manager():
    while True:
        job = print_queue.get()
        for line in job:
```

(continues on next page)
print(line)
print_queue.task_done()

t = threading.Thread(target=print_manager)
t.daemon = True
t.start()
del t

```python
def worker():
    'My job is to increment the counter and print the current count'
    counter_queue.put(1)

print_queue.put(['Starting up'])
worker_threads = []
for i in range(10):
    t = threading.Thread(target=worker)
    worker_threads.append(t)
    t.start()
for t in worker_threads:
    t.join()

counter_queue.join()
print_queue.put(['Finishing up'])
print_queue.join()
```

Starting up
The count is 1
The count is 2
The count is 3
The count is 4
The count is 5
The count is 6
The count is 7
The count is 8
The count is 9
The count is 10
Finishing up

**Careful threading with locks**

If we thread with locks instead of queues, the code looks even tidier:

[6]: import threading, time, random

```python
counter_lock = threading.Lock()
printer_lock = threading.Lock()

counter = 0

def worker():
    global counter
```
with counter_lock:
    counter += 1
with printer_lock:
    print('The count is %d' % counter)

with printer_lock:
    print('Starting up')

worker_threads = []
for i in range(10):
    t = threading.Thread(target=worker)
    worker_threads.append(t)
    t.start()
for t in worker_threads:
    t.join()

with printer_lock:
    print('Finishing up')

Starting up
The count is 1
The count is 2
The count is 3
The count is 4
The count is 5
The count is 6
The count is 7
The count is 8
The count is 9
The count is 10
Finishing up

Finally, a few notes on locks:

1. Locks are just so-called flags, they are not really reliable.

2. In general, locks should be viewed as a primitive tool that is difficult to understand in non-trivial examples. For more complex applications, it is better to use atomic message queues.

3. The more locks that are set at the same time, the less the benefits of simultaneous processing.

Multi-processing example

We’ll start with code that is clear, simple, and executed top-down. It’s easy to develop and incrementally testable:

[1]: import requests
from multiprocessing.pool import ThreadPool as Pool

sites = [
    'https://github.com/veit/jupyter-tutorial/',
    'https://github.com/veit/pyviz-tutorial/',
    'https://pyviz-tutorial.readthedocs.io/de/latest/']
def sitesize(url):
    with requests.get(url) as u:
        return url, len(u.content)

pool = Pool(10)
for result in pool.imap_unordered(sitesize, sites):
    print(result)

('https://pyviz-tutorial.readthedocs.io/de/latest/', 6556)
('https://github.com/veit/pyviz-tutorial/', 164082)
('https://github.com/veit/jupyter-tutorial/', 183345)
('https://cusy.io/en', 26974)

Note 1: A good development strategy is to use map, to test your code in a single process and thread before moving to multi-processing.

Note 2: In order to better assess when ThreadPool and when process Pool should be used, here are some rules of thumb:

• For CPU-heavy jobs, multiprocessing.pool.Pool should be used. Usually we start here with twice the number of CPU cores for the pool size, but at least 4.

• For I/O-heavy jobs, multiprocessing.pool.ThreadPool should be used. Usually we start here with five times the number of CPU cores for the pool size.

• If we use Python 3 and do not need an interface identical to pool, we use concurrent.future.Executor instead of multiprocessing.pool.ThreadPool; it has a simpler interface and was designed for threads from the start. Since it returns instances of concurrent.futures.Future, it is compatible with many other libraries, including asyncio.

• For I/O- and CPU-heavy jobs, we prefer multiprocessing.Pool because it provides better process isolation.

[2]:

import requests
from multiprocessing.pool import ThreadPool as Pool

sites = [
    'https://github.com/veit/jupyter-tutorial/',
    'https://github.com/veit/pyviz-tutorial/',
    'https://pyviz-tutorial.readthedocs.io/de/latest/',
    'https://cusy.io/en',
]

def sitesize(url):
    with requests.get(url) as u:
        return url, len(u.content)

for result in map(sitesize, sites):
    print(result)
What can be parallelised?

Amdahl's law

The increase in speed is mainly limited by the sequential part of the problem, since its execution time cannot be reduced by parallelisation. In addition, parallelisation creates additional costs, such as for communication and synchronisation of the processes.

In our example, the following tasks can only be processed serially:

- UDP DNS request request for the URL
- UDP DNS response
- Socket from the OS
- TCP-Connection
- Sending the HTTP request for the root resource
- Waiting for the TCP response
- Counting characters on the site

```python
[3]: import requests
    from multiprocessing.pool import ThreadPool as Pool

sites = [
    'https://github.com/veit/jupyter-tutorial/',
    'https://github.com/veit/pyviz-tutorial/',
    'https://pyviz-tutorial.readthedocs.io/de/latest/',
    'https://cusy.io/en',
]

def sitesize(url):
    with requests.get(url, stream=True) as u:
        return url, len(u.content)

pool = Pool(4)
for result in pool.imap_unordered(sitesize, sites):
    print(result)
```
Note: `imap_unordered` is used to improve responsiveness. However, this is only possible because the function returns the argument and result as a tuple.

Tips

- Don’t make too many trips back and forth

  If you get too many iterable results, this is a good indicator of too many trips, such as in

  ```python
  >>> def sitesize(url, start):
  ...    req = urllib.request.Request()
  ...    req.add_header('Range:%d-%d' % (start, start+1000))
  ...    u = urllib.request.urlopen(url, req)
  ...    block = u.read()
  ...    return url, len(block)
  ```

- Make relevant progress on every trip

  Once you get the process, you should make significant progress and not get bogged down. The following example illustrates intermediate steps that are too small:

  ```python
  >>> def sitesize(url, results):
  ...    with requests.get(url, stream=True) as u:
  ...      while True:
  ...        line = u.iter_lines()
  ...        results.put((url, len(line))
  ```

- Don’t send or receive too much data

  The following example unnecessarily increases the amount of data:

  ```python
  >>> def sitesize(url):
  ...    with requests.get(url) as u:
  ...      return url, u.content
  ```

Threading and forking combined

Mixing multiprocessing and threading is generally problematic and a recipe for deadlocks.

The following code was entered in 2016 at https://bugs.python.org/issue27422 in the Python bug tracker:

```python
[1]:
import sys
import multiprocessing
import subprocess
from concurrent.futures import ThreadPoolExecutor

def run(arg):
    print("starting %s" % arg)
    p = multiprocessing.Process(target=print, args=("running", arg))
    p.start()
    p.join()
    print("finished %s" % arg)
```

(continues on next page)
if __name__ == "__main__":
    n = 16
    tests = range(n)
    with ThreadPoolExecutor(n) as pool:
        for r in pool.map(run, tests):
            pass

starting 0
starting 1
starting 2
starting 3
starting 4
starting 5
starting 6
starting 7
starting 8
starting 9
starting 10
starting 11
starting 12
starting 13
starting 14
starting 15

running 15
running 6
finished 15
finished 6

finished 11
finished 7
running 9
running 7
running 11
finished 15
running 8
finished 15
running 10
finished 15
running 1
finished 15
running 4
finished 15
running 12
finished 15
running 0
running 2
finished 15
running 14
running 3
finished 15
running 13
running 5

(continues on next page)
Usually, threading is recommended after the fork, not before. Otherwise, the locks used when executing the threads are duplicated across the processes. If one of these processes dies with a lock, all other processes with this lock are deadlocked.

**asyncio example**

From IPython 7.0 you can use asyncio directly in Jupyter Notebooks; see also IPython 7.0, Async REPL.

If you get RuntimeError: This event loop is already running, [nest-asyncio] might help you.

Ihr könnt das Paket installieren mit

```
$ pipenv install nest-asyncio
```

You can then import it into your notebook and use it with:

```
[1]: import nest_asyncio

nest_asyncio.apply()
```

**Simple Hello world example**

```
[2]: import asyncio

async def hello():
    print('Hello')
    await asyncio.sleep(1)
    print('world')

await hello()

Hello
world
```
A little bit closer to a real world example

```
[3]: import asyncio
import random

async def produce(queue, n):
    for x in range(1, n + 1):
        # produce an item
        print('producing {}/{}'.format(x, n))
        # simulate i/o operation using sleep
        await asyncio.sleep(random.random())
        item = str(x)
        # put the item in the queue
        await queue.put(item)

    # indicate the producer is done
    await queue.put(None)

async def consume(queue):
    while True:
        # wait for an item from the producer
        item = await queue.get()
        if item is None:
            # the producer emits None to indicate that it is done
            break

        # process the item
        print('consuming {}'.format(item))
        # simulate i/o operation using sleep
        await asyncio.sleep(random.random())

loop = asyncio.get_event_loop()
queue = asyncio.Queue(loop=loop)
asyncio.ensure_future(produce(queue, 10), loop=loop)
loop.run_until_complete(consume(queue))
```

producing 1/10
producing 2/10
consuming 1
producing 3/10
consuming 2
producing 4/10
consuming 3
producing 5/10
consuming 4
producing 6/10
consuming 5
producing 7/10
consuming 6
producing 8/10

(continues on next page)
Exception Handling

See also: set_exception_handler

```python
[4]:
def main():
    loop = asyncio.get_event_loop()
# May want to catch other signals too
    signals = (signal.SIGHUP, signal.SIGTERM, signal.SIGINT)
    for s in signals:
        loop.add_signal_handler(
            s, lambda s=s: asyncio.create_task(shutdown(loop, signal=s)))
    loop.set_exception_handler(handle_exception)
    queue = asyncio.Queue()
```

Testing with pytest

Example:

```python
[5]:
import pytest

@pytest.mark.asyncio
async def test_consume(mock_get, mock_queue, message, create_mock_coro):
    mock_get.side_effect = [message, Exception("break while loop")]

    with pytest.raises(Exception, match="break while loop"):
        await consume(mock_queue)
```

Third-party libraries

- pytest-asyncio has helpfull things like fixtures for event_loop, unused_tcp_port, and unused_tcp_port_factory; and the ability to create your own asynchronous fixtures.
- asynctest has helpful tooling, including coroutine mocks and exhaust_callbacks so we don’t have to manually await tasks.
- aiohttp has some really nice built-in test utilities.
Debugging

asyncio already has a debug mode in the standard library. You can simply activate it with the PYTHONASYNCIODEBUG environment variable or in the code with loop.set_debug(True).

Using the debug mode to identify slow async calls

asyncio’s debug mode has a tiny built-in profiler. When debug mode is on, asyncio will log any asynchronous calls that take longer than 100 milliseconds.

Debugging in production with aiodebug

aiodebug is a tiny library for monitoring and testing asyncio programs.

Example

[6]: from aiodebug import log_slow_callbacks

    def main():
        loop = asyncio.get_event_loop()
        log_slow_callbacks.enable(0.05)

Logging

aiologger allows non-blocking logging.

Asynchronous Widgets

See also: Asynchronous Widgets

[7]: def wait_for_change(widget, value):
    future = asyncio.Future()
    def getvalue(change):
        # make the new value available
        future.set_result(change.new)
        widget.unobserve(getvalue, value)
    widget.observe(getvalue, value)
    return future

[8]: from ipywidgets import IntSlider

slider = IntSlider()

async def f():
    for i in range(10):
        print('did work %s' % i)
    x = await wait_for_change(slider, 'value')
    print('async function continued with value %s' % x)
asyncio.ensure_future(f())

slider

IntSlider(value=0)

did work 0

Read more

- asyncio: We Did It Wrong by Lynn Root
- An Intro to asyncio by Mike Driscoll
- Asyncio Coroutine Patterns: Beyond await by Yeray Diaz

ipyparallel

This section provides an overview of ipyparallel which supports different types of parallelisation, including:

- Single Program, Multiple Data (SPMD)
- Multiple program, multiple data (MPMD)
- Message Passing Interface (MPI)

Installation

1. Installation

    $ pipenv install ipython[all]

2. Activate notebook server extension:

    $ pipenv run jupyter serverextension enable --py ipyparallel
    Enabling: ipyparallel.nbextension
    - Writing config: /Users/veit/.jupyter
    - Validating...
    ipyparallel.nbextension  OK

3. Install notebook extension:

    $ pipenv run jupyter nbextension install --py ipyparallel
    ...
    - Validating: OK
    To initialize this nbextension in the browser every time the notebook (or other app) loads:
    
    jupyter nbextension enable ipyparallel --py

4. Activate notebook extension:
Overview

Architecture

The IPython.parallel architecture consists of four components:

IPython-Engine

The IPython engine is an extension of the IPython kernel for Jupyter. The module waits for requests from the network, executes code and returns the results. IPython parallel extends the Jupyter messaging protocol with native Python object serialisation and adds some additional commands. Several engines are started for parallel and distributed computing.

IPython-Hub

The main job of the hub is to establish and monitor connections to clients and engines.
IPython-Schedulers

All actions that can be carried out on the engine go through a scheduler. While the engines themselves block when user code is executed, the schedulers hide this from the user to provide a fully asynchronous interface for a number of engines.

IPython-Client

There is a main object Client to connect to the cluster. Then there is a corresponding View for each execution model. These Views allow users to interact with a number of engines. The two standard views are:

- `ipyparallel.DirectView` class for explicit addressing
- `ipyparallel.LoadBalancedView` class for target-independent scheduling

Start

1. Starting the IPython Hub:

```bash
$ pipenv run ipcontroller
[IPControllerApp] Hub listening on tcp://127.0.0.1:53847 for registration.
[IPControllerApp] Hub using DB backend: 'DictDB'
[IPControllerApp] hub::created hub
[IPControllerApp] writing connection info to /Users/veit/.ipython/profile_default/
    → security/ipcontroller-client.json
[IPControllerApp] writing connection info to /Users/veit/.ipython/profile_default/
    → security/ipcontroller-engine.json
[IPControllerApp] task::using Python leastload Task scheduler
...
```

**DB backend** The database in which the IPython tasks are managed. In addition to the in-memory database DictDB, MongoDB and SQLite are further options.

- `ipcontroller-client.json` Configuration file for the IPython client
- `ipcontroller-engine.json` Configuration file for the IPython engine

**Task-Schedulers** The possible routing scheme. leastload always assigns tasks to the engine with the fewest open tasks. Alternatively, lru (Least Recently Used), plainrandom, twobin and weighted can be selected, the latter two also need Numpy.

This can be configured in `ipcontroller_config.py`, for example with `c.TaskScheduler.scheme_name = 'leastload'` or with

```bash
$ pipenv run ipcontroller --scheme=pure
```

2. Starting the IPython controller and the engines:

```bash
$ pipenv run ipcluster start
[IPClusterStart] Starting ipcluster with [daemon=False]
    → ipcluster.pid
[IPClusterStart] Starting Controller with LocalControllerLauncher
[IPClusterStart] Starting 4 Engines with LocalEngineSetLauncher
```
**Batch systems** Besides the possibility to start ipcontroller and ipengine locally, see *Starting the controller and engine on your local machine* in ssh, there are also the profiles for MPI, PBS, SGE, LSF, HTCondor, Slurm, SSH and WindowsHPC.

This can be configured in ipcluster_config.py for example with

c.IPClusterEngines.engine_launcher_class = 'SSH' or with

```bash
$ pipenv run ipcluster start --engines=MPI
```

See also:

*MPI*

3. Starting the Jupyter Notebook and loading the IPython-Parallel-Extension:

```bash
$ pipenv run jupyter notebook
[I NotebookApp] Loading IPython parallel extension
[I NotebookApp] [jupyter.nbextensions_configurator] enabled 0.4.1
[I NotebookApp] Serving notebooks from local directory: /Users/veit//jupyter-tutorial
[I NotebookApp] The Jupyter Notebook is running at:
[I NotebookApp] http://localhost:8888/?token=4e9acb8993758c2e7f3bda3b1957614c6f3528ee5e3343b3
```

4. Finally the cluster with the default profile can be started in the browser at the URL http://localhost:8888/tree/docs/parallel/ipyparallel#ipyclusters.

**Check the installation**

```python
[1]: import ipyparallel as ipp
   c = ipp.Client()
c.ids
[1]: [0, 1, 2, 3]

[2]: c[:].apply_sync(lambda : "Hello, World")
[2]: ['Hello, World', 'Hello, World', 'Hello, World', 'Hello, World']
```

**Configuration**

For the configuration, a configuration file is created for the client and engine when the IPython hub is started, usually in ~/.ipython/profile_default/security/.

1. If we don't want to use the default profile, we should first create a new IPython profile with:

```bash
$ pipenv run ipython profile create --parallel --profile=local
[ProfileCreate] Generating default config file: '/Users/veit/.ipython/profile_parallel/__ipython_config.py'
[ProfileCreate] Generating default config file: '/Users/veit/.ipython/profile_parallel/__ipython_kernel_config.py'
[ProfileCreate] Generating default config file: '/Users/veit/.ipython/profile_parallel/__ipcontroller_config.py'
```

(continues on next page)
--parallel includes the configuration files for *Parallel Computing* (ipengine, ipcontroller etc.).

1. When the IPython controller and the engines are started, the files ipcontroller-engine.json and ipcontroller-client.json are generated in ~/.ipython/profile_default/security/.

**ipcluster in mpiexec/mpirun mode**

1. Creating the profile:

   ```
   $ pipenv run ipython profile create --parallel --profile=mpi
   [ProfileCreate] Generating default config file: '/Users/veit/.ipython/profile_mpi/ipython_config.py'
   [ProfileCreate] Generating default config file: '/Users/veit/.ipython/profile_mpi/ipython_kernel_config.py'
   [ProfileCreate] Generating default config file: '/Users/veit/.ipython/profile_mpi/ipcontroller_config.py'
   [ProfileCreate] Generating default config file: '/Users/veit/.ipython/profile_mpi/ipengine_config.py'
   ```

   2. Editing of ipcluster_config.py:

   1. so that the MPI launcher can be used:

   ```
   c.IPClusterEngines.engine_launcher_class = 'MPIEngineSetLauncher'
   ```

   3. The cluster can then be started with:

   ```
   $ pipenv run ipcluster start -n 4 --profile=mpi
   [IPClusterStart] Starting ipcluster with [daemon=False]
   [IPClusterStart] Starting Controller with LocalControllerLauncher
   [IPClusterStart] Starting 4 Engines with LocalEngineSetLauncher
   [IPClusterStart] Engines appear to have started successfully
   ```

**IPython's Direct interface**

**Create a DirectView**

1. ```
   import ipyparallel as ipp
   rc = ipp.Client()
   ```

2. ```
   rc = ipp.Client(profile='default')
   ```

3. ```
   rc.ids
   ```
Use all engines:

```
[4]: dview = rc[:]
```

### map() function

Python’s builtin `map()` function can be applied to a sequence of elements and is usually easy to parallelise. Please note that the `DirectView` version of `map()` does not do automatic load balancing. You may have to use `LoadBalancedView` for this.

```
[5]: serial_result = list(map(lambda x:x**10, range(32)))
[6]: parallel_result = dview.map_sync(lambda x: x**10, range(32))
[7]: serial_result == parallel_result
[7]: True
```

### ipyparallel magics

```
[1]: import ipyparallel as ipp
    rc = ipp.Client()
[2]: with rc[:].sync_imports():
    import numpy

    importing numpy on engine(s)
    got unknown result: 05d7cec8-912387fc506b2f40c1471286
    got unknown result: fa6fddcf-779a7f98c12df80a46c1e51f

[3]: %px a = numpy.random.rand(2,2)
[4]: %px numpy.linalg.eigvals(a)
    Out[0:2]: array([0.65462266, 1.13932216])
    Out[1:2]: array([-0.83473139, -0.29322661])
    Out[2:2]: array([0.05028953, 1.30003731])
    Out[3:2]: array([0.99612347, 0.42574895])
[5]: %px print('hi')
    [stdout:0] hi
    [stdout:1] hi
    [stdout:2] hi
    [stdout:3] hi
```
%px %pylab inline

[stdout:0] Populating the interactive namespace from numpy and matplotlib
[stdout:1] Populating the interactive namespace from numpy and matplotlib
[stdout:2] Populating the interactive namespace from numpy and matplotlib
[stdout:3] Populating the interactive namespace from numpy and matplotlib

%px plot(rand(100))

[output:0]

[output:1]

[output:2]
%%px cell magic

--targets, --block and --noblock

[8]: %%px --targets ::2
print('I am even')

[stdout:0] I am even
[stdout:2] I am even

[9]: %%px --targets 1
print('I am number 1')

7.2. Performance measurement and optimisation
I am number 1

```python
[10]: %%px
    print('still all by default')
    [stdout:0] still all by default
    [stdout:1] still all by default
    [stdout:2] still all by default
    [stdout:3] still all by default

[11]: %%px --noblock
    import time
    time.sleep(1)
    time.time()

[11]: <AsyncResult: execute>

[12]: %%pxresult
    Out[0:8]: 1570269717.543805
    Out[1:8]: 1570269717.54598
    Out[2:8]: 1570269717.5485692
    Out[3:7]: 1570269717.548548

[13]: %%px --block --group-outputs=engine
    import numpy as np
    A = np.random.random((2,2))
    ev = numpy.linalg.eigvals(A)
    print(ev)
    ev.max()
    [stdout:0] [-0.68780518  0.74951568]
    [output:0]
    Out[0:9]: 0.7495156792444352
    [stdout:1] [0.75691299 1.08526474]
    [output:1]
    Out[1:9]: 1.0852647415435284
    [stdout:2] [-0.2153414  1.46736661]
    [output:2]
    Out[2:9]: 1.467366609797983
    [stdout:3] [0.55698336  0.13540831]
    [output:3]
    Out[3:8]: 0.5569833627025539
```

Chapter 7. Refactoring
%pxresult

[14]: dview = rc[:]

[15]: dview.block = False
%px print('hi')
%pxresult

[stdout:0] hi
[stdout:1] hi
[stdout:2] hi
[stdout:3] hi
[stdout:0] hi
[stdout:1] hi
[stdout:2] hi
[stdout:3] hi

%autopx

[16]: dview.block = True

[17]: %autopx

%autopx enabled

[18]: max_evals = []
for i in range(100):
    a = numpy.random.rand(10,10)
    a = a+a.transpose()
    evals = numpy.linalg.eigvals(a)
    max_evals.append(evals[0].real)

[19]: print("Average max eigenvalue is: \%f" % (sum(max_evals)/len(max_evals)))

[stdout:0] Average max eigenvalue is: 10.209420
[stdout:1] Average max eigenvalue is: 10.214184
[stdout:2] Average max eigenvalue is: 10.071618
[stdout:3] Average max eigenvalue is: 10.132674

%pxconfig

[3]: %pxconfig --block
%px print('hi')

[stdout:0] hi
[stdout:1] hi
[stdout:2] hi
[stdout:3] hi

7.2. Performance measurement and optimisation
Multiple active views

Magics of ipyparallel are assigned to certain DirectView objects. However, the active view can be changed by calling the activate() method on a view.

```python
[3]: even = rc[::2]
even.activate()
%px print('hi')
[3]: <AsyncResult: execute>

[4]: even.block = True
%px print('hi')

got unknown result: 98a3ad5c-00fa7f40e334c8a4186d5be5
[stdout:0] hi
[stdout:2] hi
```

If you activate the view, you can also specify a suffix so that it can be assigned to a whole range of magics without replacing the existing ones.

```python
[5]: rc.activate()
[5]: <DirectView all>

[6]: even.activate('_even')
%px print('hi')

[stdout:0] hi
[stdout:1] hi
[stdout:2] hi
[stdout:3] hi

[7]: %px_even print("We aren't odd!")

[stdout:0] We aren't odd!
[stdout:2] We aren't odd!
```
This suffix is used at the end of all magics, e.g. %autopx_even, %pxresult_even etc.

For the sake of simplicity, also Client has an activate() method that creates a DirectView with block = True, activates it, and returns the new view.

The initial magics that are registered when a client is created are the result of the call rc.activate() with standard arguments.

**Engines as kernel**

Engines are actually the same object as IPython kernels, with the only exception that engines connect to a controller, while regular kernels bind their sockets directly to connections to their front end.

Sometimes you will connect your front end directly to an engine for debugging or analysing the interaction more directly. You can also do this by instructing the engine to bind its kernel to your frontend as well:

```
[ ]: %px import ipyparallel as ipp; ipp.bind_kernel()
%px %qtconsole
```

**Note:** Be careful with this statement, as it starts as many QtConsoles as there are engines available.

Alternatively, you can also display the connection information and determine how you can establish a connection to the engines, depending on where they live and where you are:

```
[ ]: %px %connect_info
```

**Task interface**

The task interface to the cluster presents the engines as a fault-tolerant, dynamic load balancing of Workers. In contrast to the direct interface, the task interface does not have direct access to individual engines. As the IPython scheduler assigns the workers, the interface becomes simpler and more powerful at the same time.

The best part, however, is that both interfaces can be used at the same time to leverage their respective strengths. If calculations do not depend on previous results, the task interface is ideal:

**Create an LoadBalancedView instance**

```
[1]: import ipyparallel as ipp

[2]: rc = ipp.Client()

[3]: rc = ipp.Client(url_file='~/Users/veit/.ipython/profile_mpi/security/ipcontroller-client.json')

[4]: rc = ipp.Client(profile='mpi')

[5]: lview = rc.load_balanced_view()
```

load_balanced_view is the default view.

See also: Views.
Fast and easy parallelism

map() - LoadBalancedView

```python
[6]: lview.block = True
    serial_result = map(lambda x: x**10, range(32))
    parallel_result = lview.map(lambda x: x**10, range(32))
    serial_result == parallel_result

[6]: True
```

@lview.parallel() decorator

```python
[7]: @lview.parallel()
    def f(x):
        return 10.0*x**4

    f.map(range(32))

[7]: [0.0, 10.0, 160.0, ...]
```

Dependencies

Note: Please note that the pure ZeroMQ scheduler does not support any dependencies.

Function dependencies

UnmetDependency

@ipp.require decorator

@ipp.depend decorator

dependent object

Dependency

```python
[ ]: client.block = False

    ar = lview.apply(f, args, kwargs)
    ar2 = lview.apply(f2)

    with lview.temp_flags(after=[ar, ar2]):
        ar3 = lview.apply(f3)

    with lview.temp_flags(follow=[ar], timeout=2.5)
        ar4 = lview.apply(f3)
```
See also: Some parallel workloads can be described as Directed acyclic graph (DAG). In DAG Dependencies we describe using an example how NetworkX is used to represent the task dependencies as DAG.

ImpossibleDependency

retries and resubmit

Schedulers

[: ]: ipcontroller --scheme=lr

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lru</td>
<td><em>Least Recently Used</em>: Always assigns the workers to the last used engine. Similar to round robin, however, it does not take into account the runtime of each individual task.</td>
</tr>
<tr>
<td>plainrand</td>
<td><em>Plain Random</em>: Randomly selects the engine to be run.</td>
</tr>
<tr>
<td>twobin</td>
<td><em>Two-Bin Random</em>: Requires numpy. Randomly select two engines and use lru. This is often better than the purely random distribution, but requires more computational effort.</td>
</tr>
<tr>
<td>leastload</td>
<td><em>Least Load</em>: Standard scheme that the engine always assigns tasks with the fewest outstanding tasks.</td>
</tr>
<tr>
<td>weighted</td>
<td><em>Weighted Two-Bin Random</em>: Weighted <em>Two-Bin Random</em> scheme.</td>
</tr>
</tbody>
</table>

AsyncResult object

apply() returns in the noblock mode an AsyncResult object. This allows inquiries with the get() method at a later point in time. In addition, metadata occurring during execution is also collected in this object.

The AsyncResult object provides a number of useful functions for parallelisation that can be accessed through Python's multiprocessing.pool.AsyncResult:

get_dict

AsyncResult.get_dict()

[1]: import os
    import ipyparallel as ipp
    rc = ipp.Client()
    ar = rc[:].apply_async(os.getpid)
    pids = ar.get_dict()
    rc[:]["pid_map"] = pids
from __future__ import print_function

import time

import ipyparallel as ipp

# create client & view
rc = ipp.Client()
dv = rc[:]
v = rc.load_balanced_view()

dv.scatter('id', rc.ids, flatten=True)
print("Engine IDs: ", dv['id'])

# create a Reference to 'id'. This will be a different value on each engine
ref = ipp.Reference('id')
print("sleeping for 'id' seconds on each engine")
tic = time.time()
ar = dv.apply(time.sleep, ref)
for i,r in enumerate(ar):
    print("%i: %.3f"%(i, time.time()-tic))

def sleep_here(t):
    import time
    time.sleep(t)
    return id,t

# one call per task
print("running with one call per task")
amr = v.map(sleep_here, [.01*t for t in range(100)])
tic = time.time()
for i,r in enumerate(amr):
    print("task %i on engine %i: %.3f" % (i, r[0], time.time()-tic))

print("running with four calls per task")
# with chunksize, we can have four calls per task
amr = v.map(sleep_here, [.01*t for t in range(100)], chunksize=4)
tic = time.time()
for i,r in enumerate(amr):
    print("task %i on engine %i: %.3f" % (i, r[0], time.time()-tic))

print("running with two calls per task, with unordered results")

(continues on next page)
# We can even iterate through faster results first, with ordered=False

```python
amr = v.map(sleep_here, [.01*t for t in range(100,0,-1)], ordered=False, chunksize=2)
tic = time.time()
for i,r in enumerate(amr):
  print("slept %.2fs on engine %i: %.3f" % (r[1], r[0], time.time()-tic))
```

Engine IDs: [0, 1, 2]
sleeping for 'id' seconds on each engine
0: 0.027
1: 1.022
2: 2.024
running with one call per task
task 0 on engine 2: 0.000
task 1 on engine 1: 0.001
task 2 on engine 0: 0.001
task 3 on engine 2: 0.001
task 4 on engine 1: 0.001
...
slept 0.12s on engine 2: 16.868
slept 0.11s on engine 2: 16.868
slept 0.14s on engine 0: 16.873
slept 0.13s on engine 0: 16.873
slept 0.16s on engine 1: 16.893
...

```python
[4]: from functools import reduce
    from math import sqrt
    import numpy as np

    X = np.linspace(0,100)
    add = lambda a,b: a+b
    sq = lambda x: x*x
    sqrt(reduce(add, map(sq, X)) / len(X))

[4]: 58.028845747399714
```

1. map(sq, X) computes the square of each item in the list.
2. reduce(add, sqX) / len(X) calculates the mean by adding the list of AsyncMapResult and dividing by the number.
3. Square root of the resulting number.

**See also:** If you want to expand the results of AsyncResult or AsyncMapResult you can do so with the msg_ids attribute. You can find an example for this at ipyparallel/examples/customresults.py.
MPI

Often, a parallel algorithm requires moving data between the engines. One way is to push and pull over the DirectView. However, this is slow because all of the data has to get through the controller to the client and then back to the final destination.

A much better option is to use the Message Passing Interface (MPI). IPython’s parallel computing architecture was designed from the ground up to integrate with MPI. This notebook gives a brief introduction to using MPI with IPython.

Requirements

- A standard MPI implementation like OpenMPI or MPICH.
  For Debian/Ubuntu these can be installed with
  
  ```
  $ sudo apt install openmpi-bin
  
  or
  
  $ sudo apt install mpich
  
  Alternatively, OpenMPI or MPICH can also be installed with Spack: the packages are openmpi or mpich.
- mpi4py

Starting the engines with activated MPI

Automatic start with mpiexec and ipcluster

This can be done with, e.g.

```
$ pipenv run ipcluster start -n 4 --profile=mpi
```

For this, however, a corresponding profile must first be created; see configuration.

Automatic start with PBS and ipcluster

The ipcluster command also offers integration in PBS. You can find more information about this in Using ipcluster in PBS mode.

Example

The following notebook cell calls psum.py with the following content:

```python
from mpi4py import MPI
import numpy as np

def psum(a):
    locsum = np.sum(a)
    rcvBuf = np.array(0.0, 'd')
    MPI.COMM_WORLD.Allreduce([locsum, MPI.DOUBLE],
```
import ipyparallel as ipp

c = ipp.Client(profile='mpi')
view = c[:]
view.activate()
view.run('psum.py')
view.scatter('a', np.arange(16, dtype='float'))
view['a']

[1]: [array([0., 1., 2., 3.]),
     array([4., 5., 6., 7.]),
     array([8., 9., 10., 11.]),
     array([12., 13., 14., 15.])]

[2]: %px totalsum = psum(a)
[2]: Parallel execution on engines: [0,1,2,3]

[3]: view['totalsum']
[3]: [120.0, 120.0, 120.0, 120.0]

See also:

- Faster CPython
- Python Speed Center
- Tracing the Python GIL

See also:

- Refactoring by Martin Fowler
Non-reproducible single occurrences are of no significance to science."\(^1\)

In order for others to be able to use your code, it should meet some conditions:

- You should not silently rely on specific resources and environments
- Required software packages and hardware should be specified in the requirements
- Path information will only work in a different context within your package or in previously generated directories and files
- Do not share secrets like login details or internal IP numbers in your published product

There are various tools that support you in creating shareable products. These can be tools on the one hand for the versioning of the source code and the training data as well as for the reproducibility of the execution environments, on the other hand for testing, logging, documenting and creating packages.

**See also:**

- Dustin Boswell, Trevor Foucher: The Art of Readable Code
- TIB workshop «FAIR Data and Software»
  - GitHub Page
  - GitHub Repository
  - Slides
- Dryad: Best practices for creating reusable data publications

### 8.1 Manage code with Git

In order to have better control over your source code, it is usually managed with Git. However, I will only briefly explain the most important terms and provide brief *installation and configuration instructions*. General introductions to Git are sufficiently available elsewhere.

**See also:**

- Git Cheat Sheet (PDF)
- Interactive Git Cheatsheet
- Software Carpentry Version Control with Git
- Flight rules for Git

\(^1\) Karl Popper in *The Logic of Scientific Discovery*, 1959
Essentially, in this tutorial, I show on the one hand how Jupyter Notebooks can be managed with Git, and on the other hand best practices and typical Git workflows.

8.1.1 Git installation and configuration

Installation

For iX distributions, Git should be in the standard repository.

- For Debian/Ubuntu:
  
  $ sudo apt install git-all

  The bash autocompletion makes Git easier to use on the command line:

  $ sudo apt install bash-completion

- For macOS:
  
  There are several different ways to install Git on a Mac. Probably the easiest way to do is to install the Xcode Command Line Tools. For this you only have to call up git in the terminal for the first time:

  $ git --version

  git-completion you can install with Homebrew:

  Then you have to add the following line in ~/.bash_profile:

  ```bash
  [[ -r "$(brew --prefix)/etc/profile.d/bash_completion.sh" ]] && "$(brew --prefix)/etc/profile.d/bash_completion.sh"
  ```

- For Windows:
  
  You can simply go to https://git-scm.com/download/win to start the download automatically. Further information can be found at https://gitforwindows.org/.

Configuration

$ git config --global user.name "[name]" defines the name associated with your commit transactions.

$ git config --global user.email "[email address]" defines the email that will be linked to your commit transactions.

$ git config --global color.ui auto activates the coloring of the command line output.
The ~/.gitconfig file

For example, the following file can be created with the commands given above:

```
[user]
  name = veit
  email = veit@cusy.io

[color]
  diff = auto
  status = auto
  branch = auto
```

However, aliases can also be specified in the ~/.gitconfig file:

```
[alias]
  st = status
  ci = commit
  br = branch
  co = checkout
  df = diff
  dfs = diff --staged
```

The editor can also be specified and space errors can be highlighted in `git diff`:

```
[core]

  editor = vim

  # Highlight whitespace errors in git diff:
  whitespace = tabwidth=4,tab-in-indent,cr-at-eol,trailing-space
```

Manage login data

Since Git version 1.7.9, the access data to git repositories can be managed with `gitcredentials`. To use this, you can, for example, specify the following:

```
$ git config --global credential.helper Cache
```

This will keep your password in the cache for 15 minutes. The timeout can be increased if necessary, e.g. with:

```
$ git config credential.helper 'cache --timeout=3600'
```
macOS

With macOS you can use `osxkeychain` to store the login information. `osxkeychain` requires Git version 1.7.10 or newer and can be installed in the same directory as Git with:

```bash
$ git credential-osxkeychain
git: 'credential-osxkeychain' is not a git command. See 'git --help'.
$ curl -s -O http://github-media-downloads.s3.amazonaws.com/osx/git-credential-osxkeychain
$ chmod u+x git-credential-osxkeychain
$ sudo mv git-credential-osxkeychain /usr/bin/
Password:
git config --global credential.helper osxkeychain
```

This enters the following in the `~/.gitconfig` file:

```ini
[credential]
    helper = osxkeychain
```

Windows

For Windows Git Credential Manager for Windows is available. First the Installer must be downloaded for the program. After double-clicking, it will guide you through the rest of the installation. As a terminal emulator for Git, you should select the standard Windows console window.

**Note:** You can find a comprehensive example of a `~/.gitconfig` file in my dotfiles repository: `.gitconfig`.

The `.gitignore` file

In the `.gitignore` file you can exclude files from version management. A typical `.gitignore` file can look like this:

```ini
/logs/*
!logs/.gitkeep
/tmp
*.swp
```

**Git-commit empty folder**

In the example above you can see that with `/logs/*` no content of the logs directory should be versioned with Git, but an exception is defined in the following line: `!logs/.gitkeep` allows the file `.gitkeep` to be managed with Git. The logs directory is then also transferred to the Git repository. This construction is necessary because empty folders cannot be managed with Git.

Another possibility is to create a `.gitignore` file in an empty folder with the following content:

```ini
# ignore everything except .gitignore
*
!.gitignore
```
excludesfile

However, you can also exclude files centrally for all Git repositories. For this purpose, you can set `excludesfile` in the `~/.gitconfig` file:

```
[cORE]

    # Use custom ".gitignore"
    excludesfile = ~/.gitignore
...
```

Note: You can find helpful templates in my dotfiles repository or on the gitignore.io website.

8.1.2 Git pre-commit hooks

`pre-commit` is a framework for managing and maintaining multilingual pre-commit hooks.

An essential task is to make the same scripts available to the entire development team. Yelp’s `pre-commit` manages such pre-commit hooks and distributes them to various projects and developers.

Git pre-commit hooks are mostly used to automatically point out problems in the code before code reviews, e.g. to check the formatting or to find debug instructions. Pre-commit simplifies the cross-project sharing of the pre-commit hook. The language in which a linter was written, for example, is also abstracted away – `scss-lint` is written in Ruby, but you can use it with pre-commit without having to add a gem file to your project.

Installation

Before you can hook the pre-commit package manager must be installed.

... on macOS:

```
$ brew install pre-commit
```

... in your Python project:

```
$ pipenv install pre-commit
```

Check the installation with, for example

```
$ pipenv run pre-commit -V
pre-commit 2.6.0
```
Configuration

After pre-commit has been installed, plugins for this project can be configured with the `.pre-commit-config.yaml` file in the root directory of your project.

```
repos:
- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: v2.5.0
  hooks:
    - id: check-yaml
    - id: end-of-file-fixer
    - id: trailing-whitespace

- repo: https://github.com/psf/black
  rev: 19.10b0
  hooks:
    - id: black
```

You can also have this file generated with

```
$ pipenv run pre-commit sample-config
# See https://pre-commit.com for more information
# See https://pre-commit.com/hooks.html for more hooks
repos:
- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: v2.4.0
  hooks:
    - id: trailing-whitespace
    - id: end-of-file-fixer
    - id: check-yaml
    - id: check-added-large-files
```

If you want to run this pre-commit hook before every commit, install it with `pre-commit install`. If the hooks are to be executed manually, this can be done with `pre-commit run --all-files`. Single hooks can then also be carried out separately, for example `pre-commit run trailing-whitespace`.

The first time a pre-commit hook is called, it is first downloaded and then installed. This can take some time, e.g. if a copy of node has to be made.

```
$ pipenv run pre-commit run --all-files
Trim Trailing Whitespace.................................................Passed
Fix End of Files.........................................................Passed
Check Yaml...............................................................Passed
Check for added large files..............................................Passed
black.................................................................Passed
```

A full list of configuration options can be found in Adding pre-commit plugins to your project.

You can also write your own hooks, see Creating new hooks.

You can also update the hooks automatically with:

```
$ pipenv run pre-commit autoupdate
```

Further options can be found in `pre-commit autoupdate [options]`.
Install the Git-Hook scripts

The scripts are installed in our project so that pre-commit is reliably executed before each commit:

```
$ pre-commit install
pre-commit installed at .git/hooks/pre-commit
```

Use in CI

Pre-commit can also be used for continuous integration.

Example of GitHub Actions

```yaml
- name: set PY
  run: echo "::set-env name=PY::$(python -VV | sha256sum | cut -d' ' -f1)"
- uses: actions/cache@v1
  with:
    path: ~/.cache/pre-commit
    key: pre-commit|${{ env.PY }}|${{ hashFiles('.pre-commit-config.yaml') }}
```

See also:

pre-commit/action

Example for GitLab Actions

```yaml
my_job:
  variables:
    PRE_COMMIT_HOME: ${CI_PROJECT_DIR}/.cache/pre-commit
  cache:
    paths:
      - ${PRE_COMMIT_HOME}
```

See also:

For more information on fine-tuning caching, see Good caching practices.

8.1.3 Configuring Git for Jupyter Notebooks

The results of the calculations can also be saved in the notebook file format nbformat. These can also be Base-64-coded blobs for images and other binary data that should not normally be included in a version management. These can be removed manually with Cell → All Output → Clear, but you have to carry out these steps before every `git add`, and it also does not solve a second cause of the noise in `git diff`, namely some in the metadata.

In order to get systematically comparable versions of notebooks in the version management, we can use `jq`, a lightweight JSON processor. It takes some time to set up `jq` because it has its own query/filter language, but the default settings are usually well chosen.
**Installation**

jq for Debian/Ubuntu can be installed with:

```
$ sudo apt install jq
```

or for macOS with:

```
$ brew install jq
```

**Example**

A typical call is:

```
jq --indent 1 "
  (.cells | select (.metadata) = []
  | (.cells | select (.metadata) = null
  | .metadata | {language_info: {name: "python", "pygments_lexer": "ipython3"}}
  | .Cells[].metadata = {} 
  | example.ipynb
```

Each line within the single quotation marks defines a filter – the first selects all entries from the cells list and deletes the output. The next entry resets all outputs. The third step deletes the notebook’s metadata and replaces it with a minimum of necessary information so that the notebook can still be run without complaints. The fourth filter line .cells[].metadata = {}, deletes all meta information. If you want to keep certain meta information, you can indicate this here.

**Set up**

1. To make your work easier, you can create an alias in the `~/.bashrc` file:

   ```
   alias nbstrip_jq="jq --indent 1 "
   
   | .cells[] | select (.metadata) = []
   | (.cells | select (.metadata) = null
   | .metadata | {language_info: {name: "python", "pygments_lexer": "ipython3"}}
   | .Cells[].metadata = {} 
   | example.ipynb"
   ```

2. Then you can conveniently enter the following in the terminal:

   ```
   $ nbstrip_jq example.ipynb > stripped.ipynb
   ```

3. If you start with an existing notebook, you should first add a filter commit by simply reading in the newly filtered version of your notebook without the unwanted metadata. After you have added the notebook with `git add`, you can see whether the filter has really worked with `git diff --cached` before you do `git commit -m 'filter'`.

4. If you want to use this filter for all Git repositories, you can also configure your Git globally:

   1. First you add the following to your `~/.gitconfig` file:
2. Then you have to specify the following in the `~/.gitattributes` file:

```
*.ipynb filter=nbstrip_jq
```

**Warning:** If you want to do `git rebase`, you should deactivate the line beforehand.

5. However, the problem remains that `git status` show changes to files when the cells of a notebook have been executed, even though `git diff` still shows no changes. Therefore the following should be entered in the `~/.bashrc` file in order to quickly clean the respective working directory:

```
function nbstrip_all_cwd {
    for nbfile in *.ipynb; do
        echo "$(nbstrip_jq $nbfile)" > $nbfile
    done
    unset nbfile
}
```

### 8.1.4 Git tools for notebooks

**nbdime**

*nbdime* is a GUI for diffs of `nbformat` and replaces `nbdiff`. It tries content-aware differencing as well as the merging of notebooks, is not limited to the display of diffs, but also prevents unnecessary changes from being checked in.

**nbstripout**

*nbstripout* automates *Clear all outputs*. It uses `nbformat` and a few auto magic to set up `.git config`. In my opinion, however, it has two drawbacks: it is limited to the problematic metadata section, and it is slow.
8.1.5 Workspaces

- **git add** adds files from the working directory to the staging area.
- **git reset HEAD** restores a file in the work area from the stage area.
- **git stash** moves files from the workspace to a stash.
- **git stash pop** brings files from the stash to the work area.
- **git push** moves files from the staging area to the repository.
  
  `git push -u origin master` -u defines the upstream reference for each branch, the arguments are then given and no longer have to be specified explicitly. In our example `master` in the external repository is referenced.

8.1.6 Working with Git

Start working on a project

Start your own project

$ git init [my_project] creates a new, local git repository

[my_project] if the project name is given, Git creates a new directory and initializes it

If no project name is given, the current directory is initialised
Work on a project

$ git clone [project_url] downloads a project with all branches and the entire history from the remote repository

    --depth
    -b specifies the name of the remote branch to be downloaded

Work on a project

$ git status shows the status of the current branch in the working directory with new, changed and files already marked for commit.

$ git add [file] adds a file to the stage area.

$ git add -p [file] adds parts of a file to the stage area.

$ git add -e [file] the changes to be adopted can be edited in the standard editor.

$ git diff [file] shows differences between work and stage areas.

$ git diff --staged [file] shows differences between the stage area and the repository.

$ git diff --word-diff shows the changed words.

$ git checkout -- [file] irrevocably discard changes in the work area.

$ git commit -m 'Commit message' make a new commit with the added changes.

    --dry-run --short

    --dry-run shows what would be committed.

    --short shows the status in short format.

$ git reset [file] return to the current file from the stage area.

$ git rm [file] remove a file from the work and stage areas.

$ git stash move the current changes from the work area to the stash.

$ git stash pop transfer the changes from the hiding place to the work area and empty the hiding place.

$ git stash drop emptying a specific stash.

8.1.7 Git branches

$ git branch [-a] shows all local branches in a repository.

    -a also shows all removed branches.

$ git branch [branch_name] creates a new branch based on the current HEAD.

$ git checkout [-b] [branch_name] changes the working directory to the specified branch.

    -b creates the specified branch if it does not already exist.

$ git merge [from name] connects the given branch with the current branch you are currently in, e.g.
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast forward
  setup.py | 1 -
  1 files changed, 0 insertions(+), 1 deletions(-)

**Fast forward** means that the new commit immediately followed the original commit and so the branch pointer only had to be continued.

In other cases the output can look like this:

$ git checkout master
$ git merge #42
Merge made by recursive.
  setup.py | 1 +
  1 files changed, 1 insertions(+), 0 deletions(-)

**recursive** is a merge strategy that is used when the merge is only to be done to **HEAD**.

**Merge conflict**

Occasionally, however, Git runs into issues with merging, such as:

$ git merge #17
Auto-merging setup.py
CONFLICT (content): Merge conflict in setup.py
Automatic merge failed; fix conflicts and then commit the result.

**See also:**
- Git Branching - Basic Branching and Merging
- Git Tools - Advanced Merging

**Branches**

$ git branch -d [name] deletes the selected branch if it has already been transferred to another.

-D instead of -d forcing the deletion.

**See also:**
- Git Branching - Branches in a Nutshell
8.1.8 Git log

$ git log [\-n count] list the commit history of the current branch.
\-n limits the number of commits to the specified number.

$ git log [\(--\text{after}=\text{"YYYY-MM-DD"}\) \[\(--\text{before}=\text{"YYYY-MM-DD"}\)] Commit history filtered by date.
Relative information such as 1 week ago or yesterday is also permitted.

$ git log \--author = "name" filters the commit history by authors.
You can also search for several authors at the same time, e.g.
$ git log \--author="veit\|vsc"

$ git log \--grep = "term" filters the commit history for regular expressions in the commit message.

See also:
• Regular expressions <../../workspace/ipython/unix-shell/regex.ipynb> Regular expressions

$ git log \-S"foo" filters commits according to certain lines in the source code.

$ git log \-G"ba*" filters commits based on regular expressions in the source code.

$ git log \- path/to/foo.py filters the commit history for specific files.

$ git log \text{master..feature} filters for different commits in different branches, in our case between the branches master and feature.

$ git log \--oneline \--graph \--decorate Show the history diagram with references, one commit per line.

$ git log \text{ref..} List commits that exist in the current branch and are not merged into \text{ref}. \text{ref} can be the name of a branch or a tag.

$ git log \..\text{ref} List commits that exist in \text{ref} and are not merged with the current branch.

$ git reflog List operations (e.g. checkout, commit) that have been performed in the local repository.

8.1.9 Git tagging

$ git tag list all tags, e.g.

```
$ git tag
0.1.0
0.2.0
0.3.0
0.3.1
0.4.0
0.4.1
0.5.0
0.6.0
0.6.1
```

$ git tag \-l regex only lists tags that match a regular expression.

$ git tag \[name\] \[commit sha\] creates a tag with the name \text{name} for the current commit.
With sha the specific commit gets a tag, not the current one.

$ git tag \-a \[name\] \[commit sha\] \[\-m 'Commit message'\] creates a tag with the name \text{name} for the current commit, e.g.:
$ git tag -a 0.6.1 -m '0.6.1 release'
$ git push origin 0.6.1

Counting objects: 1, done.
Writing objects: 100% (1/1), 161 bytes, done.
Total 1 (delta 0), reused 0 (delta 0)
To https://github.com/veit/jupyter-tutorial.git
 * [new tag] 0.6.1 -> 0.6.1

8.1.10 Undo changes

$ git reset [--hard] [target reference] switches from the current branch to the target reference and leaves the difference as an uncommitted change, e.g.

$ git reset HEAD setup.py

--hard discards all changes.

$ git revert [commit sha] creates a new commit and undoes the changes to the specified commit. The changes are inverted.

$ git fetch [remote] accepts changes from remote but does not update branches.

$ git fetch --prune [remote] Remote refs are removed when they are removed from the remote repository.

$ git commit --amend updates and replaces the most recent commit with a new commit that combines any staged changes with the contents of the previous commit. With nothing currently staged, this just rewrites the previous commit message.

$ git checkout [file] alters files in the working directory to a state previously known to Git. By default, git will checkout HEAD, the last commit on the currently checked-out branch. Alternativly you could also choose a specific branch or SHA.

$ git pull [remote] pulls changes from the remote repository and merges the current branch with the upstream.

$ git push [--tags] [remote] transfers local changes to remote.

With --tags tags can be transmitted at the same time.

$ git push -u [remote] [branch] transfers the local branch to the remote repository with the copy set as upstream.

8.1.11 Git best practices

- Commit early!

Make your first commit after you’ve finished the initial installation and before you make your first changes. For a cookiecutter template, for example, following the following steps:


→ git
full_name [Veit Schiele]:

(continues on next page)
If no .gitignore file is present in your project, you should create one and at least -exclude .ipynb_checkpoints and */.ipynb_checkpoints/*. If you have accidentally checked the corresponding files into your Git repository, you can remove them again with:

$ git rm -r .ipynb_checkpoints/

You can get an overview of other .gitignore entries either in the dotfiles repository or on the gitignore.io website.

These initial changes can then be checked in with:

$ cd cusy.example
$ git init
$ git add *
$ git add .gitignore
$ git commit -m 'Initial commit'
$ git remote add origin ssh://git@github.com:veit/cusy.example.git
$ git push -u origin master

Each repository should also have a README.rst file that describes the deployment and the basic structure of the code.

• Commit often!

This makes it easier for you
- to isolate errors
- to understand the code
- to maintain the code in the future

If you have made several changes to a file, you can split them up into several commits later with:

$ git add -p my-changed-file.py

• Don’t change the published history!

Even if you later find out that a commit that has already been published with git push contains one or more errors, you should never try to undo this commit. Rather, you should fix the error that have occurred through further commits.

• Choose a Git workflow!

Choose a workflow that fits best to your project. Projects are by no means identical and a workflow that fits one project does not necessarily have to fit in another project. A different workflow can be recommended initially than in the further progress of the project.

• Make meaningful commits!

By creating insightful and descriptive commit messages, you make working in a team a lot easier. They allow others to understand your changes. They are also helpful at a later point in time to understand which goal should be achieved with the code.
Usually short messages, 50–72 characters long, should be specified and displayed on one line, eg with `git log --oneline`.

With `git blame` you can later specify for each line in which revision and by which author the change was made. You can find more information on this in the Git documentation: `git-blame`.

GitLab also interprets certain commit messages as links, e.g.

```
$ git commit -m "Awesome commit message (Fixes #21 and Closes group/otherproject#22)"
```

- links to issues: #123
  - also for issues in other projects: othergroup/otherproject#123
- links to merge requests: !123
- links to snippets: $123

There should be at least one ticket for each commit that should provide more detailed information about the changes.

You can find more good information in A Note About Git Commit Messages.

• Maintain your repository regularly!

You should perform the following maintenance work regularly:

- Validate the repo with `git fsck`.
- Compresses the repo with `git gc` or `git gc --aggressive`.

  **See also:**

  - `git gc`
  - Git Internals - Maintenance and Data Recovery
  - Clean up the remote tracking branches with `git remote update --prune`.
  - Checks forgotten work with `git stash list`.

• Check your repositories regularly for unwanted files!

With Gitleaks you can regularly check your repositories for unintentionally saved access data.

With Git Filter-Branch, BFG Repo-Cleaner or git-filter-repo you can remove unwanted files, be it access data or large binary files, from your Git history.

Alternatively, you can also delete the data on the command line.

- Delete the last commit

  ```
  $ git reset HEAD^ --hard
  $ git push origin -f
  ```

- Delete other commits

  ```
  $ git rebase -i sha origin
  ```

  `-i Interactive mode, in which your standard editor is opened and a list of all commits after the commit with the hash value sha to be removed is displayed, e.g.```
If you now remove a line, this commit will be deleted after saving and closing the editor. Then the remote repository can be updated with:

```bash
$ git push origin HEAD:master -f
```

– Modifying a commit message

This can also be easily with rebase by not deleting the line in your editor but replace pick with r (reword).

– Remove a file from the history

A file can be completely removed from the current branch’s Git history with:

```bash
$ git filter-branch --force --index-filter 'git rm -rf --cached --ignore-unmatch path/somefile' --prune-empty --tag-name-filter cat -- --all
$ rm -rf .git/refs/original/
$ git reflog expire --expire=now --all
$ git gc --prune=now
$ git gc --aggressive --prune=now
$ git push origin <branch> --force
```

– Removing a string from the history

```bash
$ git filter-branch --force --tree-filter "[ -f <path>/<file> ] && \
     sed -i 's/<secret password>//g' <path>/<file> || /bin/true" -- --all
```

Note: On macOS /usr/bin/true must be used instead of /bin/true on Linux.

See also:

– git-reflog
– git-gc

### 8.1.12 Git workflows

Version management with git makes branching and merging much easier than older versioning systems like SVN. This enables a variety of branching strategies and workflows.
Git flow and its problems

- Feature branches
- develop
- release branches
- hotfixes
- master

Time

- Feature for future release
- Major feature for next release
- Incorporate bugfix in develop
- Severe bug fixed for production: hotfix 0.2
- Start of release branch for 1.0
- Only bugfixes!
- Bugfixes from rel. branch may be continuously merged back into develop

From this point on, "next release" means the release after 1.0

Tags:
- 0.1
- 0.2
- 1.0
Git Flow was one of the earliest suggestions for using Git branches. It recommended a master branch and a separate develop branch as well as various other branches for features, releases and hotfixes. The various developments should be brought together in the develop branch, then transferred to the release branch and finally end up in the master branch. Git Flow is a well-defined, but complex standard that practically has the following two problems:

- Most developers and tools make the assumption that the master branch is the main branch from which they run branch and merge. With Git Flow, there is now additional effort because you always have to switch to the develop branch first.
- The hotfixes and release branches also bring additional complexity, which is only likely to bring advantages in the rarest of cases.

In response to the problems of Git Flow, GitHub and Atlassian developed simpler alternatives that are mostly limited to so-called Feature branches.

See also:

Vincent Driessen: A successful Git branching model

**Feature branches**

GitHub Flow was intended as a greatly simplified alternative to *Git flow and its problems*, whereby there should only be different feature branches in addition to the master branch. Atlassian also recommends a similar strategy, but using rebase for the feature branches. These strategies offer two advantages:

- The code inventory remains relatively small as the feature branches are usually quickly adopted in the master branch.
- The workflows correspond to the usual methods of Continuous Delivery.

However, these workflows cannot answer how deployments in different environments or the division into different releases can take place. Options for this are described in *Deployment and release branches*.

See also:

- Feature Driven Development
- Feature Branches

**Deployment and release branches**

**Deployment branches**

They are recommended if, for example, you cannot determine the release time yourself, for example if an iOS application has to pass the app store validation or you only have a fixed time window available for deployment. In this case, a production branch prod, that reflects the code provided is recommended. Such a workflow prevents the additional work of git flow for releasing, tagging and merging.

Assuming you have a test, stage and prod environment, a merge request is first made for the test branch. If the tests are passed, the changes can also be adopted in the stage branch. When the QA decides that the code is ready for production, it can be transferred to the master branch. This process can be repeated several times until, for example, the time for the going life of these changes has come and a prod branch can be created.
Release branches

Release branches are recommended when software is to be delivered to customers. In this case each branch should contain a minor version, e.g. 2.7 or 3.4. Usually these branches are created from the master branch as late as possible. This reduces the number of merges that have to be distributed across multiple branches during bug fixes. Usually, these are first transferred to the master and then transferred from there to the release branch with `git cherry-pick`. This upstream first approach is e.g. used by Google and Red Hat. Every time a bug fix has been adopted in a release branch, the release is increased by a patch version with a Tag, see also Semantic Versioning.

Branches & merge requests

Merge requests support a workflow for regular deployments.

1. Create a feature branch

When you create a branch, you create a new environment in which to try new things. This does not affect the master branch. And you can be sure that the branch will not be merged with the master branch until it has been reviewed by someone you work with.

Note: Make sure that the names of the branches are as meaningful as possible, e.g. refactor-user-model oder user-content-cache.

Note: Make sure that the master branch only contains code that is suitable for deployment.

2. Adding commits

Once a feature branch has been created, you can start making changes. Whenever you add, edit or delete a file, you can record these changes in your branch. The progress will then be visible in your commits.

The commits also allow everyone else involved in the project to understand your work: what you’ve done and why.

Note: Commit messages not only facilitate the current understanding of your change, but also allow later, for example with `git blame`, to be able to understand why you made these changes.

3. Submit a merge request

Merge requests initiate a discussion about a bundle of commits. Because they are tightly integrated with the underlying Git repository, everyone involved in the project can see exactly what changes would be merged if the request is accepted.

You can also submit a merge request at any time if you get stuck and need help or advice. With @ in your comments you can also ask certain people from the project team for feedback.
4. Review and discussion of the code

After you have made a merge request, someone or the project team will submit questions or comments about your changes: The coding style may not match the project guidelines, or the lack of unit tests, or everything looks good. Merge requests are there to promote and document such discussions.

You can also execute `git push` on this branch after a merge request, e.g. to include fixes that arose from these discussions in this merge request. GitLab shows this new commit in the view of this merge request.

**Note:** The comments on your merge request are also written in Markdown so that you can use pre-formatted text blocks for source code etc. here too.

5. Deployment

GitLab can be used to create deployments, for example for automated testing or quality assurance in a staging environment. In this way, you can also ensure that the changes can be deployed.

6. Merge

If the deployment of your changes was also successful, your changes can be merged with the master branch.

**Note:** By inserting certain keywords in the text of your merge request, you can link issues with code in GitLab. If your merge request is confirmed, an issue can also be closed, for example. If you comment `/close #42` on a merge request, the item with the number 42 would also be closed when merging. You can find more information about this under GitLab quick actions.

Monorepos

Git is a very flexible version control system. In particular, branch and merge are powerful tools in distributed development environments. However, sometimes this also creates unnecessary complexity. In these cases it can make sense to work with a monolithic repository or monorepo.

**Definition**

- The repository contains several logical projects (e.g. an iOS client and a web application)
- These projects are usually only loosely connected to one another
- Most of these projects also have a linear history
Pros and cons

One advantage of monorepos can be that the effort involved in determining which versions of one project are compatible with which versions of the other project could be significantly reduced. This is at least always the case when all projects in a repository are processed by just one development team. Then it is advisable to get an executable version with each merge, even if the API was changed between the two projects.

However, a disadvantage can be a loss of performance. These can arise, for example, from:

**a large number of commits** Since Git uses DAGs (directed acyclic graphs) to display the history of a project, all operations that run through this graph, e.g. `git log` or `git blame`, will be slow.

**a large number of refs** A large number of branches and tags also slow down git. With `git ls-remote` you can display the refs of a repository and with `git gc` loose refs are summarised in a single file.

Every operation that goes through the commit process of a repository and has to take the individual refs into account, such as `git branch --contains <commit>`, becomes slow with a repo with many refs.

**a large number of versioned files** The directory cache index (`.git/index`) is used by Git to determine whether the file has been modified. Many processes, such as `git status` and `git commit`, slow down as the number of files increases.

**large files** Large files in a subtree or project reduce the performance of the entire repository.

8.1.13 git-big-picture

`git-big-picture` visualises Git repositories as DAGs. The tool comes with some filters to show only the interesting areas, e.g. the hierarchy of tags and branches.

Example

Here is the example of this tutorial with merges but without tags:

Installation

You can easily install `git-big-picture` with:

```bash
$ pipenv install git-big-picture
Installing git-big-picture...
Adding git-big-picture to Pipfile's [packages]...
✓ Installation Succeeded
...
```
Git-Integration

You can easily integrate the tool into Git by adding the script `git-big-picture` to `$PATH`. Then you can use it, e.g. with:

```
$ git big-picture -h
Usage: git-big-picture OPTIONS [<repo-directory>]

Options:
  --version           show program's version number and exit
  -h, --help          show this help message and exit
  --pstats=FILE       run cProfile profiler writing pstats output to FILE
  -d, --debug         activate debug output

Output Options:
  Options to control output and format
    -f FMT, --format=FMT
      set output format [svg, png, ps, pdf, ...]
    -g, --graphviz      output lines suitable as input for dot/graphviz
    -G, --no-graphviz   disable dot/graphviz output
    -p, --processed     output the dot processed, binary data
    -P, --no-processed  disable binary output
    -v CMD, --viewer=CMD
      write image to tempfile and start specified viewer
    -V, --no-viewer     disable starting viewer
    -o FILE, --outfile=FILE
      write image to specified file
    -O, --no-outfile    disable writing image to file

Filter Options:
  Options to control commit/ref selection
    -a, --all           include all commits
    -b, --branches      show commits pointed to by branches
    -B, --no-branches   do not show commits pointed to by branches
    -t, --tags          show commits pointed to by tags
    -T, --no-tags       do not show commits pointed to by tags
    -r, --roots         show root commits
    -R, --no-roots      do not show root commits
    -m, --merges        include merge commits
    -M, --no-merges     do not include merge commits
    -i, --bifurcations  include bifurcation commits
    -I, --no-bifurcations
      do not include bifurcation commits
```
Configuration

The standard git config infrastructure can be used to configure git-big-picture. Most of the command line arguments can be configured in a [big-picture] section. For example, to configure firefox as a viewer with

```
$ git config --global big-picture.viewer firefox
```

will create the following section in your ~/.gitconfig file:

```
[big-picture]
  viewer = firefox
```

Note: However, this disables other options at the same time. For example, you can no longer display the graph with Graphviz:

```
$ git-big-picture -g
 fatal: Options '-g | --graphviz' and '-p | --processed' are incompatible with other output options.
```

In this case you must also specify the -V or --no-viewer option:

```
$ git-big-picture -g -V
digraph {
  "c509669a01b156900eed9f1c9f927b6d2f7bb95b"[label="origin/pyup-scheduled-update-2020-11-16", color="/pastel13/2", style=filled];
  ...
```

8.1.14 Find regressions with git bisect

`git bisect` allows you to find the git commit that introduced a regression.

1. To do this, start the search with `git bisect start`. Then you can narrow down the area where a bug was introduced with `git bisect bad [COMMIT]` and `git bisect good [COMMIT]`. Alternatively, you can use the short form `git bisect start [BAD COMMIT] [GOOD COMMIT]`. `git bisect` then checks out a commit in the middle and asks you to test it, e.g.:

```
$ git bisect start v2.6.27 v2.6.25
Bisecting: 10928 revisions left to test after this (roughly 14 steps)
[2ec65f8b89ea003c27ff7723525a2ee335a2b393] x86: clean up using max_low_pfn on 32-bit
```

2. The search can now be continued manually or automatically with a script. Manually, you can use `git bisect bad` and `git bisect good` to narrow down the area in which an error was introduced. If this commit is found, the output may look like this:

```
$ git bisect bad
2ddcca36c8bca251724fe342c8327451988be0d is the first bad commit
commit 2ddcca36c8bca251724fe342c8327451988be0d
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date: Sat May 3 11:59:44 2008 -0700

  Linux 2.6.26-rc1
```

(continues on next page)
3. We then use `git show HEAD` to check what changes have been made in this commit:

```
$ git show HEAD
commit 2ddcca36c8bcfa251724fe342c8327451988be0d
Autor: Linus Torvalds <torvalds@linux-foundation.org>

Linux 2.6.26-rc1

diff --git a / Makefile b / Makefile
index 5cf8258 ..4492984 100644
--- a / Makefile
+++ b / Makefile
@@ -1,7 +1,7 @@
VERSION = 2
PATCHLEVEL = 6
-SUBLEVEL = 25
+SUBLEVEL = 26
+EXTRAVERSION = -rc1
NAME = Funky Weasel ist Jiggy wit it

# * DOKUMENTATION *
```

Checking whether faulty code was introduced with a commit can also be automated. You can find an example of this in the issue `fetch_california_housing fails in CI on master` from scikit-learn:

```
git bisect run pytest sklearn/utils/tests/test_multiclass.py -k test_unique_labels_˓→non_specific
```

4. The scikit-learn-issue also shows how you can tell others the results of your bisect search in a traceable way using `git bisect log`:

```
$ git bisect log
81f2d3a0e * massich/multiclass_type_of_target Merge branch 'master' into...
     --multiclass_type_of_target
     |\ 15f24f25d | * bad DOC Cleaning for what's new
     | fbb2c7c70 | * good-fbb2c7c707dc373c462e39ab273a183a8823d58 @ ENH Adds _
     |     —MultimetricScorer for Optimized Scoring (#14593)
...
```

With `git bisect log > bisect_log.txt` you can save your search in a reproducible way for others:

```
$ git bisect replay bisect_log.txt
```

5. Finally, you can use `git bisect reset` to return to the branch you were in before the bisect search:

```
$ git bisect reset
Checking out files: 100% (21549/21549), done.
```
8.1.15 Git glossary

**Git**  Git is a distributed version control system.

**GitLab**  Web application for version management based on **git**. RGitlab CI, a system for continuous integration, GitLab Runner, container registry and much more were added later.

**git commit**  a snapshot of the entire Git repository, compressed in a **SHA**

**branch**  a lightweight moving pointer to a commit

**clone**  local version of a repository including all commits and branches

**remote**  shared repository, e.g. on GitLab, for exchanging changes in a team

**fork**  Copy of a repository on GitLab that belongs to another user

**Merge request**  Place to compare and discuss the changes introduced in a branch with ratings, comments, tests etc.; see also Merge requests.

**HEAD**  The **HEAD** pointer represents your current working directory and can be moved to different branches, tags or commits with **git checkout**.

8.2 Manage data with **DVC**

For data analysis, and especially machine learning, it is extremely valuable to be able to reproduce different versions of analyses that have been carried out with different data sets and parameters. However, in order to obtain reproducible analyses, both the data and the model (including the algorithms, parameters, etc.) must be versioned. Versioning data for reproducible analysis is a bigger problem than versioning models because of the size of the data. Tools like **DVC** help manage data by allowing users to transfer it to a remote data store using a **Git** like workflow. This simplifies the retrieval of certain versions of data in order to reproduce an analysis.

**DVC** was developed to be able to use ML models and data sets together and to manage them in a comprehensible manner. It works with different version managements, but does not need them. In contrast to **DataLad/gita annex**, for example, it is not limited to Git as version management, but can also be used together with Mercurial, see github.com/crobarcro/dvc/dvc/scm.py. It also uses its own system for storing files with support for SSH and HDFS, among others.

**DataLad**, on the other hand, focuses more on discovering and consuming datasets, which are then easily managed with Git. **DVC**, on the other hand, stores each step in the pipeline in a separate **.dvc** file that can then be managed by Git.

These **.dvc** files, however, allow practical tools for manipulating and visualizing DAGs, see, for example, **visualisation of DAGs**.

External dependencies can also be specified with **dvc remote**.

See also:

- Tutorial
8.2.1 Installation

Finally, external dependencies can also be specified with Pipenv.

**Note:** You have to explicitly state the extras. This can be [ssh], [s3], [gs], [azure], and [oss] or [all]. For ssh the command looks like this:

```
$ pipenv install dvc[ssh]
```

Alternatively, DVC can also be installed via the package management of Ubuntu/Debian with:

```
$ sudo wget https://dvc.org/deb/dvc.list -O /etc/apt/sources.list.d/dvc.list
$ sudo apt update
$ sudo apt install dvc
```

For macOS DVC can be installed with:

```
$ brew install iterative/homebrew-dvc/dvc
```

**Note:** The following example was created with a current DVC version (1.0.0a9), which partly uses a different syntax than earlier versions. You can currently (8th June 2020) only install this with pip:

```
$ pipenv install dvc[all]==1.0.0a9
```

**Create a project**

DVC can be easily initialised with:

```
$ mkdir -p dvc-example/data
$ cd dvc-example
$ git init
$ dvc init
$ git add .dvc
$ git commit -m "Initialise DVC"
```

`dvc init` creates a directory `.dvc/` with config, .gitignore and cache directory.

`git commit` puts .dvc/config and .dvc/.gitignore under version control.
Configure

Before DVC is used, even a remote storage is established. This should be accessible to everyone who should access the data or the model. It’s similar to using a Git server. Often, however, this is also an NFS mount, which can be integrated as follows, for example:

```
$ sudo mkdir -p /var/dvc-storage
$ dvc remote add -d local /var/dvc-storage
Setting 'local' as a default remote.
$ git commit .dvc/config -m "Configure local remote"
[master efaeb84] Configure local remote
 1 file changed, 4 insertions(+)
```

-d, --default Default value for the space removed

local Name of the remote location

/var/dvc-storage URL of the remote location

In addition, other protocols are supported, which are preceded by the path, including ssh:, hdfs: and https:

Another remote data storage can simply be added, e.g. with:

```
$ dvc remote add webserver https://dvc.example.org/myproject
```

The associated configuration file .dvc/config looks like this:

```
[remote "local"]
url = /var/dvc-storage
[core]
remote = local
[remote "webserver"]
url = https://dvc.example.org/myproject
```

Add data and directories

With DVC you can save and version files, ML models, directories and intermediate results with Git without having to check the file content into Git:

```
$ dvc get https://github.com/iterative/dataset-registry get-started/data.xml -o data/data.xml
$ dvc add data/data.xml
```

This will add the file data/data.xml in data/.gitignore and write the meta information in data/data.xml.dvc. Further information on the file format of the *.dvc can be found under DVC-File Format.

In order to be able to manage different versions of your project data with Git, you only have to add the CVS file:

```
$ git add data/.gitignore data/fortune500.csv.dvc
$ git commit -m "Add raw data to project"
```
Store and retrieve data

The data can be copied from the working directory of your Git repository to the remote storage space with

```
$ dvc push
```

If you want to call up more current data, you can do so with

```
$ dvc pull
```

Import and update

You can also import data and models from another project with the command `dvc import`, e.g.:

```
$ dvc import https://github.com/iterative/dataset-registry get-started/data.xml
```

This loads the file from the `dataset-registry` into the current working directory, adds `.gitignore` and creates `data.xml.dvc`.

With `dvc update` we can update these data sources before we reproduce a pipeline that depends on these data sources, e.g.

```
$ dvc update data.xml.dvc
Stage 'data.xml.dvc' didn't change.
Saving information to 'data.xml.dvc'.
```

Pipelines

Connect code and data

Commands like `dvc add`, `dvc push` and `dvc pull` can be made independently of changes in the Git repository and therefore only provide the basis for managing large amounts of data and models. In order to actually achieve reproducible results, code and data must be linked together.

With `dvc run` you can create individual processing levels, each level being described by a source code file managed with Git as well as other dependencies and output data. All stages together then form the DVC pipeline.

In our example `dvc-example`, the first stage is to split the data into training and test data:

```
$ dvc run -n split -d src/split.py -d data/data.xml -o data/splitted \
    python src/split.py data/data.xml
```

- `n` indicates the name of the processing stage.
- `d` dependencies on the reproducible command.

  The next time `dvc repo` is called to reproduce the results, DVC checks these dependencies and decides whether they need to be updated or run again to get more current results.

- `o` specifies the output file or directory.

In our case, the work area should have changed to:

```
8.2. Manage data with DVC 347
```
The generated `dvc.yaml` file looks like this, for example:

```yaml
stages:
  split:
    cmd: pipenv run python src/split.py data/data.xml
    deps:
      - data/data.xml
      - src/split.py
    outs:
      - data/splitted
```

Since the data in the output directory should never be versioned with Git, `dvc run` has already written the file `data/.gitignore`:

```
/data.xml
+ /splitted
```

Then the changed data only has to be transferred to Git or DVC:
If several phases are now created with `dvc run` and the output of one command being specified as a dependency of another, a DVC Pipeline is created.

Parameterisation

In the next phase of our example, we parameterise the processing and create the file `params.yaml` with the following content:

```yaml
max_features: 6000
ngram_range:
  lo: 1
  hi: 2
```

To read the parameters, the option `-p <filename>:<params_list>` must be added to the command `dvc run`, in our example:

```
$ dvc run -n featurise -d src/featurisation.py -d data/splitted
  -p params.yaml:max_features,ngram_range.lo,ngram_range.hi -o data/features
  python src/featurisation.py data/splitted data/features
```

This adds to the `dvc.yaml` file:

```yaml
featurise:
  cmd: python src/featurisation.py data/splitted data/features
  deps:
    - data/splitted
    - src/featurisation.py
  params:
    - max_features
    - ngram_range.lo
    - ngram_range.hi
  outs:
    - data/features
```

So that this phase can be repeated, the MD5 hash values and parameter values are stored in the file `dvc.lock`:

```
featurise:
  cmd: python src/featurisation.py data/splitted data/features
  deps:
    - path: data/splitted
      md5: 1ce9051bf386e57c03fe779d476d93e7.dir
    - path: src/featurisation.py
      md5: a56570e715e39134adb4f7c779296373
  params:
    params.yaml:
      max_features: 1000
      ngram_range.hi: 2
      ngram_range.lo: 1
```

Finally `dvc.lock`, `dvc.yaml` and `data/.gitignore` in the Git repository need to be updated:
$ git add dvc.lock dvc.yaml data/.gitignore

See also:
- dvc params

Trial metrics

With the `dvc metrics` command, DVC is also a framework for recording and comparing the performance of experiments. `evaluate.py` calculates the AUC (Area Under the Curve). It uses the test data set, reads the features from the file `features/test.pkl` and creates the metrics file `auc.metric`. It can be identified as a DVC metric with the `-M` option of `dvc run`, in our example with:

```
$ dvc run -n evaluate -d src/evaluate.py -d model.pkl -d data/features \
   -M auc.json python src/evaluate.py model.pkl data/features auc.json
```

```
evaluate:
   cmd: python src/evaluate.py model.pkl data/features auc.json
   deps:
   - data/features
   - model.pkl
   - src/evaluate.py
   metrics:
   - auc.json:
     cache: false
```

With `dvc metrics show` experiments can be compared then through various branches and tags:

```
$ dvc metrics show
          auc.json: 0.514172
```

Now to complete our first version of the DVC pipeline, let's add the files and a tag to the Git repository:

```
$ git add dvc.yaml dvc.lock auc.json
$ git commit -m 'Add stage ‹evaluate›'
$ git tag -a 0.1.0 -m 'Initial pipeline version 0.1.0'
```

View pipelines

Such data pipelines can be displayed or represented as a dependency graph with `dvc dag`:

```
$ dvc dag
```

(continues on next page)
With dvc dag --dot a .dot file for Graphviz is generated:
Reproduce

To reproduce the results of a project, we first clone the data managed with DVC:

```
$ git clone https://github.com/veit/dvc-example.git
$ cd dvc-example
$ dvc pull -TR
A data/data.xml
1 file added
$ ls data/
data.xml  data.xml.dvc
```

Then you can easily reproduce the results with `dvc repro`:

```
$ dvc repro
Verifying data sources in stage: 'data/data.xml.dvc'
Stage 'split' didn't change, skipping
Stage 'featurize' didn't change, skipping
```

(continues on next page)
Stage 'train' didn't change, skipping
Stage 'evaluate' didn't change, skipping

You can now, for example, change parameters in the params.yaml file and then run through the pipeline again:

```bash
$ dvc repro
Stage 'data/data.xml.dvc' didn't change, skipping
Stage 'split' didn't change, skipping
Running stage 'featurize' with command:
  python src/featurization.py data/splitted data/features
...
Stage 'train' didn't change, skipping
Stage 'evaluate' didn't change, skipping
To track the changes with git, run:
  git add dvc.lock
```

In our case, changing the parameters had no effect on the result.

_**Note:**_ DVC recognises changes to dependencies and outputs via md5 hash values in dvc.lock.

### Vim and IDE integration

#### Vim

To recognize DVC files in Vim as YAML, you should add the following in `~/.vimrc`:

```vim
" DVC
autocmd BufNewFile,BufRead Dvcfile,.dvc setfiletype yaml
```

#### IntelliJ IDEs

intellij-dvc is a plugin for IntelliJ IDEs including PyCharm, IntelliJ IDEA and CLion. It can be downloaded from the JetBrains Plugins-Repository.

### 8.3 Create packages

1. Notebooks are well suited for moving forward quickly, but when the code becomes more extensive, it is advisable to store stable code in packages.
2. You can use `pytest` not only within your notebooks for testing, but also within your packages.
3. Use Clean Code principles with meaningful variable and function names, make meaningful comments and modularise the code.

   There are also tools that automatically apply coding styles such as PEP 8 for Python. With jupyter-autope8 you can not only apply this to your notebooks, but also to your Python packages, for example with black.

   For other languages you can find overviews in Awesome-Linters and awesome-code-formatters.
You can automatically execute these tools with a pre-commit hook in front of every `git commit`, e.g. `mirrors-autopep8`, `pygrep-hooks` or `blacken-docs`. You can get a good overview of available Git pre-commit hooks at pre-commit.com.

See also:

- Python Application Layouts
- The Hitchhiker’s Guide to Python: Structuring Your Project
- Poetry
- Python Modules
- Python Packaging

### 8.3.1 Example

1. Notebooks are well suited for making rapid progress, but when the code becomes more extensive, it is advisable to move stable code into modules. For example, if you wrote in your notebook:

   ```python
   df = pd.read_csv(filename)
   df.drop( ... 
   df.query( ... 
   df.groupby( ... 
   ```

   so you can outsource it to a file `dataprep.py`:

   ```python
   def load_and_preprocess_data(filename):
     """Documentation"
     # Do stuff
     # ...
     return df
   ```

   and this can be imported into the notebook with

   ```python
   import dataprep
   df = dataprep.load_and_preprocess_data(filename)
   ```

   If you change the Python script, the updated variant can be automatically adopted with `IPython.extensions.autoreload`:

   ```bash
   %load_ext autoreload
   %autoreload
   ```

### 8.3.2 Create a distribution package

`Distribution Packages` are archives that can be uploaded to a package index and installed with `Pip`.

**Note:** There are still many instructions that contain a step of calling the `setup.py`, for example `python setup.py sdist`. However, this is now seen as an Anti-Pattern by parts of the Python Packaging Authority (PyPA).
You can find a minimal yet functional `setup.py` in the `attrs` package: `setup.py`. This tells you that most of it is boilerplate and only the lines 10–37 are metadata for this particular package. Most of the other metadata is stored in the `__init__` and is accessed using regular expressions. Alternatively, this data can also be stored in a separate module and analysed with Python, as it is done in cryptography.

In both cases, duplicate metadata in package and code is avoided.

### src-Package

The `packages` field uses setuptools’s `find_packages()` to find underlying packages and the `package_dir` field describes where the root directory is.

---

**Note:** `find_packages()` without `src/` directory would package all directories with an `__init__.py` file, including `tests/` directories.

---

### version

For version there are different ways described in PEP 0440.

**See also:**

- Semantic Versioning
- Calendar Versioning
- `bump2version`
- Git Tags

### classifiers

Classifiers have a useful additional function: PyPI rejects unknown classifiers, so that accidental uploads can be avoided.

**See also:**

Add invalid classifier for non open source license to avoid upload to…

### Dependencies

Version numbers of dependencies should usually not be set in the `setup.py` but in the `requirements.txt`.

**See also:**

`setup.py` vs `requirements.txt`
Other files

**MANIFEST.in**

The file contains all files and directories that are not already recorded with `packages` or `py_module`. For example, it could look like this: `attrs/MANIFEST.in`.

Further instructions in *Manifest.in* can be found in *Creating a source distribution*.

**Note:** Often people forget to update the `Manifest.in` file. To avoid this, you can use `check-manifest` in a `pre-commit` hook.

**Note:** If you want to install files and directories from `MANIFEST.in`, for example, when it comes to runtime-relevant data, you can specify this with `include_package_data=True` in your `setup()` call.

**setup.cfg**

This file is no longer needed, at least not for packaging. Today `wheel` collects all the necessary license files automatically and `setuptools` with the `options` keyword argument creates universal whell packages e.g. `attrs-19.3.0-py2.py3-none-any.whl`.

**pyproject.toml**

PEP 517 and PEP 518 brought plugable build backends, isolated builds, and `pyproject.toml`. Since we're using `setuptools`, the file should look something like this:

```
[build-system]
requires = ["setuptools>40.6.0", "wheel"]
build-backend = "setuptools.build_meta"
```

**LICENSE**

You can get an overview of free and open-source software licenses in *Comparison of free and open-source software licenses*.

For example, if you want to achieve the widest possible distribution of your package, MIT or BSD variants are a good choice. The Apache license protects you better against patent infringements, but isn't compatible with the GPL v2. Therefore, you should see which licenses have the packages that you depend on and with which you should be compatible. To analyse licenses, you can use `licensechecker`, a command line tool that searches installation directories for licenses.

It can also be useful to publish a package under several licenses. An example of this is `cryptography/LICENSE`.
**README.rst**

This file tells potential users what to look out for when using the package. Write the document in ReStructuredText (ReST), so that you can easily transfer it to the Sphinx documentation later with .. include:: ../../README.rst.

**CHANGELOG.rst**

See also:
- Keep a Changelog
- towncrier

**Build**

Change to the directory in which the setup.py file is located.

```bash
$ rm -rf build dist
$ pipenv run python3 -m pep517.build .
```

The first line ensures that a clean build is produced with no artifacts from previous builds. The second line builds an sdist archive under Linux/Mac as a zipped tar file (.tar.gz) and under Windows a ZIP file as well as an bdist_wheel archive .whl in the dist directory.

So this command should produce the following two files:

```bash
$ tree dist/
dist/
  ├── example-0.0.1-py3-none-any.whl
  └── example-0.0.1.tar.gz
```

- **py3** Python version that the package was built with
- **none** not OS specific
- **any** suitable for every processor architecture

You can find the reference for the file names in File name convention.

See also:
For more information, see Creating a Source Distribution. and PEP 376.

**Note:** The use of pep517.build to create packages is currently (October 2019) a bit controversial. There seems to be a consensus that this functionality should be merged into either Pip or Twine. At the moment, however, the above seems like the cleanest way to package a package. I will update this article as soon as another solution prevails.
Testing

```bash
$ pipenv --rm
$ pipenv install dist/attrs-19.3.0.tar.gz
... Successfully built attrs
Installing collected packages: attrs
Successfully installed attrs-19.3.0
$ pipenv run python
...
>>> import attr; attr.__version__
'19.3.0'
```

or

```bash
$ pipenv --rm
$ pipenv install dist/attrs-19.3.0-py2.py3-none-any.whl
... Successfully built attrs
Installing collected packages: attrs
Successfully installed attrs-19.3.0
$ pipenv run python
...
>>> import attr; attr.__version__
'19.3.0'
```

See also:

- [PyPI Release Checklist](#)

### 8.3.3 Upload

Finally, you should provide the package on the [Python Package Index (PyPI)](https://pypi.org) or another index.

For this you should register on Test PyPI. Test-PyPI is a separate instance that is intended for testing and experimentation. To set up an account there, go to [https://test.pypi.org/account/register/](https://test.pypi.org/account/register/). For more information, see [Using TestPyPI](https://test.pypi.org/docs/).

Now you can create the `~/.pypirc` file:

```
[distutils]
index-servers=
    test

[test]
repository = https://test.pypi.org/legacy/
username = veit
```

See also:

If you’d like to automate PyPI registration, please read [Careful With That PyPI](https://pypi.org/). After you are registered, you can upload your [Distribution Package](https://pypi.org) with `twine`. To do this, however, you must first install `twine` with:
Now you can create your *Distribution Packages* with:

```
$ rm -rf build dist
$ pipenv run python -m pep517.build .
```

After installing Twine you can upload all archives in /dist to the Python Package Index with:

```
$ pipenv run twine upload -r test -s dist/*
```

- **-r, --repository** The repository to upload the package. In our case, the `test` section from the `~/.pypirc` file is used.
- **-s, --sign** signs the files to be uploaded with GPG.

You will be asked for the password you used to register on *Test PyPI*. You should then see a similar output:

```
Uploading distributions to https://test.pypi.org/legacy/
Enter your username: veit
Enter your password:
Uploading example-0.0.1-py3-none-any.whl
100%| 4.65k/4.65k [00:01<00:00, 2.88kB/s]
Uploading example-0.0.1.tar.gz
100%| 4.25k/4.25k [00:01<00:00, 3.05kB/s]
```

**Note:** If you get an error message similar to

```
The user 'veit' isn't allowed to upload to project 'example'
```

you have to choose a unique name for your package:

1. change the name argument in the `setup.py` file
2. remove the `dist` directory
3. regenerate the archives
Check

Installation

You can use `pipenv` to install your package and check if it works. Create a new virtual environment and install your package on Test PyPI:

```
$ mkdir test
$ cd !$
$ pipenv install --extra-index-url https://test.pypi.org/simple/ minimal_example
```

**Note:** If you have used a different package name, replace it with your package name in the command above.

`pip` should install the package from Test PyPI and the output should look something like this:

```
Collecting example_pkg
   Downloading https://test-files.pythonhosted.org/packages/.../minimal_example-0.0.1-py3-˓
   ...none-any.whl
Installing collected packages: minimal_example
Successfully installed minimal_example-0.0.1
```

You can test whether your package has been installed correctly by importing the module and referencing the name property that was previously entered in `__init__.py`:

```
$ pipenv run python
Python 3.7.0 (default, Aug 22 2018, 15:22:29)
...  
>>> import minimal_example
>>> minimal_example.name
'minimal_example'
```

**README**

Also check whether the READ ME.rst is displayed correctly on the test PyPI page.

**PyPI**

Now register on the Python Package Index (PyPI) and make sure that two-factor authentication is activated by adding the following to the ~/.pypirc file:

```
[distutils]
index-servers=
   pypi
test

[test]
repository = https://test.pypi.org/legacy/
username = veit
```

(continues on next page)
With this configuration, the name/password combination is no longer used for uploading but an upload token.

See also:

- PyPI now supports uploading via API token
- What is two factor authentication and how does it work on PyPI?

Finally, you can publish your package on PyPI:

```bash
$ pipenv run twine upload -r pypi -s dist/*
```

Note: You cannot simply replace releases as you cannot re-upload packages with the same version number. Do not remove old versions from the Python Package Index. This only causes work for those who want to keep using that version and then have to switch to old versions on GitHub. PyPI has a `yank` function that you can use instead. This will ignore a particular version if it is not explicitly specified with `==` or `===`.

See also:

- PyPI Release Checklist

**GitHub Action**

You can also create a GitHub action, which creates a package and uploads it to PyPI at every time a release is created. Such a `.github/workflows/pypi.yml` file could look like this:

```yaml
name: pypi
on:
push:
tags: 
  - *

jobs:
  package-and-deploy:

    runs-on: ubuntu-latest

    steps:
    - name: Checkout
      uses: actions/checkout@v2
      with:
        fetch-depth: 0

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: 3.8

    - name: Install dependencies
```

(continues on next page)
run: |
    python -m pip install -U pip
    python -m pip install -U setuptools twine wheel

- name: Build and publish
env:
    TWINE_PASSWORD: ${{ secrets.TWINE_PASSWORD }}
    TWINE_USERNAME: ${{ secrets.TWINE_USERNAME }}
run: |
    python setup.py sdist bdist_wheel
twine upload dist/*

See also:
- GitHub Actions

### 8.3.4 Binary Extensions

One of the features of the CPython interpreter is that in addition to executing Python code, it also has a rich C API available for use by other software. One of the most common uses of this C API is to create importable C extensions that allow things that are difficult to achieve in pure Python code.

#### Use Cases

The typical use cases for binary extensions can be divided into three categories:

**Accelerator modules** These modules are stand-alone and are only created to run faster than the corresponding pure Python code. Ideally, the accelerator modules always have a Python equivalent that can be used as a fallback if the accelerated version is not available on a particular system.

The CPython standard library uses many accelerator modules.

**Wrapper modules** These modules are created to make existing C interfaces available in Python. You can either make the underlying C interfaces directly available or provide a Pythonic API that uses features of Python to make the API easier to use.

The CPython standard library uses extensive wrapper modules.

**Low-level system access** These modules are created to access functions of the CPython runtime environment, the operating system or the underlying hardware. With platform-specific code, things can be achieved that would not be possible with pure Python code.

A number of CPython standard library modules are written in C to access interpreter internals that are not available at the language level.

A particularly noteworthy property of C extensions is that they can release the Global Interpreter Lock (GIL) of CPython for long-running operations, regardless of whether these operations are CPU or IO-bound.

Not all expansion modules fit exactly into the above categories. For example, the extension modules contained in NumPy cover all three use cases:

- They move inner loops to C for speed reasons,
- wrap external libraries in C, FORTRAN and other languages and
- use low-level system interfaces of CPython and the underlying operating system to support the concurrent execution of vectorised operations and to precisely control the memory layout of objects created.
Disadvantages

In the past, the main disadvantage of using binary extensions was that they made it difficult to distribute the software. Today this disadvantage due to wheel is hardly present. However, some disadvantages remain:

- The installation from the sources remains complicated.
- Possibly there is no suitable wheel for the build of the CPython interpreter or alternative interpreters such as PyPy, IronPython or Jython.
- The maintenance of the packages is more time-consuming because the maintainers not only have to be familiar with Python but also with another language and the CPython C API. In addition, the complexity increases if a Python fallback implementation is provided in addition to the binary extension.
- Finally, import mechanisms, such as direct import from ZIP files, often do not work for extension modules.

Alternatives

… to accelerator modules

If extensions modules are only used to make code run faster, a number of other alternatives should also be considered:

- Looks for existing optimised alternatives. The CPython standard library contains a number of optimised data structures and algorithms, especially in the builtins and the modules collections and itertools.
  Occasionally the Python Package Index (PyPI) also offers additional alternatives. Sometimes a third-party module can avoid the need to create your own accelerator module.

- For long-running applications, the JIT-compiled PyPy interpreter can be a suitable alternative to the standard CPython. The main difficulty with adopting PyPy is typically the dependence on other Binary Extensions modules. While PyPy emulates the CPython C API, modules that rely on it cause problems for the PyPy JIT, and the emulation often exposes defects in extension modules that CPython tolerates. (often with reference counting errors).

- Cython is a sophisticated static compiler that can compile most Python code into C-Extension modules. The initial compilation offers some speed increases (by bypassing the CPython interpreter level), and Cython’s optional static typing functions can provide additional speed increases. For Python programmers, Cython offers a lower barrier to entry relative to other languages such as C or C++.

  However, using Cython has the disadvantage of adding complexity to the distribution of the resulting application.

- Numba is a newer tool that uses the LLVM compiler infrastructure to selectively compile parts of a Python application to native machine code at runtime. It requires LLVM to be available on the system the code is running on. It can lead to considerable increases in speed, especially with vectorisable processes.

… to wrapper modules

The C-ABI (Application Binary Interface) is a standard for the common use of functions between several applications. One of the strengths of the CPython C-API (Application Programming Interface) is that Python users can take advantage of this functionality. However, manually wrapping modules is very tedious, so a number of other alternatives should be considered.

The approaches described below do not simplify distribution, but they can significantly reduce the maintenance effort compared to wrapper modules.

- Cython is useful not only for creating accelerator modules, but also for creating wrapper modules. Since the API still needs to be wrapped by hand, it is not a good choice when wrapping large APIs.
• **cffi** is the project of some PyPy developers to give developers who already know both Python and C the possibility to make their C modules available for Python applications. It makes wrapping a C module based on its header files relatively easy, even if you are not familiar with C itself.

One of the main advantages of cffi is that it is compatible with the PyPy JIT so that CFFI wrapper modules can fully participate in the PyPy tracing JIT optimisations.

• **SWIG** is a wrapper interface generator that combines a variety of programming languages, including Python, with C and C ++ code.

• The **ctypes** module of the standard library is useful to get access to C interfaces, but if the header information is not available, it suffers from the fact that it only works on the C ABI level and therefore no automatic consistency check between the exported Interface and the Python code. In contrast, the alternatives above can all work on the C API and use C header files to ensure consistency.

### ... for low-level system access

For applications that require low level system access, a binary extension is often the best option. This applies in particular to the low level access to the CPython runtime, since some operations (such as releasing the Global Interpreter Lock (GIL) are not permitted when the interpreter executes the code itself, especially when modules such as ctypes or cffi are used to Get access to the relevant C-API interfaces.

In cases where the expansion module is manipulating the underlying operating system or hardware (instead of the CPython runtime), it is sometimes better to write a normal C library (or a library in another programming language such as C++ or Rust) that provides a C-compatible ABI) and then use one of the wrapping techniques described above to make the interface available as an importable Python module.

### Implementation

The CPython Extending and Embedding guide provides an introduction to writing your own extension modules in C: Extending Python with C or C++. ItPlease note, however, that this tutorial only covers the basic extension building tools provided with CPython. Third-party tools like Cython, cffi, SWIG and Numba offer both simpler and more sophisticated approaches to creating C and C ++ extensions for Python.

See also:

Python Packaging User Guide: Binary Extensions not only covers various tools available to make creating Binary Extensions easier, but it also explains the various reasons why it might be desirable to create an Extension Module.

### Creating binary extensions

**Binary extensions for Windows**

Before you can create a binary extension, you have to make sure that you have a suitable compiler available. On Windows, Visual C is used to create the official CPython interpreter, and it should also be used to create compatible binary extensions:

**for Python 3.4**

1. install Microsoft Windows SDK for Windows 7 and .NET Framework 4
2. work with the SDK command prompt (with the environment variables and the SDK in PATH).
3. set DISTUTILS_USE_SDK=1.

**for Python 3.5+**
1. **install Visual Studio Code with Python Extension**

*Note:* Visual Studio is backwards compatible from Python 3.5, which means that any future version of Visual Studio can create Python extensions for all Python versions from version 3.5.

Building with the recommended compiler on Windows ensures that a compatible C library is used throughout the Python process.

### Binary Extensions for Linux

Linux binaries must use a sufficiently old glibc to be compatible with older distributions. Distrowatch prepares in table form which versions of the distributions deliver which library:

- Red Hat Enterprise Linux
- Debian
- Ubuntu
- ...

The PYP/Manylinux project facilitates the distribution of Binary extensions as *Wheels* for most Linux platforms. This also resulted in PEP 513, which defines the `manylinux1_x86_64` and `manylinux1_i686` platform tags.

### Binary Extensions for Mac

Binary compatibility on macOS is determined by the target system for the minimal implementation, e.g. *10.9*, which is defined in the environment variable `MACOSX_DEPLOYMENT_TARGET`. When creating with setuptools/distutils the deployment target is specified with the flag `--plat-name`, for example `macosx-10.9-x86_64`. For more information on deployment targets for Mac OS Python distributions, see the MacPython Spinning Wheels-Wiki.

### Deployment of binary extensions

In the following, the deployment on the *Python Package Index (PyPI)* or another index will be described.

*Note:* When deploying on Linux distributions, it should be noted that these make demands on the specific build system. Therefore, *Source Distributions (sdist)* should also be provided in addition to *Wheels*.

See also:

- Deploying Python applications
- Supporting Windows using Appveyor

---

**8.3. Create packages**
8.3.5 cibuildwheel

cibuildwheel simplifies the creation of Python Wheels for the different platforms and Python versions through Continuous Integration (CI) workflows. More precisely it builds manylinux, macOS 10.9+, and Windows wheels for CPython and PyPy with GitHub Actions, Azure Pipelines, Travis CI, AppVeyor, CircleCI, or GitLab CI.

In addition, it bundles shared library dependencies on Linux and macOS through auditwheel and delocate.

Finally, the tests can also run against the wheels.

See also:
- Docs
- GitHub

GitHub Actions

To build Linux, macOS, and Windows wheels, create a .github/workflows/build_wheels.yml file in your GitHub repo:

```yaml
name: Build

on: [push, pull_request]

jobs:
  build_wheels:
    name: Build wheels on ${{ matrix.os }}
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-20.04, windows-2019, macos-10.15]
    steps:
      - uses: actions/checkout@v2
      - name: Build wheels
        uses: pypa/cibuildwheel@v1.11.0
        # to supply options, put them in 'env', like:
        # env:
        # CIBW_SOME_OPTION: value
      - uses: actions/upload-artifact@v2
        with:
          path: ./wheelhouse/*.whl
```

This runs the CI workflow with the following default settings:

- package-dir: .
- output-dir: wheelhouse

You can extend the file to automatically upload the wheels to the Python Package Index (PyPI) with:

```yaml
upload_pypi:
  needs: [build_wheels, build_sdist]
  runs-on: ubuntu-latest
```

(continues on next page)
# upload to PyPI on every tag starting with 'v'
if: github.event_name == 'push' & startsWith(github.event.ref, 'refs/tags/v')
# alternatively, to publish when a GitHub Release is created, use the following rule:
# if: github.event_name == 'release' && github.event.action == 'published'
steps:
  - uses: actions/download-artifact@v2
    with:
      name: artifact
      path: dist

  - uses: pypa/gh-action-pypi-publish@master
    with:
      user: __token__
      password: ${{ secrets.pypi_password }}

# To test: repository_url: https://test.pypi.org/legacy/

See also:
- Workflow syntax for GitHub Actions

**Gitlab CI**

To build Linux wheels on Gitlab CI, create a `.gitlab-ci.yml` file in your repo:

```yaml
# To build Linux wheels on Gitlab CI, create a .gitlab-ci.yml file in your repo:

## Linux

image: python:3.8

# make a docker daemon available for cibuildwheel to use
services:
  - name: docker:dind
    entrypoint: ["env", "-u", "DOCKER_HOST"]
    command: ["dockerd-entrypoint.sh"]

variables:
  DOCKER_HOST: tcp://docker:2375/
  DOCKER_DRIVER: overlay2
# See https://github.com/docker-library/docker/pull/166
  DOCKER_TLS_CERTDIR: ""

script:
  - curl -sSL https://get.docker.com/ | sh
  - python -m pip install cibuildwheel==1.11.0
  - cibuildwheel --output-dir wheelhouse

artifacts:
  paths:
    - wheelhouse/

See also:
- Keyword reference for the .gitlab-ci.yml file
```

8.3. Create packages 367
Examples

- Coverage.py: .github/workflows/kit.yml
- matplotlib: .github/workflows/cibuildwheel.yml
- MyPy: .github/workflows/build.yml
- psutil: .github/workflows/build.yml
- scikit-learn: build_tools/github/build_wheels.sh

8.3.6 Templating

With Cookiecutter, file structures can be created which simplify the creation of Python packages significantly.

CookieCutter features

- Cross-platform: Windows, Mac and Linux are supported
- works with Python 3.6, 3.7, 3.8 and PyPy3
- The project templates can be created for any programming language and any markup format: Python, JavaScript, Ruby, ReST, CSS, HTML. Several languages can also be used in the same template.
- Templates can be easily adapted in the terminal:
  
  $ cookiecutter https://github.com/veit/cookiecutter-namespace-template
  full_name [Veit Schiele]: ...

- You can also use local templates:
  
  $ cookiecutter cookiecutter-namespace-template

- Alternatively you can also use CookieCutter with Python:
  
  $ bin/python
  >>> from cookiecutter.main import cookiecutter
  >>> cookiecutter('https://github.com/veit/cookiecutter-namespace-template.git')
  full_name [Veit Schiele]: ...

- Directory and file names can be assigned to templates, for example:

  {{cookiecutter.project_name}}/{{cookiecutter.namespace}}/{{cookiecutter.package_name}}/{{cookiecutter.project_slug}}.py

- The nesting depth is unlimited
- The templating is based on Jinja
- You can simply save your template variables in a cookiecutter.json file, for example:

  ```json
  {
    "full_name": "Veit Schiele",
    "email": "veit@example.org",
  }
  ```
  
(continues on next page)
You can also save the values for several templates in ~/cookiecutterrc:

```json

default_context:
    full_name: "Veit Schiele"
    email: "veit@cusy.io"
    github_username: "veit"
cookiecutters_dir: "~/.cookiecutters/"
```

CookieCutter templates loaded from a repository are usually stored in ~/.cookiecutters/. Then they can be referenced directly via their directory name, e.g. with:

```
$ cookiecutter cookiecutter-namespace-package
```

---

**Available templates**

**Python**

- **cookiecutter-namespace-template** Namespace template for Python packages
- **cookiecutter-pypackage** Template for Python packages
- **cookiecutter-pytest-plugin** Minimal Cookiecutter template for creating Pytest plugins
- **cookiecutter-pylibrary** Comprehensive template for Python packages with support for tests and Deployments (C extension support for cffi and Cython, test support for Tox, Pytest, Travis-CI, Coveralls, Codacy, and Code Climate, documentation with Sphinx, packaging checks with Landscape, scrutinizer, Isort <https://github.com/PyCQA/isort> etc.
- **cookiecutter-python-cli** Template for creating a Python CLI application with Click
- **widget-cookiecutter** Template for creating Jupyter widgets
Ansible

cookiecutter-ansible-role-ci  Template for Ansible roles

C

bootstrap.c Template for projects written in C with Autotools
cookiecutter-avr  Template for AVR development

C++

BoilerplatePP cmake template with unit tests for C++ projects

Scala

cookiecutter-scala Template for a Hello world example with a few libraries
cookiecutter-scala-spark Template for an Apache-Spark application

LaTeX/XeTeX

pandoc-talk Template for presentations with pandoc and XeTeX

Overview

A minimal CookieCutter template looks like this:

```
cookiecutter-namespace-template/
  {{ cookiecutter.project_name }}/ <--- Project template
  ... 
  cookiecutter.json <--- Prompts & default values
```

For json example, the file `cookiecutter.json` can look like this:

```
{
  "full_name": "Veit Schiele",
  "email": "veit@example.org",
  "github_username": "veit",
  "project_name": "vsc.example",
  "project_slug": "{{ cookiecutter.project_name.lower().replace(' ', '_').replace('-', '_') }}",
  "namespace": "{{ cookiecutter.project_slug.split('.')[0] }}",
  "package_name": "{{ cookiecutter.project_slug.split('.')[1] }}",
  "project_short_description": "Python Namespace Package contains all you need to create a Python namespace package."
  "pypi_username": "veit",
  "use_pytest": "y",
  "command_line_interface": ["Click", "No command-line interface"],
  "version": "0.1.0"
}
```

(continues on next page)
"create_author_file": "y",
"license": ["MIT license", "BSD license", "ISC license", "Apache Software License 2.0",
"GNU General Public License v3", "Not open source"]
}

In addition, any number of directories and files can be created.
As a result you will get the following file structure:

```
my.package/ <-- Value corresponding to what you enter
    at the project_name prompt
    ...
 <-- Files corresponding to those in your
    cookiecutter's
    {{ cookiecutter.project_name }}/ directory
```

**Installation**

**Requirements**

- Python interpreter
- Path to the base directory for your Python packages

Make sure your bin bindirectory is in the path. Usually this is `~/.local/` for Linux and Mac OS or `%APPDATA%\Python` on Windows. You can find more information at site.USER_BASE.

  - Linux and MacOS
    
    For bash you can enter the path in your `~/.bash_profile`:

    ```
    export PATH=$HOME/.local/bin:$PATH
    ```
    
    and then read the file with:

    ```
    $ source ~/.bash_profile
    ```

  - Windows
    
    Make sure the directory where CookieCutter will be installed is in your Path so you can go directly to it.
    
    To do this, look for `Environment Variables` on your computer and add this directory to Path, for example `%APPDATA%\Python\Python3x\Scripts`. Then you probably have to restart the session in order to be able to use the environment variables.

    **See also:**
    
    Configuring Python

    - `pipenv`
Jupyter Tutorial, Release 0.8.0

Installation

```
$ pipenv install --user cookiecutter
```

Advanced usage

Hooks

You can write pre- or post-generate hooks. The Jinja template variables will be integrated into the scripts, for example:

```
if 'Not open source' == '{cookiecutter.license}':
    remove_file('LICENSE')
```

Variables, for example, can be validated in a pre-generate hook:

```python
import re
import sys

MODULE_REGEX = r'^[a-zA-Z][_a-zA-Z0-9]+$'

module_name = '{cookiecutter.module_name}'

if not re.match(MODULE_REGEX, module_name):
    print('ERROR: %s is not a valid Python module name!' % module_name)
    # exits with status 1 to indicate failure
    sys.exit(1)
```

User config

If you use CookieCutter frequently, we recommend your own user config `~/cookiecutterrc`, e.g.:

```
default_context:
    full_name: "Veit Schiele"
    email: "veit@cusy.io"
    github_username: "veit"
    cookiecutters_dir: "~/.cookiecutters/"
    replay_dir: "~/.cookiecutter_replay/"
```

Replay

When calling cookiecutter a json file is created in `~/cookiecutter_replay/`, for example `~/cookiecutter_replay/cookiecutter-namespace-template.json`:

```
{"cookiecutter": {"full_name": "Veit Schiele", "email": "veit@cusy.io", "github_username": "veit", "project_name": "vsc.example", "project_slug": "vsc.example", "namespace": "vsc", "package_name": "example", "project_short_description": "Python Namespace Package contains all you need to create a Python namespace package.", "pypi_username": "veit", "use_pytest": "y", "command_line_interface": "Click", "version": "0.1.0", "create_author_file": "y", "license": "MIT license", "_template": "https://github.com/veit/cookiecutter-namespace-template"}}
```
If you want to use this information without having to confirm them again in the command line, you can simply enter the following:

```bash
$ cookiecutter --replay gh:veit/cookiecutter-namespace-template
```

Alternatively, the Python API can also be used:

```python
from cookiecutter.main import cookiecutter
cookiecutter('gh:veit/cookiecutter-namespace-template', replay=True)
```

This function is helpful if you want to create a project from an updated template, for example.

### Selection variables

Selection variables offer various options when creating a project. Depending on the user's choice, the template renders it differently, e.g. if in the `cookiecutter.json` file the following selection is offered:

```json
{
    "license": ["MIT license", "BSD license", "ISC license", "Apache Software License 2.0", "GNU General Public License v3", "Other/Proprietary License"]
}
```

This is interpreted in `cookiecutter-namespace-template/{{cookiecutter.project_name}}/README.rst`

```rst
{% set is_open_source = cookiecutter.license != 'Not open source' -%}
{% if is_open_source %}
...
{% endif %}
{% if is_open_source %}
...
{% endif %}
```

and in `cookiecutter-namespace-template/hooks/post_gen_project.py`:

```python
if 'Not open source' == '{{ cookiecutter.license }}':
    remove_file('LICENSE')
```

### 8.3.7 Next steps

Please keep in mind that your package on Test-PyPI is only stored temporarily. If you want to upload a package to the real Python Package Index (PyPI), you can do so by creating an account on https://pypi.org and following the same instructions, but using `twine upload dist/*`.

And if you want to learn more about Python libraries, you can

- read more about how `setuptools` can be used to pack and distribute packages: Packaging and distributing projects.
- look at alternatives to `setuptools` like `flit`, `hatch` and `poetry`.

---

8.3. Create packages
8.4 Document

So that your product can be used effectively, documentation is required for the target groups of data scientists and data engineers as well as for system engineers:

- Data scientists want to see documented
  - which problems your product solves and what the main functions and limitations of the software are (README)
  - how the product can be used
  - which changes have come in more recent software versions (CHANGELOG)
- Data engineers want to know how troubleshooting can help improve the product (CONTRIBUTING) and how they can communicate with others (CODE_OF_CONDUCT)
- System engineers need installation instructions for your product and the required dependencies

Together, they all need information about how the product is licensed (LICENSE file or LICENSES folder and how they can get help if needed.

Badges are helpful in getting a quick overview of a product. For the cookiecutter-namespace-template these are, for example:

For extensive documentation you can, for example, use Sphinx, a documentation tool that converts reStructuredText, a simple markup language, into HTML or PDF, EPub and man pages. The Jupyter tutorial was also created with Sphinx. To get a first impression of Sphinx, you can have a look at the source code of this page by following the link Sources. Originally, Sphinx was developed for the documentation of Python and is now used in almost all Python projects, including NumPy and SciPy, Matplotlib, Pandas and SQLAlchemy.

The Sphinx autodoc feature, which can be used to create documentation from Python Docstrings, may also be conducive to the spread of Sphinx among Python developers. Overall, Sphinx allows developers to create complete documentation in place. Often the documentation is also stored in the same Git repository, so that the creation of the latest software documentation remains easy.

Sphinx is also used in projects outside the Python community, e.g. for the documentation of the Linux kernel: Kernel documentation update.

Read the Docs was developed to further simplify documentation. Read the Docs makes it easy to create and publish documentation after each commit.

See also:
- Eric Holscher: Why You Shouldn’t Use “Markdown” for Documentation
- Tom Johnson: 10 reasons for moving away from DITA
- Tom Johnson: Jekyll versus DITA
- Google developer documentation style guide
- Google Technical Writing Courses for Engineers
- Cusy-Design-System: Schreiben
8.4.1 Create a Sphinx project

Installation and start

```
$ mkdir example
$ cd !$
cd example
$ pipenv install sphinx
Creating a virtualenv for this project...
...
$ pipenv run sphinx-quickstart docs
Selected root path: docs
> Separate source and build directories (y/n) [n]: y
> Name prefix for templates and static dir [-]:
> Project name: my.package
> Author name(s): Veit Schiele
> Project release []: 1.0
> Project language [en]:
> Source file suffix [.rst]:
> Name of your master document (without suffix) [index]:
> autodoc: automatically insert docstrings from modules (y/n) [n]: y
> doctest: automatically test code snippets in doctest blocks (y/n) [n]: y
> intersphinx: link between Sphinx documentation of different projects (y/n) [n]: y
> todo: write "todo" entries that can be shown or hidden on build (y/n) [n]: y
> coverage: checks for documentation coverage (y/n) [n]:
> imgmath: include math, rendered as PNG or SVG images (y/n) [n]:
> mathjax: include math, rendered in the browser by MathJax (y/n) [n]:
> ifconfig: conditional inclusion of content based on config values (y/n) [n]:
> viewcode: include links to the source code of documented Python objects (y/n) [n]:
> githubpages: create .nojekyll file to publish the document on GitHub pages (y/n) [n]:
> Create Makefile? (y/n) [y]:
> Create Windows command file? (y/n) [y]:

Creating file docs/source/conf.py.
Creating file docs/source/index.py.
Creating file docs/makefile.
Creating file docs/make.bat.
```

Sphinx layout

```
example
  ├── Pipfile
  │    └── docs
  │        ├── Makefile
  │        │    └── _build
  │        │        └── _static
  │        │        └── _templates
  │        ├── conf.py
  │        └── index.rst
  │                  └── make.bat
```

index.rst is the initial file for the documentation, in which the table of contents is located. The table of contents will
be expanded by you as soon as you add new *.rst files.

8.4.2 reStructuredText

Quick guide

You can view the following reStructuredText as HTML at rest-example:

```plaintext
Underline the title with punctuation marks
==========================================

Change the punctuation mark for subtitles
-----------------------------------------

*Italic*, **bold** and `preformatted`

'hyperlink <http://en.wikipedia.org/wiki/Hyperlink> `_ `link`_

.. _link: http://en.wikipedia.org/wiki/Link_(The_Legend_of_Zelda)
.. image:: python-logo.png

A paragraph consists of one or more lines of non-indented text, separated from the material above and below by blank lines.

»Block quotation marks look like paragraphs, but are indented with one or more spaces.«

| Because of the pipe character, this becomes one line.
| And this will be another line.

term

Definition of the term

Different term

...and its definition

* Each entry in a list begins with an asterisk (or `1.`

`a.` etc.).

* List items can be displayed for multiple lines as long as the list items remain indented

Blocks of code are introduced and indented with a colon::

.. code-block:: python

    import docutils
    print help(docutils)

>>> print 'But doctests start with ">>>" and don’t need to be indented.'
```

Note: If the content of long_description in setuptools.setup() is written in reStructured Text, it is displayed as well-formatted HTML on the Python Package Index (PyPI).

See also:
Directives

reStructuredText can be expanded with Directives. Sphinx makes extensive use of this. Here are some examples:

Table of Contents

.. toctree::
   :maxdepth: 2

   rest
docstrings

Docstrings

With the Sphinx extension `sphinx.ext.autodoc`, docstrings can also be included in the documentation. The following three directives can be specified:

.. automodule::
.. autoclass::
.. autoexception::

These document a module, a class or an exception using the docstring of the respective object.

Installation

`sphinx.ext.autodoc` is usually already specified in the Sphinx configuration file `docs/conf.py`:

```python
extensions = [
    'sphinx.ext.autodoc',
    ...
]
```

If your package and its documentation are part of the same repository, they will always have the same relative position in the filesystem. In this case you can simply edit the Sphinx configuration for `sys.path` to indicate the relative path to the package, so:

```python
sys.path.insert(0, os.path.abspath('..'))
import requests
```

If you have installed your Sphinx documentation in a virtual environment, you can also install your package there with:

```
$ pipenv install my.package
```

or, if you want to develop the package further with:

```
$ pipenv install -e https://github.com/veit/my.package.git
```
Examples

Here are some examples from the API documentation for the `requests` module:

```
Developer Interface
-------------------
.. module:: requests
...
Main Interface
--------------
...
.. autofunction:: head
...
Exceptions
---------
..
.. autoexception:: requests.RequestException
...
Request Sessions
---------------
...
.. autoclass:: Session
:inherited-members:
```

This leads to the docstrings-example, generated from the following docstrings:

- `requests.head`
- `requests.RequestException`
- `requests.Session`

**Note:** You should follow these guidelines when writing docstrings:

- Python Style Guide: comments
- The Docstring Conventions Guide

`sphinx-autodoc-typehints`

With PEP 484 a standard method for expressing types in Python code was introduced. This also allows types to be expressed differently in docstrings. The variant with types according to PEP 484 has the advantage that type testers and IDEs can be used for static code analysis.

Python 3 type annotations:

```
def func(arg1: int, arg2: str) -> bool:
    """Summary line.
    Extended description of function.
    Args:
    arg1: Description of arg1
```

(continues on next page)
Types in Docstrings:

```python
def func(arg1, arg2):
    
    """Summary line.
    
    Extended description of function.
    
    Args:
    arg1 (int): Description of arg1
    arg2 (str): Description of arg2
    
    Returns:
    bool: Description of return value
    
    """
    return True
```

**Note:** Python 2/3 compatible annotations are currently not supported by Sphinx and do not appear in the generated documentation.

**sphinx.ext.napoleon**

The sphinx extension `sphinx.ext.napoleon` allows you to define different sections in docstrings, including:

- Attributes
- Example
- Keyword Arguments
- Methods
- Parameters
- Warning
- Yield

There are two styles of docstrings in `sphinx.ext.napoleon`:

- Google
- NumPy

The main differences are that Google uses indentations and NumPy uses underscores:

Google:
def func(arg1, arg2):
    """Summary line.

    Extended description of function.

    Args:
    arg1 (int): Description of arg1
    arg2 (str): Description of arg2

    Returns:
    bool: Description of return value
    """
    return True

NumPy:

def func(arg1, arg2):
    """Summary line.

    Extended description of function.

    Parameters
    ----------
    arg1 : int
        Description of arg1
    arg2 : str
        Description of arg2

    Returns
    -------
    bool
        Description of return value
    """
    return True

You can find the detailed configuration options in sphinxcontrib.napoleon.Config.

Meta information

.. sectionauthor:: Veit Schiele <veit@cusy.io>
.. codeauthor:: Veit Schiele <veit@cusy.io>

Code block

.. code-block:: python
    :emphasize-lines: 3,5

def some_function():
    interesting = False
    print 'This line is highlighted.'
    print 'This one is not...'
    print '...but this one is.'
```python
def some_function():
    interesting = False
    print 'This line is highlighted.'
    print 'This one is not...'
    print '...but this one is.'
```

See also

.. seealso::
   `Sphinx Directives

See also:

Sphinx Directives

Glossary

.. glossary::

   environment
   A structure where information about all documents under the root is saved, and used for cross-referencing. The environment is pickled after the parsing stage, so that successive runs only need to read and parse new and changed documents.

   source directory
   The directory which, including its subdirectories, contains all source files for one Sphinx project.

8.4.3 Intersphinx

`sphinx.ext.intersphinx` allows the linking of other project documentation.

Configuration

In `source/conf.py` Intersphinx must be indicated as an extension:

```python
extensions = [
    ...
    'sphinx.ext.intersphinx',
]
```

External Sphinx documentation can then be specified, e.g. with:
intersphinx_mapping = {
    'python': ('https://docs.python.org/3', None),
    'bokeh': ('https://bokeh.pydata.org/en/latest/', None)
}

However, alternative files can also be specified for an inventory, for example:

intersphinx_mapping = {
    'python': ('https://docs.python.org/3', None, 'python-inv.txt'),
    ...
}

**Determine link targets**

To determine the links available in an inventory, you can enter the following, for example:

```
$ python -m sphinx.ext.intersphinx https://docs.python.org/3/objects.inv
```

c:func:`PyAnySet_Check`

c:func:`PyAnySet_CheckExact`

c:func:`PyArg_Parse`

...  

**Create a link**

In order to link to `https://docs.python.org/3/c-api/arg.html#c.PyArg_Parse`, one of the following variants can be specified:

`PyArg_Parse()`

:c:func:`PyArg_Parse`

`PyArg_Parse()`

:c:func:`PyArg_Parse`

**Parsing arguments**

:c:func:`Parsing arguments <PyArg_Parse>`

**Custom links**

You can also create your own `intersphinx` assignments, e.g. if `objects.inv` in `Beautiful Soup` has errors.

The error can be corrected with:

1. Installation of `sphobjinv`:

```
$ pipenv install sphobjinv
```

2. Then the original file can be downloaded with:
$ cd docs/build/
$ mkdir _intersphinx
$ !$
$ curl -O https://www.crummy.com/software/BeautifulSoup/bs4/doc/objects.inv
$ mv objects.inv bs4_objects.inv

3. Change the Sphinx configuration docs/source/conf.py:

```python
intersphinx_mapping = {
    'bs4': ('https://www.crummy.com/software/BeautifulSoup/bs4/doc/', "intersphinx/bs4_objects.inv")
}
```

4. Convert to a text file:

```
$ pipenv run sphobjinv convert plain bs4_objects.inv bs4_objects.txt
```

5. Editing the text file
e.g.:

<table>
<thead>
<tr>
<th>bs4.BeautifulSoup</th>
<th>py:class 1 index.html#beautifulsoup</th>
</tr>
</thead>
<tbody>
<tr>
<td>bs4.BeautifulSoup.get_text</td>
<td>py:method 1 index.html#get-text</td>
</tr>
<tr>
<td>bs4.element.Tag</td>
<td>py:class 1 index.html#tag</td>
</tr>
</tbody>
</table>

These entries can then be referenced in a Sphinx documentation with:

- :class:`bs4.BeautifulSoup`
- :meth:`bs4.BeautifulSoup.get_text`
- :class:`bs4.element.Tag`

See also:

- `Sphinx objects.inv v2 Syntax`

6. Create a new objects.inv file:

```
$ pipenv run sphobjinv convert zlib bs4_objects.txt bs4_objects.txt
```

7. Create Sphinx documentation:

```
$ pipenv run sphinx-build -ab html source/ build/
```
Add roles

If you get an error message that a certain text role is unknown, e.g.

```python
def setup(app):
    # from sphinx.ext.autodoc import cut_lines
    # app.connect('autodoc-process-docstring', cut_lines(4, what=['module']))
    app.add_object_type(
        "confval",
        "confval",
        objname="configuration value",
        indextemplate="pair: %s; configuration value",
    )
```

so you can add them in the `conf.py`:

---

### 8.4.4 Extensions

#### Built-in extensions

- `sphinx.ext.autodoc` Integrate documentation from docstrings
- `sphinx.ext.autosummary` generates summaries of functions, methods and attributes from docstrings
- `sphinx.ext.autosectionlabel` references section using the title
- `sphinx.ext.graphviz` Rendering of Graphviz graphs
- `sphinx.ext.ifconfig` includes content only under certain conditions
- `sphinx.ext.intersphinx` allows the linking of other project documentation
- `sphinx.ext.mathjax` Rendering via JavaScript
- `sphinx.ext.napoleon` Support for NumPy and Google style docstrings
- `sphinx.ext.todo` Support for ToDo items
- `sphinx.ext.viewcode` adds links to the source code of the Sphinx documentation

See also:

You can get a complete overview at [Sphinx Extensions](https://sphinx.readthedocs.io/en/stable/)

#### Third-party extensions

- `nbsphinx` Jupyter Notebooks in Sphinx
- `jupyter-sphinx` allows rendering of Jupyter interactive widgets in Sphinx, see also
  
  Embedding Widgets in the Sphinx HTML Documentation
- `numpydoc` NumPy’s Sphinx extension
- `Releases` writes a changelog file
- `sphinxcontrib-napoleon` Napoleon is a pre-processor for parsing NumPy- and Google-style docstrings
- `Sphinx-autodoc-typehints` Type hints support for the Sphinx autodoc extension
sphinx-git  git-Changelog for Sphinx
sphinx-intl  Sphinx extension for translations
sphinx-autobuild  monitors a Sphinx repository and creates new documentation as soon as changes are made
Sphinxcontrib-mermaid  allows you to embed Mermaid graphics in your documents.

Own Extensions

Local extensions in a project should be specified relative to the documentation. The appropriate path is specified in the Sphinx configuration docs/conf.py. If your extension is in the directory exts in the file foo.py, then the conf.py should look like this:

```python
import sys
import os
sys.path.insert(0, os.path.abspath('exts'))

extensions = [
    'foo',
    ...
]
```

8.5 Licensing

In order for others to use your software, it should be given a license that describes the terms of use. Otherwise, it should mostly be protected by copyright. Authors are those who originally contributed to the software. If software is to be relicensed, the consent of all authors is required.

**Note:** This does not constitute legal advice. If in doubt, contact a lawyer or the legal department of your company.

8.5.1 Proprietary software licenses

Proprietary software licenses are rarely standardised; they can be commercial, shareware, or freeware.

8.5.2 Free and open source software licenses

They are defined by the Free Software Foundation (FSF) and the Open Source Initiative (OSI). A distinction can essentially be made between copyleft, permissive and public domain licenses.
**Copyleft licenses**

Copyleft licenses oblige the licensees to place any processing of the software under the license of the original work. This is to prevent usage restrictions of the software. The best known copyleft license is the GNU General Public License (GPL). The copyleft of the GPL is seen as very strong, while that of the Mozilla Public License is seen as very weak. Since the licensors are not bound by their own copyleft, they can also publish new versions under a proprietary license or allow third parties to do so (multiple licensing).

Copyleft licenses can quickly lead to incompatibilities with free licenses without copyleft. For example, the 3 Clause BSD license is incompatible with the GPL.

**Permissive open source licenses**

Permissive open source licenses allow broader reuse than copyleft licenses. Derivatives and copies of the source code can be distributed under conditions that have fundamentally different properties than those of the original license. The best known examples of such licenses are the MIT license and the BSD license.

**Public domain licenses**

In the case of public domain licenses, copyrights are transferred to the general public. The Creative Commons Zero license was created to identify the release of the greatest possible usage rights.

### 8.5.3 Choosing a suitable license

Overviews of possible licenses can be found in the SPDX License List or OSI Open Source Licenses by Category. The website Choose an open source license supports you in choosing a suitable license.

### 8.5.4 GitHub

On GitHub you can have an open source license created in your repository.

1. Go to the main page of your repository.
2. Click on *Create new file* and then enter LICENSE or LICENSE.md as the file name.
3. Then you can click on *Choose a license template*.
4. Now you can select the open source license that is suitable for your repository.
5. You will now be asked for additional information if the selected license requires this.
6. After you have given a commit message, for example Add  license, you can click on *Commit new file*.

If you’ve already added a /LICENSE file to your repository, GitHub uses licensee to compare the file with a short list of open source licenses. If GitHub can’t detect your repository’s license, it might contain multiple licenses or be too complex. Then consider whether you can simplify the license, for example by outsourcing complexity to the /README file.

Conversely, you can also search for repositories with specific licenses or license families on GitHub. You can get an overview of the license keywords in Searching GitHub by license type.

Finally, you can have Shields.io generate a license badge for you, which you can include in your README file, for example
8.5.5 Standard format for licensing

We recommend using `SPDX-FileCopyrightText: [year] [copyright holder]`. Usually the information should include the entire software product, but you can also exclude elements.

8.5.6 Check conformity

REUSE was initiated by the Free Software Foundation Europe (FSFE) to facilitate the licensing of free software projects. The REUSE tool checks licenses and supports you in compliance with the license. With the REUSE API you can also generate a dynamic compliance badge:

![REUSE compliant]

**CIR workflow**

You can easily integrate REUSE into your continuous integration workflow, e.g. for GitLab in the `.gitlab-ci.yml` file with:

```yaml
reuse:
  image:
    name: fsfe/reuse:latest
  entrypoint: [""
  script:
    - reuse lint
```

**Alternatives**

- **SPDX** SPDX defines a standardised method for exchanging copyright and license information between projects and people
- **ClearlyDefined** It collects and displays information about the licensing and copyright situation of a software project
- **OpenChain** It recommends REUSE as a component to improve the clarity of the licensing and copyright situation, but has more stringent requirements to achieve full compliance.
- **FOSSology** Free software compliance toolkit that stores information in a database with license, copyright, and export scanners

See also:

- A Quick Guide to Software Licensing for the Scientist-Programmer
- Producing Open Source Software by Karl Fogel


8.6 Citing

Today software and data are integral parts of scientific research. Software is used to create, process and analyse research data and to model and simulate complex processes. Despite their increasing importance in research, it’s little known how they can be embedded in the scientific recognition and reputation systems. Quotations are an essential option in these systems, but few researchers know how software and data could be cited.

Unfortunately, there are no recognised guidelines for software and data authorship. In addition to the programmers role, other roles such as software architects, technical writers and maintainers can also be defined.

See also:

- ICMJE: Defining the Role of Authors and Contributors
- Bot Recognize All Contributors

8.6.1 Cite data

DataCite Metadata Schema

The DataCite Metadata Working Group published the DataCite Metadata Schema Documentation for the publication and citation of research data in 2019: DataCite Metadata Schema 4.3 together with a XSD (XML Schema Definition): metadata.xsd.

A simple datacite example can look like this:

```xml
  <identifier identifierType="DOI">10.5072/D3P26Q35R-Test</identifier>
  <creators>
    <creator>
      <creatorName nameType="Personal">Fosmire, Michael</creatorName>
      <givenName>Michael</givenName>
      <familyName>Fosmire</familyName>
    </creator>
    <creator>
      <creatorName nameType="Personal">Wertz, Ruth</creatorName>
      <givenName>Ruth</givenName>
      <familyName>Wertz</familyName>
    </creator>
    <creator>
      <creatorName nameType="Personal">Purzer, Senay</creatorName>
      <givenName>Senay</givenName>
      <familyName>Purzer</familyName>
    </creator>
  </creators>
  <titles>
    <title xml:lang="en">Critical Engineering Literacy Test (CELT)</title>
  </titles>
  <publisher xml:lang="en">Purdue University Research Repository (PURR)</publisher>
  <publicationYear>2013</publicationYear>
</resource>
```

(continues on next page)
We developed an instrument, Critical Engineering Literacy Test (CELT), which is a multiple choice instrument designed to measure undergraduate students' scientific and information literacy skills. It requires students to first read a technical memo and, based on the memo's arguments, answer eight multiple-choice and six open-ended response questions. We collected data from 143 first-year engineering students and conducted an item analysis. The KR-20 reliability of the instrument was .39. Item difficulties ranged between .17 to .83. The results indicate low reliability index but acceptable levels of item difficulties and item discrimination indices. Students were most challenged when answering items measuring scientific and mathematical literacy (i.e., identifying incorrect information).

W3C-PROV

The PROV document family of the W3C working group defines various aspects that are necessary to be able to exchange provenance information interoperably.

See also:

- Provenance: An Introduction to PROV by Luc Moreau and Paul Groth
- Provenance storage and distribution
- ProvStore’s API documentation

Python prov

With prov, a Python3 library is available that supports the import and export of the PROV data model into the following serialisation formats:

- PROV-O (RDF)
- PROV-XML
- PROV-JSON

In addition, PROV documents can be created with NetworkX MultiDiGraph and vice versa. Finally, PROV documents can also be generated as graphs in PDF, PNG and SVG formats.

See also:
Git2PROV

Git2PROV generates PROV data from the information in a Git repository.

On the command line, the conversion can be easily executed with:

```
$ git2prov git_url [serialization]
```

For example:

```
$ git2prov git@github.com:veit/jupyter-tutorial.git PROV-JSON
```

In total, the following serialisation formats are available:

- PROV-N
- PROV-JSON
- PROV-O
- PROV-XML

Alternatively, Git2PROV also provides a web server with:

```
$ git2prov-server [port]
```

See also:

- Git2PROV: Exposing Version Control System Content as W3C PROV
- GitHub-Repository

### 8.6.2 Cite software

James Howison and Julia Bullard listed the following examples in descending reputations in their 2016 article Software in the scientific literature:

1. citing publications that describe the respective software
2. citing operating instructions
3. citing the software project website
4. link to a software project website
5. mention the software name

Nevertheless, the situation remains unsatisfactory for software authors, especially if they differ from the authors of the software description. Conversely, research software is unfortunately not always well suited to be cited. Colleagues will hardly be able to cite your software directly if you send them the software as an attachment to an email. Even a download link is not really useful here. But how can authors make their software citable?

Digital object identifier (DOI) are commonly used in science for citations. Zenodo enables software to be archived and a DOI to be provided for it. In the following I will show which steps are required on the example of the Jupyter tutorial:

1. If you haven’t already, create an account on Zenodo, preferably with GitHub.
2. Now select the repository that you want to archive:
3. Check whether Zenodo has created a webhook in your repository for the `Releases` event:

4. Create a new release:

5. Check that the DOI (Digital object identifier) was created correctly:

6. Finally, you can include a badge in the readme of your software, e.g.:

   **Markdown:**

   ```markdown
   ![DOI](https://zenodo.org/badge/199994535.svg)
   ```
Chapter 8. Create a product
The FORCE11 working group has published a paper in which the principles of scientific software citation are presented: FORCE11 Software Citation Working Group by Arfon Smith, Daniel Katz and Kyle Niemeyer 2016. Two projects are currently emerging for structured metadata:

**CodeMeta** Exchange scheme for general software metadata and reference implementation for JSON for Linking Data (JSON-LD).

A *codemeta.json* file is expected in the root directory of the software repository. The file can look like this:

```json
{
   "@context": "https://doi.org/10.5063/schema/codemeta-2.0",
   "@type": "SoftwareSourceCode",
   "author": [{
      "@type": "Person",
      "givenName": "Stephan",
      "familyName": "Druskat",
      "@id": "http://orcid.org/0000-0003-4925-7248"
   }],
   "name": "My Research Tool",
   "softwareVersion": "2.0",
   "identifier": "https://doi.org/10.5281/zenodo.1234",
   "datePublished": "2017-12-18",
   "codeRepository": "https://github.com/research-software/my-research-tool"
}
```

See also:
- CodeMeta generator
- Codemeta Terms
- GitHub Repository

**Citation File Format** Scheme for software citation metadata in machine-readable YAML format

A file `CITATION.cff` should be stored in the root directory of the software repository. The content of the file can look like this:

```yaml
cff-version: "1.1.0"
message: "If you use this tutorial, please cite it as below."
authors:
   - family-names: Schiele
given-names: Veit
   orcid: "https://orcid.org/https://orcid.org/0000-0002-2448-8958"
identifiers:
   - type: doi
     value: "10.5281/zenodo.4147287"
keywords:
   - "data-science"
   - jupyter
```

(continues on next page)
You can easily adapt the example above to create your own CITATION.cff file or use the cffinit website.

There are also some tools for processing CITATION.cff files:

- cff-converter-python converts CITATION.cff files to BibTeX, RIS, CodeMeta and other file formats
- doi2cff creates a CITATION.cff file from a Zenodo DOI

GitHub also offers a service to copy the information from CITATION.cff files in APA and BibTex format.

See also:

- GitHub Docs: About CITATION files

When registering a DOI via Zenodo the CITATION.cff file in the GitHub repository is also be used. Also Zotero interprets the CFF file in GitHub repositories; however, Zotero can take meta-information of the repository, such as company, programming language etc., even without a CFF file.

You should provide a Persistent Identifier (PID) to ensure the long-term availability of your software. Both the Zenodo and figshare repositories accept source code including binary files and provide DOIs for this. And citation information for software can also be called up with CiteAs.

See also:

- Should I cite?
- How to cite software “correctly”
- Daniel S. Katz: Compact identifiers for software: The last missing link in user-oriented software citation?
- Neil Chue Hong: How to cite software: current best practice
- Recognizing the value of software: a software citation guide
- Stephan Druskat, Radovan Bast, Neil Chue Hong, Alexander Konovalov, Andrew Rowley, Raniere Silva: A standard format for CITATION files
- Module-5-Open-Research-Software-and-Open-Source
- Software Heritage: Save and reference research software
- Mining software metadata for 80 M projects and even more
- Extensions to schema.org to support structured, semantic, and executable documents
8.6. Citing

Cite this repository

If you use this software in your work, please cite it using the following metadata. Learn more

APA
Schiele V. (2020). Jupyter tutorial (version

BibTeX

View citation file
8.6.3 Software journals

General

- IEEE Computer Society Digital Library
- Wiley Online Library
- Journal of Open Source Software
- Journal of Open Research Software (JORS)
- Journal of Software: Practice and Experience
- Nature Toolbox
- Research Ideas and Outcomes (RIO)
- SIAM Journal on Scientific Computing (SISC) Software section
- SoftwareX

Image processing

- Image Processing On Line
- Insight Journal

Biology

- American Journal of Human Genetics
- Artificial Life
- Psychonomic Society: Behaviour Research Methods
- Oxford Academic: Bioinformatics
- Bioinformatics and Biology Insights
- Biophysical Journal
- BMC Bioinformatics
- BMC Systems Biology
- Bone
- Computer Methods and Programs in Biomedicine
- Current Protocols in Bioinformatics
- Database: The Journal of Biological Databases and Curation
- Ecography
- eLife
- Epidemiology
- Evolutionary Bioinformatics
- F1000 Research
- Frontiers in Neuroinformatics
• Gigascience
• Methods in Ecology and Evolution
• Nature Methods
• Neurocomputing
• Neuroinformatics
• Nucleic Acids Research
• PeerJ
• PLoS Computational Biology: Software collection
• PLoS ONE
• Trends in Parasitology

Chemistry

• International Journal of Quantum Chemistry
• Journal of Applied Crystallography
• Journal of Chemical Theory and Computation
• Journal of Chemical Information and Modelling
• Journal of Cheminformatics
• Journal of Computational Chemistry
• Molecular Simulation
• Wiley Interdisciplinary Reviews: Computational Molecular Science

Human and social sciences

• Digital Humanities Quarterly
• Journal of Artificial Societies and Social Simulation
• Journal of Economic Dynamics and Control

Engineering

• Advances in Engineering Software
• Coastal Engineering
• Renewable Energy

8.6. Citing
Computer science, mathematics and statistics

- ACM Transactions on Mathematical Software
- The Archive of Numerical Software
- Future Generation Computer Systems
- Journal of Multiscale Modelling and Simulation
- Journal of Parallel and Distributed Computing
- Journal of Software for Algebra and Geometry
- Journal of Statistical Software
- Knowledge-Based Systems
- LMS Journal of Computation and Mathematics
- The Mathematica Journal
- Mathematical Programming Computation
- Numerical Algorithms
- PeerJ Computer Science
- The R Journal
- Science of Computer Programming
- The Stata Journal

Physics and Earth Sciences

- AAS: The Astronomy Journal
- AAS: The Astrophysical Journal
- Astronomy and Computing
- Communications in Computational Physics
- Computational Astrophysics and Cosmology
- Computer Physics Communications
- Computers and Geosciences
- Computing and Software for Big Science
- Environmental Modelling & Software
- Geoscientific Model Development
8.7 Reproduce environments

Run the notebook in a dedicated environment (e.g. with Pipenv, devpi and Spack). Save the file with the specifications, e.g. Pipfile, Pipfile.lock, package-lock.json etc. In this way, you and others can reproduce your calculations and deployment into the production environment is simplified.

8.7.1 Spack

Modeling and simulation environments are very heterogeneous. Spack therefore supports many different production environments:

- 7 different compilers: Intel, GCC, Clang, PGI, …
- Resolving dependencies
- Resolving different versions of dependencies

Previous systems

They usually do not offer any support for combinatorial versioning.

- Traditional binary package managers like RPM, yum, APT, yast, etc.
  - are designed to manage a single software stack
  - install one version of a package
  - usually problem-free upgrades to a stable, well-tested stack
- Port systems
  - BSD Ports, portage, NixOS, Macports, Homebrew, etc.
  - mostly little support for builds that are parameterised by compilers or dependent versions
- Virtual machines and Linux containers
  - Containers allow the creation of different environments for different applications
  - However, they do not solve the build problem for the image
  - Performance, security and upgrades become very complex with many different builds.

Spack installation

Requirements

- Python 2 or Python 3
- C/C++ compiler
- git and curl

For Linux:

```
$ apt install curl git environment-modules
```

… or for macOS:
Jupyter Tutorial, Release 0.8.0

```
$ brew install curl git modules
```

Then the shell is configured by entering for example the following in the Bash configuration:

```
$ source /usr/local/opt/modules/init/bash
```

- gnupg2 for the gpg subcommand

### Installation

```
$ git clone https://github.com/spack/spack.git
Cloning into 'spack'...
...`

### Configure the shell

1. To configure the bash environment, the following is entered in the `~/.bashrc`:

   ```bash
   export SPACK_ROOT=~/spack
   . $SPACK_ROOT/share/spack/setup-env.sh
   ```

2. The changed configuration is read with

   ```bash
   $ source ~/.bashrc
   ```

### Checking the installation

```
$ spack spec python
Input spec
-------------------------------
python
Concretized
-------------------------------
python@3.7.6%gcc@7.4.0+bz2+ctypes+dbm-debug+libxml2+lzma-nis-
  ...optimizations-pic+pyexpat+pythoncmd+readline+shared+sqlite3+ssl-tix.tkinter~ucs4-
  ...uuid+zlib arch=linux-ubuntu18.04-sandybridge
  ^bzip2@1.0.8%gcc@7.4.0+shared arch=linux-ubuntu18.04-sandybridge
  ^diffutils@3.7%gcc@7.4.0 arch=linux-ubuntu18.04-sandybridge
  ^libiconv@1.16%gcc@7.4.0 arch=linux-ubuntu18.04-sandybridge
  ^expat@2.2.9%gcc@7.4.0+libbsd arch=linux-ubuntu18.04-sandybridge
  ^libbsd@0.10.0%gcc@7.4.0 arch=linux-ubuntu18.04-sandybridge
  ^gdbm@0.11.1%gcc@7.4.0 arch=linux-ubuntu18.04-sandybridge
  ^readline@8.0%gcc@7.4.0 arch=linux-ubuntu18.04-sandybridge
  ^ncurses@6.1%gcc@7.4.0~symlinks+termlib arch=linux-ubuntu18.04-sandybridge
  ^pkgconf@1.6.3%gcc@7.4.0 arch=linux-ubuntu18.04-sandybridge
  ^gettext@0.20.1%gcc@7.4.0+bzzip2+curses+git-libunistring+libxml2+tar+xz arch=linux-
  ...ubuntu18.04-sandybridge
  ^libxml2@2.9.9%gcc@7.4.0-python arch=linux-ubuntu18.04-sandybridge
```
(continues on next page)
Compiler configuration

$ spack compilers
==Available compilers
-- clang mojave-x86_64
clang@10.0.1-apple

GPG signing

Spack supports the signing and verification of packages with GPG keys. A separate key ring is used for Spack, why no keys are available from users’ home directories.

When Spack is first installed, this key ring will be empty. The keys stored in /var/spack/gpg are the standard keys for a Spack installation. These keys are imported by spack gpg init. This will import the standard keys into the keyring as trusted keys.

Trust keys

Additional keys can be added to the key ring using spack gpg trust <keyfile>. Once a key is trusted, packages signed by the owner of that key can be installed.

Create a key

You can also create your own keys to be able to sign your own packages with

$ spack gpg export <location> [key...]

List keys

The keys available in the keyring can be listed with

$ spack gpg list

8.7. Reproduce environments
Remove a key

Keys can be removed with

```
$ spack gpg untrust <keyid>
```

Key IDs can be email addresses, names or fingerprints.

Combinatorial builds

Environment modules

```
$ module avail
--------------------------- /opt/modules/modulefiles ----------------------------
acml-gnu/4.4 intel/12.0 mvapich2-pgi-ofa/1.7
acml-gnu_mp/4.4 intel/13.0 mvapich2-pgi-psm/1.7
acml-intel/4.4 intel/14.0(default) mvapich2-pgi-shmem/1.7...
$ module load intel/13.0
$ module load mvapich2-pgi-shmem/1.7
```

- Pros
  - replace different versions dynamically in the shell
  - abstract a lot from the complexity of the environment
- Cons
  - Users need to keep in mind which versions of the build were made
  - It’s easy to load the wrong module and cause a build to fail

Dependency DAG

![Dependency DAG](image)
Installation layout

```
$ tree /Users/veit/spack/opt/spack/
/Users/veit/spack/opt/spack/
  darwin-mojave-x86_64
    clang-10.0.1-apple
      autoconf-2.69-ymadj7a7gg52r76payi7jd7qu7qcuasp
      autoheader
      bin
      autoconf
      autoheader
... 
```

- Each unique dependency graph is given a unique configuration
- Each configuration is installed in a unique directory
  - Configurations of the same package coexist
- The hash value of a directed acyclic graph is appended
- Installed packages automatically find their dependencies
  - Spack embeds RPATH in binary files
  - There is no need to use modules or to set the LD_LIBRARY_PATH

`spack list` shows the available packages:

```
$ spack list
===> 3250 packages.
abinit    py-fiona
abyss     py-fiscalyear
accfft     py-flake8
... 
```

Spack provides a `spec` syntax for describing custom DAGs:

- without restrictions
  
  ```
  $ spack install mpileaks
  ```

- @: custom version
  
  ```
  $ spack install mpileaks@3.3
  ```

- %: custom compiler
  
  ```
  $ spack install mpileaks@3.3 %gcc@4.7.3
  ```

- +/-: Build option
  
  ```
  $ spack install mpileaks@3.3 %gcc@4.7.3 +threads
  ```

- =: Cross compile
  
  ```
  $ spack install mpileaks@3.3 =bgq
  ```

- ^: Version of dependencies

8.7. Reproduce environments
$ spack install mpileaks %intel@12.1 ^libelf@0.8.12

• Spack ensures a configuration of each library per DAG
  – ensures the consistency of the Application Binary Interface (ABI)
  – The user does not need to know the DAG structure, just the names of the dependent libraries
• Spack can ensure that builds use the same compiler
• Different compilers can also be specified for different libraries of a DAG
• Spack can also provide ABI-incompatible, versioned interfaces such as the Message Passing Interface (MPI)
• For example, you can create mpi in different ways:

  $ spack install mpileaks ^mvapich@1.9
  $ spack install mpileaks ^openmpi@1.4

• Alternatively, Spack can also choose the right build himself if only the MPI 2 interface is implemented:

  $ spack install mpileaks ^mpi@2

• Spack packages are simple Python scripts:

```python
from spack import *

class Dyninst(Package):
    """API for dynamic binary instrumentation.""
    homepage = "https://paradyn.org"

    version('8.2.1', 'abf60b7faabe7a2e', url="http://www.paradyn.org/release8.2/\n    →DyninstAPI-8.2.1.tgz")
    version('8.1.2', 'bf03b33375afa66f', url="http://www.paradyn.org/release8.1.2/\n    →DyninstAPI-8.1.2.tgz")
    version('8.1.1', 'd1a04e995b7aa709', url="http://www.paradyn.org/release8.1/\n    →DyninstAPI-8.1.1.tgz")

    depends_on("libelf")
    depends_on("libdwarf")
    depends_on("boost@1.42:")

    def install(self, spec, prefix):
        libelf = spec['libelf'].prefix
        libdwarf = spec['libdwarf'].prefix

        with working_dir('spack-build', create=True):
            cmake('...',
                  '-DBOOST_INCLUDE_DIR=%s' % spec['boost'].prefix.include,
                  '-DBOOST_LIBRARY_DIR=%s' % spec['boost'].prefix.lib,
                  '-DBOOST_NO_SYSTEM_PATHS=TRUE'
                  *std_cmake_args)
            make()
            make("install")

        @when('@:8.1')
```

(continues on next page)
def install(self, spec, prefix):
    configure("--prefix=" + prefix)
    make()
    make("install")

• Dependencies in Spack can be optional:
  – You can define named variants, e.g. in ~/spack/var/spack/repos/builtin/packages/vim/package.py:

    class Vim(AutotoolsPackage):
        ...
        variant('python', default=False, description="build with Python")
        depends_on('python', when='+python')

        variant('ruby', default=False, description="build with Ruby")
        depends_on('ruby', when='+ruby')

  – … and use to install:

    $ spack install vim +python
    $ spack install vim –python

  – Depending on other conditions, dependencies can optionally apply, e.g. gcc dependency on mpc from
    version 4.5:

    depends_on("mpc", when="@4.5:")

• DAGs are not always complete before they are specified. Concretisations fill in the missing configuration details
  if you do not name them explicitly:

  1. Normalisation

    $ spack install mpileaks ^callpath@1.0+debug ^libelf@0.8.11

  2. Specification

    The detailed origin is saved with the installed package in spec.yaml:

    spec:
    - mpileaks:
        arch: linux-x86_64
        compiler:
            name: gcc
            version: 4.9.2
        dependencies:
            adept-utils: kszrtpzbac3ss2ixcjckorlaybnptp4
            callpath: bah5f4h4d2n47mgycej2mtrrvvxy77
            mpich: aa4ar6ifj23yiqmdabeakpejcli72t3
            hash: 33hjihxi7p6gyzn5ptgyles7sghyprujh
            variants: {}
            version: '1.0'
        - adept-utils:
            arch: linux-x86_64

(continues on next page)
1. If unspecified, values based on the user settings are selected during the specification.
2. During the concretisation, new dependencies are added taking the constraints into account.
3. With the current algorithm, it is not possible to trace why a decision was made.
4. In the future there should be a full constraint solver.

Benefits of the build automation

- Spack makes it easy for teams to share their code
  - Recipes for common libraries
  - reduce the effort for reproducible builds
  - making it easier to share builds.
- Patches allow bug fixes to be provided quickly
  - Application developers who use a library often do not have write access to their repositories.
  - Library developers may not be able to fix problems as quickly as desired.
  - With Spack, application developers can quickly make corrections and undo changes.
- Python allows rapid adoption by development teams.
  - Many application developers are already familiar with Python.
  - The yaml syntax of the specs are expressive.
Use case 1: managing combinatorial installations

Display all installed configurations

$ spack find

=> 103 installed packages.
-- linux-x86_64 / gcc@4.8.2 -----------------------------
gdk-pixbuf@2.31.2  libpng@1.6.16  otf2@1.4  qhull@1.0
adept-utils@1.0.1  boost@1.55.0  cmake@5.6-special  libdwarf@20130729  mpich@3.0.4
-- linux-x86_64 / intel@14.0.2 -----------------------------
hwloc@1.9  mpich@3.0.4  starpu@1.1.4
-- linux-x86_64 / intel@15.0.0 -----------------------------
adept-utils@1.0.1  boost@1.55.0  libdwarf@20130729  libelf@0.8.13  mpich@3.0.4
-- linux-x86_64 / intel@15.0.1 -----------------------------
adept-utils@1.0.1  callpath@1.0.2  libdwarf@20130729  mpich@3.0.4
boost@1.55.0  hwloc@1.9  libelf@0.8.13  starpu@1.1.4

- spack find shows all installed configurations
- There can also be different versions of the same package
- Packages are differentiated between architecture and compiler
- Spack also generates modulefiles, but these do not have to be used

Spack syntax to restrict the requests

$ spack find mpich

=> 5 installed packages.
-- linux-x86_64 / gcc@4.4.7 -----------------------------
mpich@3.0.4
-- linux-x86_64 / gcc@4.8.2 -----------------------------
mpich@3.0.4
-- linux-x86_64 / intel@14.0.2 -----------------------------
mpich@3.0.4

$ spack find libelf %intel

-- linux-x86_64 / intel@15.0.0 ----
libelf@0.8.13
-- linux-x86_64 / intel@15.0.1 ----
libelf@0.8.13

$ spack find libelf %intel@15.0.1

-- linux-x86_64 / intel@15.0.1 ----
libelf@0.8.13
Spack syntax for displaying the dependencies

```
$ spack find callpath
=> 2 installed packages.
-- linux-x86_64 / clang@3.4        -- linux-x86_64 / gcc@4.9.2
  callpath@1.0.2                    callpath@1.0.2

$ spack find -dl callpath
=> 2 installed packages.
-- linux-x86_64 / clang@3.4        -- linux-x86_64 / gcc@4.9.2
  xv2clz2                          udltshs callpath@1.0.2
  ckjazss  ^adept-utils@1.0.1       rfsu7fb  ^adept-utils@1.0.1
  3ws43m4  ^boost@1.59.0            ybet64y  ^boost@1.55.0
  ft7znm6  ^mpich@3.1.4             aa4ar6i  ^mpich@3.1.4
  qnuuet3  ^dyninst@0.2.1           tmmnge5  ^dyninst@0.2.1
  3ws43m4  ^boost@1.59.0            ybet64y  ^boost@1.55.0
  g65rdud  ^libdwarf@20130729       g2mxr12  ^libdwarf@20130729
  cj5p5fk  ^libelf@0.8.13           ynpai3j  ^libelf@0.8.13
  cj5p5fk  ^libelf@0.8.13           ynpai3j  ^libelf@0.8.13
  g65rdud  ^libdwarf@20130729       g2mxr12  ^libdwarf@20130729
  cj5p5fk  ^libelf@0.8.13           ynpai3j  ^libelf@0.8.13
  cj5p5fk  ^libelf@0.8.13           ynpai3j  ^libelf@0.8.13
  ft7znm6  ^mpich@3.1.4             aa4ar6i  ^mpich@3.1.4
```

Use case 2: Python and other interpreted languages

```
$ spack install python@2.7.10
=> Building python.
=> Successfully installed python.
  Fetch: 5.01s. Build: 97.16s. Total: 103.17s.
++ /srv/jupyterhub/spack/opt/spack/linux-x86_64/gcc-4.9.2/python-2.7.10-y2zr767

$ spack extensions python@2.7.10
=> python@2.7.10%gcc@4.9.2=linux-x86_64-y2zr767
   => 49 extensions:
   geos             py-h5py                    py-numpy        py-pytool     py-setuptools
   libxml2          py-ipython                py-pandas       py-pyparsing  py-shiboken
   py-basemap       py-libxml2                py-pexpect      py-pyqt       py-sip
   py-biopython     py-lockfile              py-pil          py-pyside     py-six
   py-cffi          py-mako                   py-pmw          py-python-daemon py-sphinx
   py-cython        py-matplotlib             py-pychecker    py-pytz       py-sympy
   py-dateutil      py-mock                   py-pycparser    py-rpy2       py-virtualenv
   py-epydoc        py-mpi4py                 py-pyeltools    py-scientificpython py-yapf
   py-genders       py-nx                     py-pygments     py-sckit-learn thrift
   py-gnuplot       py-nose                   py-pylint       py-scipy
   => 3 installed:
   -- linux-x86_64 / gcc@4.9.2
   py-nose@1.3.6    py-numpy@1.9.2          py-setuptools@18.1
   => None currently activated.

$ spack activate py-numpy
```

(continues on next page)
Future features

- Lmod (Lua based module system) integration
- Resolve external dependencies
- Custom compiler flag injection
- XML test results (JUnit)

See also:
Pull requests

Use spack

List the available packages

$ spack list
==> 3247 packages.
abinit          py-fiona
abyss           py-fiscalyear
...

or to filter for certain packages, e.g.

$ spack list numpy
==> 2 packages.
py-numpy  py-numpydoc

List the installed packages

$ spack find
==> 17 installed packages
-- darwin-mojave-x86_64 / clang@10.0.1-apple --------------
bzip2@1.0.8  libffi@3.2.1  perl@5.26.2  python@3.7.4  zlib@1.2.11
difutils@3.7  ncurses@6.1  pkgconf@1.6.1  readline@7.0
expat@2.2.5  openblas@0.3.6  py-numpy@1.16.4  sqlite@3.28.0
gdbm@1.18.1  openssl@1.0.2  py-setuptools@41.0.1  xz@5.2.4

8.7. Reproduce environments

409
spack info

$ spack info py-numpy
PythonPackage: py-numpy

Description:
NumPy is the fundamental package for scientific computing with Python. It contains among other things: a powerful N-dimensional array object, sophisticated (broadcasting) functions, tools for integrating C/C++ and Fortran code, and useful linear algebra, Fourier transform, and random number capabilities

Homepage: http://www.numpy.org/

Tags: None

Preferred version:
1.16.4 https://pypi.io/packages/source/n/numpy/numpy-1.16.4.zip

Safe versions:
1.16.4 https://pypi.io/packages/source/n/numpy/numpy-1.16.4.zip
1.16.3 https://pypi.io/packages/source/n/numpy/numpy-1.16.3.zip
1.16.2 https://pypi.io/packages/source/n/numpy/numpy-1.16.2.zip
1.16.1 https://pypi.io/packages/source/n/numpy/numpy-1.16.1.zip
1.16.0 https://pypi.io/packages/source/n/numpy/numpy-1.16.0.zip
1.15.4 https://pypi.io/packages/source/n/numpy/numpy-1.15.4.zip
1.15.3 https://pypi.io/packages/source/n/numpy/numpy-1.15.3.zip
1.15.2 https://pypi.io/packages/source/n/numpy/numpy-1.15.2.zip
1.15.1 https://pypi.io/packages/source/n/numpy/numpy-1.15.1.zip
1.15.0 https://pypi.io/packages/source/n/numpy/numpy-1.15.0.zip
1.14.5 https://pypi.io/packages/source/n/numpy/numpy-1.14.5.zip
1.13.3 https://pypi.io/packages/source/n/numpy/numpy-1.13.3.zip
1.13.1 https://pypi.io/packages/source/n/numpy/numpy-1.13.1.zip
1.13.0 https://pypi.io/packages/source/n/numpy/numpy-1.13.0.zip
1.12.1 https://pypi.io/packages/source/n/numpy/numpy-1.12.1.zip
1.12.0 https://pypi.io/packages/source/n/numpy/numpy-1.12.0.zip
1.11.3 https://pypi.io/packages/source/n/numpy/numpy-1.11.3.zip
1.11.2 https://pypi.io/packages/source/n/numpy/numpy-1.11.2.zip
1.11.1 https://pypi.io/packages/source/n/numpy/numpy-1.11.1.zip
1.11.0 https://pypi.io/packages/source/n/numpy/numpy-1.11.0.zip
1.10.4 https://pypi.io/packages/source/n/numpy/numpy-1.10.4.zip
1.9.3 https://pypi.io/packages/source/n/numpy/numpy-1.9.3.zip
1.9.2 https://pypi.io/packages/source/n/numpy/numpy-1.9.2.zip
1.9.1 https://pypi.io/packages/source/n/numpy/numpy-1.9.1.zip

(continues on next page)
### Variants:

<table>
<thead>
<tr>
<th>Name</th>
<th>Allowed values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blas</td>
<td>on</td>
<td>Build with BLAS support</td>
</tr>
<tr>
<td>lapack</td>
<td>on</td>
<td>Build with LAPACK support</td>
</tr>
</tbody>
</table>

#### Installation Phases:

- build
- install

### Build Dependencies:

- blas
- lapack
- py-setuptools
- python

### Link Dependencies:

- blas
- lapack
- python

### Run Dependencies:

- python

### Virtual Packages:

- None

---

#### `spack version`

`spack version` shows the available versions, e.g.

```bash
$ spack versions python

=> Safe versions (already checksummed):
  3.7.4 3.7.0 3.6.5 3.6.1 3.5.1 3.3.6 2.7.15 2.7.11
  3.7.3 3.6.8 3.6.4 3.6.0 3.5.0 3.2.6 2.7.14 2.7.10
  3.7.2 3.6.7 3.6.3 3.5.7 3.4.10 3.1.5 2.7.13 2.7.9
  3.7.1 3.6.6 3.6.2 3.5.2 3.4.3 2.7.16 2.7.12 2.7.8

=> Remote versions (not yet checksummed):
  3.8.0b2 3.6.9 3.5.7rc1 3.5.0a2 3.4.0 3.1.2 2.7 2.4.3
  3.8.0b1 3.6.8rc1 3.5.6rc1 3.5.0al 3.3.7rc1 3.1.1 2.6.9 2.4.2
  ...
```

---

#### Installation of certain packages

e.g.

```bash
$ spack install python@3.7.4
```

or to install `py-numpy` for Python 3.7.4:

```bash
$ spack install py-numpy ^python@3.7.4
```

Then the installation can be checked with
Jupyter Tutorial, Release 0.8.0

$ spack find --deps py-numpy
===> 1 installed package
-- darwin-mojave-x86_64 / clang@10.0.1-apple ---------------------
   py-numpy@1.16.4
      ^openblas@0.3.6
      ^python@3.7.4
         ^bzip2@1.0.8
         ^expat@2.2.5
         ^gdbm@1.18.1
            ^readline@7.0
            ^ncurses@6.1
            ^libffi@3.2.1
            ^openssl@1.1.1b
            ^zlib@1.2.11
            ^sqlite@3.28.0
            ^xz@5.2.4

Uninstall

$ spack uninstall py-numpy

or

$ spack uninstall --dependents py-numpy

Extensions and Python support

The Spack installation model assumes that each package lives in its own installation prefix. Modules in interpreted languages such as Python are typically installed in $prefix/lib/python-3.7/site-packages/ , e.g. /Users/veit/spack/opt/spack/darwin-mojave-x86_64/clang@10.0.1-apple/py-numpy-1.16.4-45sqnufha2yprrpx6ryelsokky65ucdy/lib/python3.7/site-packages/numpy. However, packages installed in a different prefix can also be used. Such a package is called an extension in Spack.

Suppose Python was installed with

$ spack find python
===> 1 installed package
-- darwin-mojave-x86_64 / clang@10.0.1-apple ---------------------
   python@3.7.4

so Extensions can be found with

$ spack extensions python
===> python@3.7.4%clang@10.0.1-apple+bz2+ctypes+dbm+lzma-nis-optimizations˓
      patches=210df3f28cde02a8135b58cc4168e70ab91dbf9097359d05938f1e2843875e57˓
      →+pic+pyexpat+pythoncmd+readline-shared+sqlite3+ssl-tkinter-ucs4-uuid+zlib˓
      →arch=darwin-mojave-x86_64/jqlxzxp
===> 623 extensions:
adios2                   py-munch
antlr                    py-mx

(continues on next page)
numpy can be added to the PYTHONPATH of the current shell with load:

```
$ spack load python
$ spack load py-numpy
$ python
Python 3.7.4 (default, Jul 28 2019, 20:00:06)
[Clang 10.0.1 (clang-1001.0.46.4)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy

Often, however, certain packages should be permanently available to a Python installation. Spack offers activate for this:

```
$ spack activate py-numpy

Environments, spack.yaml and spack.lock

1. Create a virtual environment:

```
$ spack env create python-374

Alternatively, it can also be saved in any other location, for example:

```
$ cd spackenvs/
$ spack env create -d python-374

2. Check the virtual environment:

```
$ spack env list

8.7. Reproduce environments
3. Activate the virtual environment:

```bash
$ spack env activate python-374
```

4. Check activation:

If you have activated an environment, you will only see what is in the current environment. That shouldn’t be anything immediately after activation:

```bash
$ spack find
===> In environment python-374
===> No root specs
===> 0 installed packages
```

And if you want to check what environment you are in, you can query this with:

```bash
$ spack env status
===> In environment python-374
```

5. Finally, you can leave the activated environment with `spack env deactivate` or briefly `despacktivate`.

```bash
$ despacktivate
$ spack env status
===> No active environment
$ spack find
===> 17 installed packages
-- darwin-mojave-x86_64 / clang@10.0.1-apple --------------------
bzip2@1.0.8  libffi@3.2.1  perl@5.26.2  python@3.7.4  zlib@1.2.11
diffutils@3.7  ncurses@6.1  pkgconf@1.6.1  readline@7.0
expat@2.2.5  openblas@0.3.6  py-numpy@1.16.4  sqlite@3.28.0
gdbm@1.18.1  openssl@1.1.1  py-setuptools@41.0.1  xz@5.2.4
```

### Install compiler

1. Installation

```bash
$ spack install gcc@9.1.0
...
[+] /srv/jupyter/spack/opt/spack/linux-ubuntu18.04-sandybridge/gcc-7.4.0/gcc-9.1.0-˓
→zaj3xkm5onfgeweaeav5kuubwmjaokmz
```

2. Configuration

In order to be able to use the new gcc compiler, it must be entered in `~/.spack/linux/compilers.yaml`:

```bash
$ spack compiler find /srv/jupyter/spack/opt/spack/linux-ubuntu18.04-sandybridge/ ˓
gcc-7.4.0/gcc-9.1.0-zaj3xkm5onfgeweaeav5kuubwmjaokmz
===> Added 1 new compiler to /srv/jupyter/.spack/linux/compilers.yaml
gcc@9.1.0
===> Compilers are defined in the following files:
    /srv/jupyter/.spack/linux/compilers.yaml
```

3. Check
Install packages

$ spack env activate python-374
$ spack install python@3.7.4
$ spack find

===> In environment python-374
===> Root specs
py-numpy python@3.7.4

===> 14 installed packages
-- linux-debian9-x86_64 / gcc@9.1.0 -----------------------------
bzip2@1.0.6 expat@2.2.5 gdbm@1.18.1 libbsd@0.9.1 libffi@3.2.1 ncurses@6.1
˓→ openblas@0.3.5 openssl@1.1.1b py-numpy@1.16.2 python@3.7.2 readline@7.0 sqlite@3.
˓→ xz@5.2.4 zlib@1.2.11

With spack cd -e python-374 you can change to this directory, for example:

$ spack cd -e python-374
$ pwd
/Users/veit/spack/var/spack/environments/python-374

There you will find the two files spack.yaml and spack.lock.

spack.yaml is the configuration file for the virtual environment. It is created in ~/spack/var/spack/environments/ when you call spack env create.

As an alternative to spack install Python-3.7.4, Numpy etc. can also be added to the specs list in spack.yaml:

specs: [gcc@9.1.0, python@3.7.4%gcc@9.1.0, py-numpy ^python@3.7.4, ...]

Finally, the virtual environment can be created with:

$ spack install

===> Concretizing python@3.7.4%gcc@9.1.0
- zd32kkg python@3.7.4%gcc@9.1.0+bl2+cytypes+dbm+lzma-nis-optimizations
- patches=210df3f28cde02a8135b58cc4168e70ab91dbf9097359d05938f1e2843875e57
- +pic+pyexpat+pythoncmd+readline-shared+sqlite3+ssl-tix-tkinter-ucs4-uuid+zlib
- arch=darwin-mojave-x86_64
[+] qeu2v43 ^bzip2@1.0.8%gcc@9.1.0+shared arch=darwin-mojave-x86_64
[+] ndtr5vr ^diffutils@3.7%gcc@9.1.0 arch=darwin-mojave-x86_64
...

===> Concretizing py-numpy ^python@3.7.4%gcc@9.1.0
- hcfve7o py-numpy@1.16.4%gcc@9.1.0+blas+lapack arch=darwin-mojave-x86_64

(continues on next page)
spack.lock With spack install the specs are concretised, written in spack.lock and installed. In contrast to spack.yaml spack.lock is written in json format and contains the necessary information to be able to create reproducible builds of the environment:

```json
{
  "concrete_specs": {
    "wlfygd7yywirujlpmgebjwozq5nbvftz": {
      "libffi": {
        "version": "3.2.1",
        "arch": {
          "platform": "darwin",
          "platform_os": "mojave",
          "target": "x86_64"
        },
        "compiler": {
          "name": "gcc",
          "version": "9.1.0"
        },
        "namespace": "builtin",
        "parameters": {
          "cflags": [],
          "cppflags": [],
          "cxxflags": [],
          "fflags": [],
          "ldflags": [],
          "ldlibs": []
        },
        "hash": "wlfygd7yywirujlpmgebjwozq5nbvftz"
      }
    },
    "i5gui4jqndx6kpxt7q52fpjgexwatcp": {
      "py-sphinxautomodapi": {
        "version": "0.9",
        "arch": {
          "platform": "darwin",
          "platform_os": "mojave",
          "target": "x86_64"
        },
        "compiler": {
          "name": "gcc",
          "version": "9.1.0"
        },
        "namespace": "builtin",
```
Installation of additional packages

Additional packages can be installed in the virtual environment with spack add and spack install. For Matplotlib it looks like this:

```
$ spack add py-matplotlib ^python@3.7.3
=> Adding py-matplotlib ^python@3.7.3 to environment /srv/jupyter/jupyterhub/spackenvs/python-374
$ spack install
=> Concretizing py-matplotlib ^python@3.7.3
...=> Installing environment /srv/jupyter/jupyterhub/spackenvs/python-374
...=> Successfully installed py-matplotlib
  Fetch: 2.22s. Build: 52.67s. Total: 54.89s.
[+] /srv/jupyter/spack/opt/spack/linux-debian9-x86_64/gcc-9.1.0/py-matplotlib-3.0.2-4d6nj4hfo3yvkqovp243p4qeebeb5z16
```

**Note:** If a Pipenv environment has already been derived from this Spack environment, it must be rebuilt in order to receive the additional Spack package:

```
$ pipenv install --python=/srv/jupyter/jupyterhub/spackenvs/python-374/.spack-env/view/bin/python
Virtualenv already exists!
Removing existing virtualenv...
Creating a virtualenv for this project...
Pipfile: /srv/jupyter/jupyterhub/pipenvs/python-374/Pipfile
Using /srv/jupyter/jupyterhub/spackenvs/python-374/.spack-env/view/bin/python (3.7.3) to...
  --create virtualenv...
  Creating virtual environment...Using base prefix '/srv/jupyter/jupyterhub/spackenvs/python-374/.spack-env/view'
New python executable in /srv/jupyter/.local/share/virtualenvs/python-374-cwl7BqNA/bin/
  --python
Installing setuptools, pip, wheel...done.
Running virtualenv with interpreter /srv/jupyter/jupyterhub/spackenvs/python-374/.spack-env/view/bin/python
```

8.7. Reproduce environments
✓ Successfully created virtual environment!
Virtualenv location: /srv/jupyter/.local/share/virtualenvs/python-374-cwl7BqNA
Installing dependencies from Pipfile.lock (66106e)...
59/59 - 00:00:28
To activate this project's virtualenv, run pipenv shell. Alternatively, run a command inside the virtualenv with pipenv run.

The installation can then be checked with:

$$
pipenv run python
Python 3.7.3 (default, May 25 2019, 10:40:28)
[GCC 9.1.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import matplotlib.pyplot as plt
Configuration

spack spec specifies the dependencies of certain packages, e.g.

$ spack spec py-matplotlib ^python@3.7.3
Input spec
--------------------------------
py-matplotlib
  ^python@3.7.3
Concretized
--------------------------------
py-matplotlib@3.0.2%
gcc@9.1.0~animation+image~ipython~latex~qt+tk arch=linux-debian9-x86_64
  ^freetype@2.9.1%gcc@9.1.0
  ^patches=08466355e8649235ff0f13b3e56bd551c7cfb2ca97903cc11575c163ea32a3 arch=linux-debian9-x86_64
  ^bzip2@1.0.6%gcc@9.1.0+shared arch=linux-debian9-x86_64
  ^diffutils@3.7%gcc@9.1.0 arch=linux-debian9-x86_64
  ^libpng@1.6.34%gcc@9.1.0 arch=linux-debian9-x86_64
  ^zlib@1.2.11%gcc@9.1.0+optimize+pic+shared arch=linux-debian9-x86_64

With spack config get you can look at the configuration of a certain environment:

$ spack config get
# This is a Spack Environment file.
#
# It describes a set of packages to be installed, along with
# configuration settings.
spack:
  # add package specs to the `specs` list
  specs: [python@3.7.2, py-numpy ^python@3.7.2, py-pandas ^python@3.7.2, py-geopandas
           ^python@3.7.2, py-matplotlib ^python@3.7.2]
mirrors: {}
With `spack config edit` the configuration file `spack.yaml` can be edited.

**Note:** If packages are already installed in the environment, all dependencies should be specified again with `spack concretize -f`.

### Loading the modules

With `spack env loads -r <env>` all modules are loaded with their dependencies.

**Note:** However, this does not currently work when loading modules from environments that are not in `$SPACK_ROOT/var/environments`.

Therefore we replace the directory `$SPACK_ROOT/var/environments` with a symbolic link:

```bash
$ rm $SPACK_ROOT/var/environments
$ cd $SPACK_ROOT/var/
$ ln -s /srv/jupyter/supyterhub/spackenvs environments
```

### See also:

- Environments Tutorial

### Spack mirrors

Some machines may not have internet access to get packages. Then you will need a local repository of tarballs from which to retrieve your files. Spack supports this with **Spack mirrors**. A mirror is a URL that points to a directory on the local file system or on a server and contains tarballs for all Spack packages.

Here is an example of the directory structure of a mirror:

```bash
$ tree /path/to/mirror/
/path/to/mirror/
    ├── autoconf
    │    └── autoconf-2.69.tar.gz
    ├── automake
    │    └── automake-1.16.1.tar.gz
    └── bzip2
        └── bzip2-1.0.8.tar.gz
```

spack mirror create

You can create a mirror with the command `spack mirror create`, provided you are on a machine that can access the Internet. The command iterates through all of Spack’s packages and downloads the ones you want.

spack mirror add

Once you’ve created a mirror, you need to let Spack know about it. It’s relatively easy. First find out the URL of your mirror. If it’s a directory, you can use a file url like this:

```
$ spack mirror add local_filesystem file://$HOME/spack-mirror
```

Order of mirrors

`spack mirror add` adds a line in `~/.spack/mirrors.yaml`:

```
mirrors:
  local_filesystem: file:///home/veit/spack-mirror
  remote_server: https://spack-mirror.cusy.io
```

If you want to change the order in which mirrors are searched for packages, you can edit this file and rearrange the sections: Spack searches them from top to bottom until a suitable entry is found.

Local default cache

Spack creates a cache for resources that are downloaded as part of installations. This cache is a valid Spack mirror: it uses the same directory structure and naming scheme as other Spack mirrors. The mirror is managed locally in the Spack installation directory at `~/.spack/var/spack/cache/`.

8.7.2 Pipenv

Pipenv is a Python package manager. He uses Pip to install Python packages, but also simplifies the management and maintenance of dependencies.
**Installation**

This section covers the basics of installing *Python packages*.

**Requirements for installing packages**

Before installing Python packages, a few prerequisites must be met.

1. Make sure you are using the version of Python you want:

   ```
   $ python --version
   Python 3.6.3
   ```

   **Note:** In iPython or a Jupyter Notebook you can find out the version with:

   ```
   In [1]: import sys
   !{sys.executable} --version
   Python 3.6.3
   ```

   **Note:** If you use the system Python of your Linux distribution, you should first create a virtual environment with Python 3 and *Pip*.

2. Make sure *Pip* is installed:

   ```
   $ pip --version
   pip 10.0.1
   ```

   1. If Pip is not yet installed, you can install it
      * for Python 2 with:
        ```
        $ sudo apt install python-pip
        ```
      * for Python 3 with:
        ```
        $ sudo apt install python3-venv python3-pip
        ```

**Install Pipenv**

*pipenv* is a dependency manager for Python projects. It to install Python packages, but it simplifies dependency management. Pip can be used to install Pipenv, but the *--user* flag should be used so that it is only available to that user. This is to prevent system-wide packages from being accidentally overwritten:

```
$ python3 -m pip install --user pipenv
Downloading pipenv-2018.7.1-py3-none-any.whl (5.0MB): 5.0MB downloaded
Requirement already satisfied (use --upgrade to upgrade): virtualenv in /usr/lib/python3/
--dist-packages (from pipenv)
Installing collected packages: pipenv, certifi, pip, setuptools, virtualenv-clone ...
```

(continues on next page)
Successfully installed pipenv certifi pip setuptools virtualenv-clone
Cleaning up...

Note: If pipenv is not available in the shell after the installation, the USER_BASE/bin directory may have to be specified in PATH.

- On Linux and MacOS the USER_BASE can be determined with:

  ```bash
  $ python3 -m site --user-base /Users/veit/.local
  ```

  Then the bin directory must be appended and added to PATH. Alternatively, PATH can be set permanently by changing ~/.profile or ~/.bash_profile, in my case:

  ```bash
  export PATH=/Users/veit/.local/bin:$PATH
  ```

- On Windows, the directory can be determined with `py -m site --user-site` and then site-packages can be replaced by Scripts. His then gives, for example:

  ```bash
  C:\Users\veit\AppData\Roaming\Python36\Scripts
  ```

  In order to be permanently available, this path can be entered in PATH in the control panel.

Further information on user-specific installations can be found in User Installs.

Create virtual environments

Python virtual environments allow Python packages to be installed in an isolated location for a specific application, rather than installing them globally. So you have your own installation directories and do not share libraries with other virtual environments:

```bash
$ mkdir myproject
$ cd !$
cd myproject
$ pipenv install requests
Creating a virtualenv for this project..
...
Virtualenv location: /Users/veit/.local/share/virtualenvs/myproject-9TTuTZjx
Creating a Pipfile for this project...
Installing requests...
...
```

Usage

Example

Now that requests is installed, it can be used.

1. As an example, we create the file `main.py` with the following content:
```python
import requests

response = requests.get('https://cusy.io')

print(response.status_code)
```

1. Then the script can be executed with:

```bash
$ pipenv run python main.py
```

1. As a result of the call you should receive the HTTP status code 200.

Using `pipenv run` ensures that your installed packages are available for your script.

Alternatively, you can also create a new shell `pipenv shell` with which all installed packages can be accessed:

```bash
$ pipenv shell
Launching subshell in virtual environment...
bash-4.3.30$ . /Users/veit/.local/share/virtualenvs/myproject-9TTuTJzx/bin/activate
```

**Options**

`--venv` specifies the path to the Virtualenv, usually in `~/.local/share/virtualenvs/`. However, if you have created a directory `myproject/.venv`, `pipenv` use this folder to create the associated Python environment there.

`--py` specifies the path to the Python interpreter.

`--envs` outputs options of the environment variables.

For `PIPENV_DONT_LOAD_ENV`, `PIPENV_DONT_USE_PYENV` and `PIPENV_DOTENV_LOCATION` see Environment variables.

If you want to set these environment variables per project, you can use direnv.

Also note that pip itself supports environment variables in case you need additional adjustments: Pip Environment Variables.

Here is another example:

```bash
$ PIP_INSTALL_OPTION="-- -DCMAKE_BUILD_TYPE=Release" pipenv install -e .
```

Further information can be found at Configuration With Environment Variables

`--three`, `--two`, `--python` uses Python 2 or Python 3 or a specific Python to which the path is given.

`--site-packages` enables site packages for the virtual environment.

`--pypi-mirror` indicates a PyPI mirror. The standard is the Python Package Index PyPI).

However, you can also specify your own mirrors:

- with the environment variable `PIPENV_PYPI_MIRROR`
- in the command line, e.g. with:

```bash
$ pipenv install --pypi-mirror https://pypi.cusy.io/simple
$ pipenv update --pypi-mirror https://pypi.cusy.io/simple
...```

- or in pipfile:

8.7. Reproduce environments
You can find more options at pipenv.

check

pipenv check checks for security holes and for PEP 508 <https://www.python.org/dev/peps/pep-0508/> markers in the pipfile. For this it uses safety.

Example:

```
$ pipenv install django==1.10.1
Installing django==1.10.1...
...
$ pipenv check
Checking PEP 508 requirements...
Passed!
Checking installed package safety...

33075: django >=1.10,<1.10.3 resolved (1.10.1 installed)!
Django before 1.8.x before 1.8.16, 1.9.x before 1.9.11, and 1.10.x before 1.10.3, when
--settings.DEBUG is True, allow remote attackers to conduct DNS rebinding attacks by
leveraging failure to validate the HTTP Host header against settings.ALLOWED_HOSTS.

33076: django >=1.10,<1.10.3 resolved (1.10.1 installed)!
Django 1.8.x before 1.8.16, 1.9.x before 1.9.11, and 1.10.x before 1.10.3 use a
--hardcoded password for a temporary database user created when running tests with an
--Oracle database, which makes it easier for remote attackers to obtain access to the
--database server by leveraging failure to manually specify a password in the database
--settings TEST dictionary.

33300: django >=1.10,<1.10.7 resolved (1.10.1 installed)!
CVE-2017-7233: Open redirect and possible XSS attack via user-supplied numeric redirect
--URLs
```

(continues on next page)
Django relies on user input in some cases (e.g. :func:`django.contrib.auth.views.login` and :doc:`i18n </topics/i18n/index>` to redirect the user to an "on success" URL. The security check for these redirects (namely ``django.utils.http.is_safe_url()``) considered some numeric URLs (e.g. ``http:999999999``) "safe" when they shouldn't be.

Also, if a developer relies on ``is_safe_url()`` to provide safe redirect targets and puts such a URL into a link, they could suffer from an XSS attack.

CVE-2017-7234: Open redirect vulnerability in ``django.views.static.serve()``

A maliciously crafted URL to a Django site using the :func:`~django.views.static.serve` view could redirect to any other domain. The view no longer does any redirects as they don't provide any known, useful functionality.

Note, however, that this view has always carried a warning that it is not hardened for production use and should be used only as a development aid.

---

**Note:** pipenv embeds an API client key from pyup.io, instead of including a full copy of the CC-BY-NC-SA licensed database.

In order to install the complete database you can check it out with:

```bash
$ pipenv install -e git+https://github.com/pyupio/safety-db.git#egg=safety-db
```

To use the local database, you have to enter the path to this database, in my case:

```bash
$ pipenv check --db /Users/veit/.local/share/virtualenvs/myproject-9TTuTZjx/src/safety-db/data
```

<table>
<thead>
<tr>
<th>$$</th>
<th>$$</th>
<th>$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$</td>
<td>$$</td>
<td>$$</td>
</tr>
</tbody>
</table>

by pyup.io

REPORT

checked 21 packages, using local DB

---

8.7. Reproduce environments
No known security vulnerabilities found.

**clean**

`pipenv clean` uninstalls all packages not specified in `Pipfile.lock`.

**graph**

`pipenv graph` displays the dependency graph information for the currently installed packages.

**install**

`pipenv install` installs the provided packages and adds them to the pipfile. `pipenv install` knows the following options:

- `-d, --dev` installs the packages in `[dev-packages]`, for example:

  ```
  $ pipenv install --dev pytest
  ...
  $ cat Pipfile
  ...
  [dev-packages]
  pytest = "*
  ```

- `--deploy` aborts if `Pipfile.lock` is out of date or an incorrect Python version is used.

- `-r, --requirements <requirements.txt>` imports a `requirements.txt` file.

- `--sequential` installs the dependency in a specific order, not at the same time.

  While this slows down the installation, it increases the determinability of the builds.

**sdist vs. wheel**

Pip can install packages as *Source Distribution (sdist)* or *Wheel*. If both are present on PyPI, pip will prefer a compatible Wheel.

**Note:** However, dependencies on wheels are not covered by `pipenv lock`.
### Requirement specifier

**Requirement specifier** specify the respective package.

- The latest version can be installed, for example:

  ```
  $ pipenv install requests
  ```

- A specific version can be installed, for example:

  ```
  $ pipenv install requests==2.18.4
  ```

- If the version has to be in a specific version range, this can also be specified:

  ```
  $ pipenv install requests>=2,<3
  ```

- A compatible version can also be installed:

  ```
  $ pipenv install requests~=2.18
  ```

  This is compatible with $2.18.\ast$.

- For some packages, installation options **Extras** can also be specified with square brackets:

  ```
  $ pipenv install requests[security]
  ```

- It can also be specified that certain packages are only installed on certain systems, so for the following **Pipfile**, the module **pywinusb** is only installed on Windows systems.

  ```
  [packages]
  pywinusb = {version = "+", sys_platform = "== 'win32'"}
  ```

  A more complex example differentiates which module versions should be installed with which Python versions:

  ```
  [packages]
  unittest2 = {version = ">=1.0,<3.0", markers="python_version < '2.7.9' or (python_version >= '3.0' and python_version < '3.4')"}
  ```

### VCS

You can also install Python packages from version control, for example:

```
$ pipenv install -e git+https://github.com/requests/requests.git#egg=requests
```

**Note:** If `editable=false`, sub-dependencies are not resolved.

Further information on pipenv and VCS can be found in **Pipfile spec**.

The version management credentials can also be specified in the pipfile, e.g.

```
[[source]]
url = "https://$USERNAME:$PASSWORD@pypi.cusy.io/simple"
verify_ssl = true
name = "cusy-pypi"
```
Note: pipenv hashes Pipfile before the environment variables are determined, and the environment variables are also written to Pipfile.lock, so that no credentials need to be stored in the version control.

lock

pipenv lock generates the file Pipfile.lock that lists all the dependencies and sub-dependencies of your project including the latest available versions and the current hash values for the downloaded files. This ensures repeatable and, above all, deterministic builds.

Note: In order to increase the determinism, the installation sequence can also be guaranteed in addition to the hash values. The --sequential flag is used for this.

Security features

Pipfile.lock uses some security enhancements from pip: by default, sha256 hashes are generated for each downloaded package.

We strongly recommend lock using to deploy development environments to production. In the development environment you use pipenv lock to compile your dependencies and then you can use the compiled file Pipfile.lock in the production environment for reproducible builds.

open

pipenv open MODULE shows a specific module in your editor.

If you use PyCharm <https://www.jetbrains.com/pycharm/>_, you have to configure pipenv for your Python project. How to do this is described in Configuring Pipenv Environment.

run

pipenv run spawns a command that is installed in the virtual environment, for example:

$ pipenv run python main.py

shell

pipenv shell spawns a shell in the virtual environment. This gives you a Python interpreter that contains all Python packages and is therefore ideal for debugging and testing, for example:

```bash
$ pipenv shell --fancy
Launching subshell in virtual environment...
bash-4.3.30$ python
Python 3.6.4 (default, Jan 6 2018, 11:51:59)
>>> import requests
>>> 
```
Note: Shells are usually not configured so that a subshell can be used. This can lead to unexpected results. In these cases `pipenv shell` should be used instead of `pipenv shell --fancy` as this uses a compatibility mode.

sync

`pipenv sync` installs all packages specified in `Pipfile.lock`.

uninstall

`pipenv uninstall` uninstalls all provided packages and removes them from the `Pipfile`. `uninstall` supports all parameters of `install` plus the following two options:

- `--all` deletes all files from the virtual environment, but leaves the `Pipfile` untouched.
- `--all-dev` removes all development packages from the virtual environment and removes them from the `Pipfile`.

update

`pipenv update` runs first `pipenv lock`, then `pipenv sync`.

`pipenv update` has the following options:

- `--clear` clears the dependency cache.
- `--outdated` lists obsolete dependencies.

Deterministic builds

All you have to do is specify what you want:

For example, `pipenv install requests` creates a `Pipfile` like the following:

```python
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
requests = "*

[dev-packages]
[requires]
python_version = "3.6"
```

However, the associated `Pipfile.lock` file specifies the packages exactly, for example:

```json
{
  "default": {
    "requests": {
      "hashes": [
```
Pipfile.lock also specifies all the dependencies of your project, whereby the hash values of the downloaded files are saved. This is to ensure repeatable and deterministic builds.

Workflows

Import and export of requirements.txt files

If you already have a requirements.txt file in an existing project, pipenv can resolve dependencies. If the requirements.txt file is in the same directory, simply with $ pipenv install or, if it is in a different directory, with $ pipenv install -r /path/to/requirements.txt.

Conversely, you can also create a requirements.txt file from an existing Pipenv environment with:

$ pipenv run pip freeze > requirements.txt

Upgrade workflow

1. Find out what has changed upstream:

   $ pipenv update --outdated
   Package 'requests' out-of-date: '==2.13.0' installed, '==2.19.1' available.

2. To update the Python packages, you have the following two options:
   - update everything with $ pipenv update
   - update individual packages, e.g. requests with $ pipenv update requests
**Pipfile vs. setup.py**

A distinction must be made whether you are developing an application or a library.

**Libraries** They offer reusable functions for other libraries and applications/projects. You have to work with other libraries, each with their own dependencies. To avoid version conflicts in dependencies between different libraries within a project, libraries should never commit dependency versions. However, you can specify lower or upper limits if you are relying on a particular feature or bug fix. Library dependencies are noted in `install_requires` of the `setup.py` file.

**Applications** They use libraries and are mostly not dependent on other projects. They should be implemented in a specific environment and only then should the exact versions of all their dependencies and sub-dependencies be specified. Facilitating this process is the main goal of Pipenv.

**Environment variables**

**pipenv environment variables**

`pipenv --envs` outputs options of the environment variables.

For more information, see Configuration With Environment Variables.

**.env file**

If an `.env` file exists in your virtual environment, `$ pipenv shell` and `$ pipenv run` will automatically load it:

```
$ cat .env
USERNAME=Veit

$ pipenv run python
Loading .env environment variables...
...
>>> import os
>>> os.environ['USERNAME']
'Veit'
```

The credentials, e.g. of the version management, can also be specified in the Pipfile, e.g.:

```
[[source]]
url = "https://$USERNAME:${PASSWORD}@pypi.cusy.io/simple"
verify_ssl = true
name = "cusy-pypi"
```

**Note:** `pipenv` hashes the Pipfile before determining the environment variables, and the environment variables are also written in Pipfile.lock, so that no credentials need to be stored in the version management.

You can also save the `.env` file outside your virtual environment. You then only have to specify the path to this file in `PIPENV_DOTENV_LOCATION`:

```
$ PIPENV_DOTENV_LOCATION=/path/to/.env pipenv shell
```

You can also prevent `pipenv` from using an existing `.env` file with:

---

8.7. Reproduce environments
$ PIPENV_DONT_LOAD_ENV=1 pipenv shell

Pipenv and Spack

Pipenv has already been used to install Jupyter notebooks. However, we need Pipenv for our Spack environments to be able to generate binary-compatible builds with Spack on the one hand and to be able to easily use Python packages for data collection, visualization, etc. on the other.

To do this, first activate the appropriate Python version from the Spack environment:

```bash
$ spack env activate python-374
$ spack env status
===> In environment python-374
$ which python
/Users/veit/jupyter-tutorial/spackenvs/python-374/.spack-env/view/bin/python
```

Then you can install the existing Pipenv environment with:

```bash
$ cd ~/jupyter-tutorial/pipenvs/python-374/
$ pipenv --python=/Users/veit/jupyter-tutorial/spackenvs/python-374/.spack-env/view/bin/python --site-packages
$ pipenv install
Creating a virtualenv for this project...
Pipfile: /Users/veit/jupyter-tutorial/pipenvs/python-374/Pipfile
Using /Users/veit/jupyter-tutorial/spackenvs/python-374/.spack-env/view/bin/
→python3.7 (3.7.4) to create virtualenv...
...
```

This uses the environment installed with Spack and installs additional packages.

See also:

- Pipenv and Other Python Distributions

8.7.3 devpi

devpi is a PyPI server that simplifies the packaging, testing and publishing of Python packages.

8.7.4 Glossary

**built distribution** A structure of files and metadata that only needs to be moved to the correct location on the target system during installation. *wheel* is such a format, but not *distutil’s source distributions* that require a build step.

**conda** Package management tool for the Anaconda distribution from Continuum Analytics. It’s specifically aimed at the scientific community, particularly Windows, where installing binary extensions is often difficult.

Conda does not install packages from PyPI and can only install from the official Continuum repositories or from anaconda.org or local (e.g. intranet) package servers. Note, however, that Pip can be installed in conda and can work side by side to manage distributions of PyPI.

See also:

- Conda: Myths and Misconceptions
- Conda build variants
devpi  

Devpi is a powerful PyPI compatible server and PyPI proxy cache with a command line tool to enable packaging, testing and publishing activities.

distribution package  

A versioned archive file that contains Python packages, modules, and other resource files used to distribute a release.

egg  

A built distribution format introduced by setuptools that is now being replaced by wheel. For more information, see The Internal Structure of Python Eggs and Python Eggs.

import package  

A Python module that can contain other modules or recursively other packages.

module  

The basic unit of code reusability in Python, which exists in one of two types:

- pure module  
  A module written in Python contained in a single .py file (and possibly associated .pyc- and/or .pyo files).

- extension module  
  Usually a single dynamically loadable precompiled file, e.g. a common object file (.so).

pip  

A tool for installing Python packages.

Docs | GitHub | PyPI |

Pipfile  

User-friendly, on TOML based alternative to the requirements.txt file of pip.

A distinction can be made between two different groups of packages: [packages] and [dev-packages].

GitHub

Pipfile.lock  

Machine-readable file based on JSON that contains all transitive dependencies with their exact versions and download hashes.

Pipfile.lock also differentiates between [packages] and [dev-packages].

Pipenv  

Pipenv is a project that aims to bring the best of all packaging worlds to the Python world. It combines pipfile, pip and virtualenv in a single toolchain.

Docs | GitHub | PyPI |

pypi.org  

Pypi.org is the domain name for the Python Package Index (PyPI). In 2017 it replaced the old index domain name pypi.python.org. He is supported by warehouse.

Python Package Index (PyPI)  

PyPI is the standard package index for the Python community. All Python developers can use and distribute their distributions.

release  

The snapshot of a project at a specific point in time, identified by a version identifier.

One release can result in several Built Distributions.

setuptools  

Setuptools (and easy_install) is a collection of improvements to the Python Distutils that make it easier to create and distribute Python distributions, especially those that have dependencies on other packages.

source distribution (sdist)  

A distribution format (typically generated using) python setup.py sdist.

It provides metadata and the essential source files required for installation with a tool like Pip or for generating built distributions.

Spack  

A flexible package manager that supports multiple versions, configurations, platforms, and compilers. Spack is similar to the Nix package manager, but allows the definition of virtual dependencies and offers a syntax for parameterisation. The packages are written in Python for easy exchange of compilers, library versions, build options, etc. Any number of versions of packages can coexist on the same system. Spack was developed for rapidly building scientific applications on clusters and supercomputers.

Docs | GitHub | Slides | The Spack package manager: bringing order to HPC software chaos |

virtualenv  

An isolated Python environment that allows packages to be installed for a specific application rather than installing them system-wide.

8.7. Reproduce environments
8.8 Testing

See also:

• Python Testing and Continuous Integration

8.8.1 Concepts

Test Case tests a single scenario.

See also:

• pytest fixtures

Test Fixture is a consistent test environment.

Test Suite is a collection of several test cases.

Test Runner runs through a test suite and presents the results.

8.8.2 Notebooks

Unit tests

[1]:
```python
def add(a, b):
    return a + b
```

[2]:
```python
import unittest
class TestNotebook(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(2, 2), 5)
```

unittest.main(argv=[''], verbosity=2, exit=False)

test_add (__main__.TestNotebook) ... FAIL

======================================================================
FAIL: test_add (__main__.TestNotebook)
======================================================================

(continues on next page)
Traceback (most recent call last):
  File "<ipython-input-2-26d0f87b13a5>" , line 6, in test_add
    self.assertEqual(add(2, 2), 5)
AssertionError: 4 != 5

Ran 1 test in 0.004s

FAILED (failures=1)

Alternatively, ipython-unittest can also be used. This enables the following Cell Magics to be used in iPython:

- %%unittest_main executes test cases that are defined in a cell
- %%unittest_testcase creates a test case with the function defined in a cell and executes it
- %%unittest converts Python assert to unit test functions
- %%external to perform external unit tests
- %%write {mode} to write external files

%reload_ext ipython_unittest

```python
class MyTest(unittest.TestCase):
    def test_1_plus_1_equals_2(self):
        sum = 1 + 1
        self.assertEqual(sum, 2)

    def test_2_plus_2_equals_4(self):
        self.assertEqual(2 + 2, 4)
```

Success

Ran 2 tests in 0.002s

OK
Success
..
-----------------------------------------
Ran 2 tests in 0.000s
OK

[5]: <unittest.runner.TextTestResult run=2 errors=0 failures=0>

[6]:
```python
unittest
"1 plus 1 equals 2"
sum = 1 + 1
assert sum == 2
"2 plus 2 equals 4"
assert 2 + 2 == 4
```

Success
..
-----------------------------------------
Ran 2 tests in 0.000s
OK

[6]: <unittest.runner.TextTestResult run=2 errors=0 failures=0>

By default, Docstring separates the unit test methods in this magic. However, if docstrings are not used, the Cell Magics create one for each `assert` method.

These Cell Magics support optional arguments:

- `-p (--previous) P` puts the cursor in front of P cells (default -1 corresponds to the next cell)

However, this only works if `jupyter_dojo` is also installed.

- `-s (--stream) S` sets the output stream (default is: `sys.stdout`)

- `-t (--testcase) T` defines the name of the TestCase for `%%unittest` and `%%unittest_testcase`

- `-u (--unparse)` outputs the source code after the transformations
**Debugging**

[7]:
```
import doctest
doctest.testmod()

def multiply(a, b):
    
    # This is a test:
    >>> multiply(2, 2)
    5
    
    import pdb; pdb.set_trace()
    return a * b
```

**Doctests**

[1]:
```
import doctest
doctest.testmod()

def add(a, b):
    
    # This is a test:
    >>> add(2, 2)
    5
    
    return a + b
```

**ipytest**

Setup

[1]:
```
# Set the file name (required)
__file__ = 'testing.ipynb'

# Add ipython magics
import ipytest.magics

import ipytest
```

**Test Case**

[2]:
```
%%run_pytest[clean]

def test_sorted():
    assert sorted([4, 2, 1, 3]) == [1, 2, 3, 4]
```

(continues on next page)
rootdir: /Users/veit/jupyter-tutorial/docs/basics/ipython
collected 1 item

testing.py .

[100%]

============== 1 passed in 0.18 seconds ===============

[clean] deletes all previously run tests, ie only the tests defined in the current cell are executed.

**Test Fixture**

```
# [3]: %run_pytest[clean]

@pytest.fixture
def dict_list():
    return [
        dict(a='a', b=3),
        dict(a='c', b=1),
        dict(a='b', b=2),
    ]

def test_sorted_key_example_1(dict_list):
    assert sorted(dict_list, key=lambda d: d['a']) == [
        dict(a='a', b=3),
        dict(a='b', b=2),
        dict(a='c', b=1),
    ]

def test_sorted_key_example_2(dict_list):
    assert sorted(dict_list, key=lambda d: d['b']) == [
        dict(a='c', b=1),
        dict(a='b', b=2),
        dict(a='a', b=3),
    ]

============== test session starts ===============
platform darwin -- Python 3.7.0, pytest-5.0.1, py-1.8.0, pluggy-0.12.0
rootdir: /Users/veit/jupyter-tutorial/docs/basics/ipython
collected 2 items
testing.py ..

[100%]

============== 2 passed in 0.15 seconds ===============
```
---

### Test parameterisation

```python
@pytest.mark.parametrize('input,expected', [(2, 1), (1, 2)],
('zasdqw', list('adqswz')))
def test_examples(input, expected):
    actual = sorted(input)
    assert actual == expected
```

---

#### Reference

**%%run pytest** ...

IPython magic that executes first the cell and then run pytest. Arguments passed in the cell are passed directly to pytest. The Magics should have been imported with `import ipytest.magics` beforehand.

```python
ipytest.run_pytest(module=None, filename=None, pytest_options=(), pytest_plugins=())
```

runs the tests in the existing module (by default `main`) with pytest.

Arguments:

- **module**: the module that contains the tests. If not specified, `__main__` is used.
- **filename**: Filename of the file containing the tests. If nothing is specified, the `__file__` attribute of the passed module is used.
- **pytest_options**: additional options passed to pytest.
- **pytest_plugins**: additional pytest plugins.
**ipytest.run_tests**(doctest=False, items=None)

Arguments:

- **doctest**: If True is specified, angegeben wird, then doctests are searched for.
- **items**: The *globals* object that contains the tests. If None is specified, the *globals* object is obtained from the call stack.

**ipytest.clean_tests**(pattern="test*", items=None)

deletes those tests whose names match the specified pattern.

In IPython, the results of all evaluations are saved in global variables, unless they are explicitly deleted. This behavior implies that if tests are renamed, the previous definitions will still be found if they are not deleted. This method aims to simplify this process.

An effective method is **clean_tests** to start with a cell, then define all test cases and finally **run_tests** call them. That way, renaming tests works as expected.

Arguments:

- **pattern**: A glob pattern that is used to find the tests to delete.
- **items**: The *globals* object that contains the tests. If None is specified, the globals object is obtained from the call stack.

**ipytest.collect_tests**(doctest=False, items=None)

collects all test cases and sends them to unittest.TestSuite.

The arguments are the same as for **ipytest.run_tests**.

**ipytest.assert_equals**(a, b, *args, **kwargs)

compares two objects and throws an exception if they are not the same.

The method **ipytest.get_assert_function** determines the assert implementation to be used depending on the following arguments:

- **a, b**: the two objects to be compared.
- **args, kwargs**: (Keyword) arguments that are passed to the underlying test function.

**ipytest.get_assert_function**(a, b)

determines the assert function to be used depending on the arguments.

If one of the objects is **numpy.ndarray**, **pandas.Series**, **pandas.DataFrame** or **pandas.Panel** the assert functions provided by numpy and pandas will be returned.
ipyleaflet.unittest.assert_equals(a, b)

compares two objects using the assertEqual method of unittest.TestCase.

### 8.8.3 Tools

**unittest2**

unittest2 is a backport of unittest, with improved API and better assertions than in previous Python versions.

**Example**

You may want to import the module under the name unittest to simplify the porting of code to newer versions of the module in the future:

```python
import unittest2 as unittest

class MyTest(unittest.TestCase):
    ...
```

In this way, if you switch to a newer Python version and no longer need the module unittest2, you can simply change the import in your test module without having to change any further code.

**Installation**

```
$ pipenv install unittest2
```

**Note:** If you haven’t installed pipenv yet, you can find instructions on how to do this in *Install Jupyter Notebook*.

**Mock**

unittest.mock is a Python test library. As of Python 3.3, it is available in the Python standard library.

**Example**

It allows you to test parts of your application with mock objects and make assertions about how they have been used. For example, you can do a monkey patch for a method:

```python
from mock import MagicMock
thing = ProductionClass()
thing.method = MagicMock(return_value=3)
thing.method(3, 4, 5, key='value')
thing.method.assert_called_with(3, 4, 5, key='value')
```

The `patch` decorator can be used to create mock classes or objects in a module under test. In the following example, an external search system is replaced by a mock class that always delivers the same result:
**def** mock_search(self):
    
    class MockSearchQuerySet(SearchQuerySet):
        
        def __iter__(self):
            return iter(['foo', 'bar', 'baz'])
        
        return MockSearchQuerySet()

@mock.patch('myapp.SearchForm.search', mock_search)
def test_new_watchlist_activities(self):
    self.assertEqual(len(myapp.get_search_results(q='fish')), 3)

SearchForm refers here to the imported class reference in myapp, not to the SearchForm class itself.

get_search_results performs a search and iterates over the result.

**Installation**

For older Python versions it can be installed with

```
$ pipenv install mock
```

**Note:** If you haven’t installed pipenv yet, you can find instructions on how to do this in *Install Jupyter Notebook*.

**See also:**

With responses you can create mock objects for the Requests library.

**Hypothesis**

Hypothesis is a library that allows you to write tests that are parameterised from a source of examples. Then simple and comprehensible examples are generated, which can be used to fail your tests and to find errors with little effort.

**Example**

To test lists with floating point numbers, many examples are tried, but only a simple example is given in the report for each bug (unique exception type and position):

```
from hypothesis import given
from hypothesis.strategies import lists, floats

given(lists(floats(allow_nan=False, allow_infinity=False), min_size=1))
def test_mean(ls):
    mean = sum(ls) / len(ls)
    assert min(ls) <= mean <= max(ls)
```

```
$ pipenv run pytest
================================================================== test session starts ===================================================================
platform linux -- Python 3.6.8, pytest-4.6.2, py-1.8.0, pluggy-0.12.0
... collected 1 item
```
In our example we let `pytest` find the test and run it. However, we could also have defined class: `python:unittest.TestCase` explicitly:

```python
from hypothesis import given
from hypothesis.strategies import lists, floats
import unittest
class TestMean(unittest.TestCase):
    @given(lists(floats(allow_nan=False, allow_infinity=False), min_size=1))
    def test_mean(ls):
        mean = sum(ls) / len(ls)
        assert min(ls) <= mean <= max(ls)
if __name__ == '__main__':
    unittest.main()
```

**Installation**

```
$ pipenv install hypothesis
```

Alternatively, Hypothesis can also be installed with extras, e.g.

```
$ pipenv install hypothesis[numpy,pandas]
```

**Note:** If you haven’t installed pipenv yet, you can find instructions on how to do this in `Install Jupyter Notebook`. 

8.8. Testing
See also:
Hypothesis for the Scientific Stack

Coverage

You can create coverage reports with `coverage.py`.

See also:
- GitHub
- Docs

Installation

```
pipenv install coverage
```

Note: If you want to determine the test coverage for Python 2 and Python<3.6, you must use Coverage<6.0.

Use

You can run your usual test runner together with coverage with

- ... `pytest`:
  
  ```
pipenv install pytest-cov
  ```

  or for distributed tests

  ```
pipenv install pytest-xdist
  ```

  Afterwards you can check the test coverage with

  ```
pytest --cov=myproj tests/
  ```

See also:
  - pytest-cov's documentation

- ... `Unit tests`:
  
  ```
coverage run -m unittest discover
  ```

- ... `nose`:
  
  ```
coverage run -m nose arg1 arg2
  ```
Codecov

Codecov collects coverage reports for Python, C#/.net, Java, Node/Javascript/Coffee, C/C++, D, Go, Groovy, Kotlin, PHP, R, Scala, Xterm, Xcode, Lua, and other languages, then submits them to codecov.io

See also:

- GitHub
- Docs

Installation

Codecov can be easily installed with

```bash
$ pipenv install codecov
```

Use

... in Terminal

```bash
$ codecov -t <repository-upload-token>
```

... together with GitHub Actions

To do this, you can add the following to your `ci.yml` file, for example:

```yaml
- name: "Convert coverage"
  run: "python -m coverage xml"
- name: "Upload coverage to Codecov"
  uses: "codecov/codecov-action@v1"
  with:
    fail_ci_if_error: true
```

See also:

- Codecov GitHub Action

... together with Travis CI

For this you can add the following in the `.travis.yml` file:

```yaml
language: python
after_success:
  - bash <(curl -s https://codecov.io/bash)
```
... together with tox

Codecov can be set up with tox:

```
[testenv]
passenv = TOXENV CI TRAVIS TRAVIS_* CODECOV_*
deps = codecov>=1.4.0
commands = codecov -e TOXENV
```

Badge

Finally, you can also add a badge for code coverage in your README.rst file, for example with:

```
.. image:: https://codecov.io/gh/YOU/YOUR_PROJECT/branch/main/graph/badge.svg
   :target: https://codecov.io/gh/YOU/YOUR_PROJECT
   :alt: Code Coverage Status (Codecov)
```

Open Coverage

Free and open source solution for Codecov for publishing coverage reports.

See also:
- GitHub

Requirements

- SQLAlchemy compatible database, e.g. PostgreSQL.
- Docker
- Docker Compose

**tox**

tox is a tool for automating virtualenv environment management and testing with multiple interpreter configurations. With tox you can configure complex multi-parameter test matrices via a simple configuration file in the INI style.

Example

Creates a tox.ini file:

```
[tox]
envlist = py27,py36

[testenv]
deps = pytest
commands =
    pytest
```
When you call `pipenv run tox`, the following steps are performed:

1. Optionally create a Python package with

   ```bash
   $ pipenv run python setup.py sdist
   ```

2. Create the environments specified in `envlist`
   In each of these environments, then
   1. the dependencies and packages are installed
   2. the commands specified in `commands` are executed

3. Create a report with the results from each of the environments, e.g.

<table>
<thead>
<tr>
<th>summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>py27: commands succeeded</td>
</tr>
<tr>
<td>ERROR: py36: commands failed</td>
</tr>
</tbody>
</table>

See also:

- Examples

### Installation

```bash
$ pipenv install tox
```

**Note:** If you haven’t installed pipenv yet, you can find instructions on how to do this in under [Install Jupyter Notebook](#).

See also:

- tox plugins

### GitHub Actions

If your project is hosted on GitHub, you can use GitHub Actions to automatically run your tests in different environments: [github.com/actions/virtual-environments].

1. To create a GitHub Action in your project, click Actions → set up a workflow yourself. This usually creates a `.github/workflows/main.yml` file.
2. Give this file a more descriptive name. We usually use `ci.yml`, where `ci` stands for Continuous Integration,
3. The pre-filled YAML file not very useful for our purposes. You can replace the text, for example with:

   ```yaml
   name: CI
   on:
     push:
       branches: ['main']
     pull_request:
       branches: ['main']
   workflow_dispatch:
   ```

   (continues on next page)
jobs:
  tests:
    name: "Python ${{ matrix.python-version }}"
    runs-on: "ubuntu-latest"
    env:
      USING_COVERAGE: '3.6,3.8'
  strategy:
    matrix:
      python-version: ["3.6", "3.7", "3.8"]
  steps:
    - uses: "actions/checkout@v2"
    - uses: "actions/setup-python@v2"
      with:
        python-version: ${{ matrix.python-version }}"
    - name: "Install dependencies"
      run: |
        set -xe
        python -VV
        python -m site
        python -m pip install --upgrade pip setuptools wheel
        python -m pip install --upgrade coverage[toml] virtualenv tox tox-gh-actions
    - name: "Run tox targets for ${{ matrix.python-version }}"
      run: "python -m tox"

Note: Adjust the python versions in python-version if necessary; however, you do not need to change the environment variable in USING_COVERAGE as well, as this is done by the tox plugin tox-gh-actions (see below).

4. You can then click on Start commit. Since we want to make further changes before the tests are executed automatically, we select Create a new branch for this commit and start a pull request and the name for the new branch github-actions. Finally you can click on Create pull request.

5. To switch to the new branch, we go to Code → main → github-actions.

6. tox-gh-actions simplifies running tox in GitHub actions by providing the environment for the tests as the one tox itself uses. However, for this we still need to adapt our tox.ini file, for example:

```
[gh-actions]
python =
  3.6: py36
  3.7: py37, docs
  3.8: py38, lint, typing, changelog
```

This maps GitHub Actions to tox environments.

Note:
- Not all variants of your environment need to be specified. This distinguishes tox-gh-actions from tox
- `py`.

- Make sure that the versions in the `[gh-actions]` section match the available Python versions and, if applicable, those in the GitHub actions for *GitHub Actions for Git pre-commit hooks*.

- Since all tests for a specific Python version are executed one after the other in a container, the advantages of parallel execution are lost here.

---

### See also:

- Build & test Python
- Workflow syntax

7. Now you can add a badge of your CI (Continuous Integration) status to your `README.rst` file, for example with:

```
.. image:: https://github.com/YOU/YOUR_PROJECT/workflows/CI/badge.svg?branch=main
:target: https://github.com/YOU/YOUR_PROJECT/actions?workflow=CI
:alt: CI Status
```

8. You can publish the code coverage on *Codecov*, see *Codecov and GitHub Actions*.

9. You can also add a badge for code coverage in your `README.rst` file, see *Codecov Badge*.

### 8.9 Logging

The *logging* module is part of the Python standard library. It is described in PEP 0282. You can get a first introduction to the module in the *Basic Logging Tutorial*.

Logging usually serves two different purposes:

- **Diagnosis:**
  - You can display the context of certain events.
  - Tools like *Sentry* group related events and facilitate user identification, etc., so that developers can find the cause of the error more quickly.

- **Monitoring:**
  - The logging records events for user-defined heuristics, e.g., for business analyses. These records can be used for reports or optimisation of the business goals and, if necessary, visualised.

What are the advantages of logging over `print`?

- The log file contains all available diagnostic information such as file name, path, function and line number.
- All events are automatically available via the root logger unless they are explicitly filtered out.
- Logging can be muted using either of the following two methods: `logging.Logger.setLevel()` or `logging.disabled`.

### See also:

- *loguru*, which makes logging almost as easy as using print instructions.
- *structlog* adds structure to your log entries.
8.9.1 Logging configuration

[1]: import logging

Example configuration via an INI file

The following example loads the file development.ini in this directory:

```ini
[loggers]
keys=root

[handlers]
keys=stream_handler

[formatters]
keys=formatter

[logger_root]
level=DEBUG
handlers=stream_handler

[handler_stream_handler]
class=StreamHandler
level=DEBUG
formatter=formatter
args=(sys.stderr,)

[formatter_formatter]
format=%(asctime)s %(name)-12s %(levelname)-8s %(message)s
```

[2]: from logging.config import fileConfig

[3]: fileConfig('development.ini')

d = {'clientip': '192.168.0.1', 'user': 'fbloggs'}
logger = logging.getLogger('tcpserver')
logger.warning('Protocol problem: %s', 'connection reset', extra=d)


Pro:

- Possibility to update the configuration on the fly by using the function logging.config.listen() to listen on a socket.
- Different configurations can be used in different environments, for example in development.ini the log level can be specified as DEBUG while the log level in production.ini used WARN.

Con:

- Less control, for example, compared to user-defined filters or loggers that are configured in the code.
Example for a configuration via a dictionary

```python
[4]: import logging
    from logging.config import dictConfig

    logging_config = dict(
        version = 1,
        formatters = {'f':
            'format':
                '%(asctime)s %(name)-12s %(levelname)-8s %(message)s'},
        handlers = {'h':
            'class': 'logging.StreamHandler',
            'formatter': 'f',
            'level': logging.DEBUG},
        root =
            'handlers': ['h'],
            'level': logging.DEBUG},
    )

dictConfig(logging_config)

Pro:
  • Update during operation

Con:
  • Less control than when configuring a logger in code

Example configuration directly in the code

```python
[5]: logger = logging.getLogger()
    handler = logging.StreamHandler()
    formatter = logging.Formatter(
        '%(asctime)s %(name)-12s %(levelname)-8s %(message)s')
    handler.setFormatter(formatter)
    logger.addHandler(handler)
    logger.setLevel(logging.DEBUG)

Alternatively, you can also use Magic Commands:

```
%logstart

Starts logging anywhere in a session

%logstart [-o|-r|-t|-q] [log_name [log_mode]]

If no name is given, ipython_log.py is used in the current directory.

log_mode is an optional parameter. The following modes can be specified:
- append appends the logging information to the end of an existing file
- backup renames the existing file to name~ and writes to name
- global appends the logging information at the end of an existing file
- over overwrites an existing log file
- rotate creates rotating log files: name.1~, name.2~, etc.

Options:
- -o also logs the output of IPython
- -r logs raw output
- -t writes a time stamp in front of each log entry
- -q suppresses the logging output

%logon

Restart the logging

%logoff

Temporary termination of logging

Pro:
- Complete control over the configuration

Con:
- Changes in the configuration require a change in the source code

See also:
- logging configuration
I will introduce you to three different types of web applications:

- **Dashboards** generated from Jupyter notebooks
- Web applications that go beyond notebooks, such as integrating bokeh plots, as demonstrated in Bokeh-Plots in Flask einbinden
- Finally, the provision of your data via a RESTful API, e.g. with the FastAPI framework.

### 9.1 Dashboards

**Jupyter Dashboards Layout Extension** Add-on for Jupyter notebooks, with which outputs (text, plots, widgets, etc.) can be arranged in a design grid or in report form.

**Appmode** Jupyter extension that turns notebooks into web applications.

**nbviewer** It's great for viewing static reports, but it can also be used in conjunction with interaktiven Widgets.

**Panel** was developed on the basis of Bokeh and Param and offers a toolkit especially for creating apps and dashboards, which not only supports bokeh plots, see also Panel: A high-level app and dashboarding solution for the PyData ecosystem.

**Voilà** was developed by QuantStack, see also And voilà!.

**jupyter-flex/index** Jupyter extension that turns notebooks into dashboards.

#### 9.1.1 Activities and licenses

With this tabular overview you can quickly compare the activities and licenses of the various libraries.

<table>
<thead>
<tr>
<th>Name</th>
<th>Stars</th>
<th>Contributors</th>
<th>Commit activity</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jupyter Dashboards Layout Extension</td>
<td>stars 987</td>
<td>contributors 36</td>
<td>commit activity 1/0/year</td>
<td>license not identifiable by github</td>
</tr>
<tr>
<td>Appmode</td>
<td>stars 373</td>
<td>contributors 8</td>
<td>commit activity 1/1/year</td>
<td>license MIT</td>
</tr>
<tr>
<td>nbviewer</td>
<td>stars 1.9k</td>
<td>contributors 36</td>
<td>commit activity 1/1/year</td>
<td>license not identifiable by github</td>
</tr>
<tr>
<td>Panel</td>
<td>stars 1.8k</td>
<td>contributors 72</td>
<td>commit activity 1/1/year</td>
<td>license BSD-3-Clause</td>
</tr>
<tr>
<td>Voilà</td>
<td>stars 3.4k</td>
<td>contributors 56</td>
<td>commit activity 1/1/year</td>
<td>license not identifiable by github</td>
</tr>
<tr>
<td>jupyter-flex/index</td>
<td>stars 133</td>
<td>contributors 3</td>
<td>commit activity 1/0/year</td>
<td>license Apache-2.0</td>
</tr>
</tbody>
</table>
Jupyter Dashboards

The Jupyter Dashboards Layout Extension is an add-on for Jupyter notebooks, with which outputs (text, plots, widgets, etc.) can be arranged in a design grid or in report form. It saves the information on the layout directly in the notebook so that other users of this extension can also see the notebook in the same layout. For examples of dashboards, see Jupyter Dashboards Demos.

Use case

The Jupyter dashboards should solve the following problem:

1. Alice creates a Jupyter notebook with plots and interactive widgets.
2. Alice arranges the notebook cells in a grid or report format.
3. Alice provides the dashboard on a dashboard server.
4. Bob calls up the dashboard on the Jupyter Dashboards Server and interacts with Alice Dashboard application.
5. Alice updates her Jupyter notebook and then makes the dashboard available again on the dashboard server.

Note: For steps 3–5, Jupyter Dashboards Bundler and Jupyter Dashboards Server are also required; however, both are now retired and should not be used any longer.

The roadmap for the Voila-Gridstack-Template is to support the entire specification for the Jupyter dashboards. Currently, however, the Voilà gridstack template is still in an early stage of development, see also And voilà!

Installation of Jupyter dashboards

```bash
$ pipenv install jupyter_dashboards
Installing jupyter_dashboards...
...
$ pipenv run jupyter dashboards quick-setup --sys-prefix
Enabling notebook extension jupyter_dashboards/notebook/main...
  - Validating: OK
$ pipenv run jupyter nbextension enable jupyter_dashboards --py --sys-prefix
Enabling notebook extension jupyter_dashboards/notebook/main...
  - Validating: OK
```

Create dashboard layouts

You can use a normal notebook with markdown and code cells. When you run the cells, text, charts, widgets, etc. are generated. Then you can choose in the Dashboard View either Grid Layout or Report Layout:
With the *Grid Layout* you can change the size of the cells in the grid and move them. You can also use *Cell → Dashboard*:

In the *Report Layout* you can show or hide cells.

In both layouts you can click on *MORE INFO* to get additional information:

- **Move cell**: Click and drag \(\rightarrow\) to move a cell. Hold \(\text{Shift}\) to drag from anywhere on a cell.
- **Resize cell**: Click and drag cell edges or corners to resize.
- **Hide cell**: Click \(\times\) to hide a cell from the dashboard view.
- **Show cell**: Click \(\rightarrow\) to return a hidden cell to the dashboard view.
- **Hide/Show all**: Select the options under *Cell > Dashboard* to hide or show all cells.
- **Edit cell**: Click \(\text{Edit}\) to jump to the Notebook and edit the code.

With *Dashboard Preview* you get a preview, e.g. for the *Matplotlib example*:
**Matplotlib example**

This notebook demonstrates the use of matplotlib in a Jupyter dashboard.

1. To run it, click on *Cell → Run All* in the Notebook menu.
2. Then you can use *View → Dashboard-Preview* to view the diagram in the *Report Layout*.
3. In *View → Dashboard Layout → Grid Layout* you can also display the plot in the *Grid Layout*.

```
[1]: %matplotlib notebook

[2]: import matplotlib.pyplot as plt
    import numpy as np

[3]: x = np.linspace(0, 2*np.pi, 2000)
    y = np.sin(x)

[4]: fig, ax = plt.subplots(figsize=(5,3.5))
    ax.plot(x, y)
    plt.tight_layout()
```

*View → Notebook* brings you back to the notebook editor.
Appmode

Jupyter extension that turns notebooks into web applications.

app-example.ipynb

```python
[1]: from __future__ import division
    import ipywidgets as ipw

    output = ipw.Text(placeholder="0", layout=ipw.Layout(width="190px"), disabled=True)

    def on_click(btn):
        if btn.description == ":=":
            try:
                output.value = str(eval(output.value))
            except:
                output.value = "ERROR"
        elif btn.description == "AC":
            output.value = ""
        elif btn.description == "del":
            output.value = output.value[:-1]
        else:
            output.value = output.value + btn.description

    def mk_btn(description):
        btn = ipw.Button(description=description, layout=ipw.Layout(width="45px"))
        btn.on_click(on_click)
        return btn

    row0 = ipw.HBox([mk_btn(d) for d in ("\(\)", ":=", "del", "AC")])
    row1 = ipw.HBox([mk_btn(d) for d in ("7", "8", "9", "/")])
    row2 = ipw.HBox([mk_btn(d) for d in ("4", "5", "6", "+")])
    row3 = ipw.HBox([mk_btn(d) for d in ("1", "2", "3", "-")])
    row4 = ipw.HBox([mk_btn(d) for d in ("0", ".", ":=", "+")])

    ipw.VBox((output, row0, row1, row2, row3, row4))
```

9.1. Dashboards
Example

When you click on Appmode the notebook app-example.ipynb becomes a clear web application for a calculator:

```
$ jupyter Edit App Logout
```

```
+ | del | AC
7 | 8 | 9 |
4 | 5 | 6 |
1 | 2 | 3 |
0 | + |
```

Installation

For the Jupyter service appmode must be installed with

```
$ pipenv install appmode
Installing appmode...
Collecting appmode
... Installing collected packages: appmode
Successfully installed appmode-0.6.0
Adding appmode to Pipfile's [packages]...
Locking [dev-packages] dependencies...
Locking [packages] dependencies...
Updated Pipfile.lock (ea9a56)!
Installing dependencies from Pipfile.lock (ea9a56)...
  128/128 - 00:09:58
... $ pipenv run jupyter nbextension enable --py --sys-prefix appmode
Enabling notebook extension appmode/main...
  - Validating: OK
$ pipenv run jupyter serverextension enable --py --sys-prefix appmode
Enabling: appmode.server_extension
  - Writing config: /Users/veit/.local/share/virtualenvs/jupyter-tutorial--qS5BvmfG/bin/../etc/jupyter
  - Validating... appmode.server_extension OK
```
Configuration

Server-side configuration

The server can be configured with the following three options:

**Appmode.trusted_path** runs the app mode only for notebooks under this path; Default setting: *no restrictions*.

**Appmode.show_edit_button** displays *Edit App* button in app mode; Default setting: *True*.

**Appmode.show_other_buttons** shows other buttons in app mode, e.g. *Logout*; Default setting: *True*.

You can find more information about the server configuration in *Jupyter paths and configuration*.

Client-side configuration

The UI elements can also be adapted on the client side in the *custom.js* file, e.g. with:

```javascript
// Hides the edit app button.
$('appmode-leave').hide();

// Hides the kernel busy indicator.
$('appmode-busy').hide();

// Adds a loading message.
$('appmode-loader').append('<h2>Loading...</h2>');
```

**Note:** Hiding the *Edit App* button does not prevent users from exiting app mode by manually changing the URL.

Panel

Panel was developed on the basis of Bokeh and Param and offers a toolkit especially for creating apps and dashboards, which not only supports bokeh plots.

See also:
- Panel Announcement
- Panel: A high-level app and dashboarding solution for the PyData ecosystem.

Installation

You can install Panel in the virtual environment of your Jupyter kernel with:

```bash
$ pipenv install panel
Installing panel... Collecting panel ...
Installing collected packages: param, pyviz-comms, pycx, markdown, bokeh, panel
Successfully installed bokeh-1.3.4 markdown-3.1.1 panel-0.6.2 param-1.9.1 pycx-0.4.6 ...
pyviz-comms-0.7.2 ...
```
For some of the following examples additional packages are required such as Holoviews and hvPlot. They can be installed with:

```shell
$ pipenv install "holoviews[recommended]"
Installing holoviews[recommended]...
... Installing collected packages: param, pyviz-comms, kiwisolver, cycler, pyparsing, ...
   ... matplotlib, pyct, markdown, packaging, bokeh, panel, holoviews
Successfully installed bokeh-1.3.4 cycler-0.10.0 holoviews-1.12.5 kiwisolver-1.1.0 ...
   ... markdown-3.1.1 matplotlib-3.1.1 packaging-19.1 panel-0.6.2 param-1.9.1 pyct-0.4.6 ...
   ... pyparsing-2.4.2 pyviz-comms-0.7.2
...
$ pipenv install hvplot
Installing hvplot...
Collecting hvplot
... Installing collected packages: hvplot
Successfully installed hvplot-0.4.0
...```

**Examples**

1. Download

```shell
$ pipenv run panel examples
Copied examples to /Users/veit/jupyter-tutorial/panel-examples
```
2. View

Then you can look at the examples, for example `Introduction.ipynb` with

```shell
$ pipenv run panel serve panel-examples/getting_started/Introduction.ipynb
2019-08-18 10:55:44,056 Starting Bokeh server version 1.3.4 (running on Tornado 6.0. → 3)
2019-08-18 10:55:44,067 Starting Bokeh server with process id: 86677
```

**Overview**

You can add interactive controls in a panel. This allows you to create simple interactive apps, but also complex multi-page dashboards. We’ll start with a simple example of a function for drawing a sine wave with Matplotlib:

[1]:
```python
import numpy as np, pandas as pd, matplotlib.pyplot as plt
%matplotlib inline

def mplplot(df, **kwargs):
    fig = df.plot().get_figure()
    plt.close(fig)
    return fig

def sine(frequency=1.0, amplitude=1.0, n=200, view_fn=mplplot):
```

(continues on next page)
xs = np.arange(n)/n*20.0
ys = amplitude*np.sin(frequency*xs)
df = pd.DataFrame(dict(y=ys), index=xs)
return view_fn(df, frequency=frequency, amplitude=amplitude, n=n)
sine(1.5, 2.5)

Interactive panels

If we wanted to try many combinations of these values to understand how frequency and amplitude affect this graph, we could reevaluate the above cell many times. However, this would be a slow and tedious process. Instead of having to re-enter the values in the code each time, it is advisable to adjust the values interactively with the help of sliders. With such a panel app you can easily examine the parameters of a function. The function of `pn.interact` is similar to `ipywidgets interact`:

```python
[2]: import panel as pn
pn.extension()
pn.interact(sine)
```

As long as a live Python process is running, dragging these widgets calls the `sine` callback function and evaluates
the combination of parameter values you selected and displays the results. With such a panel you can easily examine all functions that provide a visual result of a supported object type (see Supported object types and libraries, e.g. Matplotlib, Bokeh, Plotly, Altair or various text and image types.

Components of panels

interact is handy, but what if you want more control over how it looks or works? First, let’s see what interact is actually created by grabbing the object and viewing its representation:

```python
[3]: i = pn.interact(sine, n=(5,100))
i.pprint()
```

```
Column
  [0] Column
      [0] FloatSlider(end=3.0, name='frequency', start=-1.0, value=1.0)
      [1] FloatSlider(end=3.0, name='amplitude', start=-1.0, value=1.0)
      [2] IntSlider(end=100, name='n', start=5, value=200)
  [1] Row
      [0] Matplotlib(Figure, name='interactive00023')
```

We can see here that the interact call has created a pn.Column object that consists of a WidgetBox (with 3 widgets) and a pn.Row Matplotlib figure. The control panel is compositional, so you can mix and match these components as you like by adding as many objects as needed:

```python
[4]: pn.Row(i[1][0], pn.Column("<br>

# Sine waves", i[0][0], i[0][1]))
```

```
Row
  [0] Matplotlib(Figure, name='interactive00023')
  [1] Column
      [0] Markdown(str)
      [1] FloatSlider(end=3.0, name='frequency', start=-1.0, value=1.0)
      [2] FloatSlider(end=3.0, name='amplitude', start=-1.0, value=1.0)
```

Note that the widgets remain linked to their plot, even if they are in a different notebook cell:

```python
[5]: i[0][2]
```

```python
[5]: IntSlider(end=100, name='n', start=5, value=200)
```

New panels

With this compositional approach, you can combine different components such as widgets, charts, text and other elements that are needed for an app or a dashboard in any way you want. The interact example is based on a reactive programming model in which an input for the function changes and the control panel reactively updates the output of the function. interact is a handy way to automatically build widgets from the arguments for your function. However, Panel also provides a more explicit reactive API that allows you to define connections between widgets and function arguments, and then manually create the resulting dashboard from scratch.

In the following example we explicitly declare every component of an app:

1. Widgets
2. a function for calculating sine values
3. Column and row containers
4. the finished sine_panel app.

Widget objects have several parameters (current value, allowed ranges, etc.), and here we use the `depends` Panel decorator to declare that the input values of the function should come from the value parameters of the widgets. Now, when the function and widgets are displayed, the panel automatically updates the displayed output if one of the inputs changes:

```python
import panel.widgets as pnw

frequency = pnw.FloatSlider(name='frequency', value=1, start=1.0, end=5)
amplitude = pnw.FloatSlider(name='amplitude', value=1, start=0.1, end=10)

@pn.depends(frequency.param.value, amplitude.param.value)
def reactive_sine(frequency, amplitude):
    return sine(frequency, amplitude)

widgets = pn.Column("<br>
# Sine waves", frequency, amplitude)
sine_panel = pn.Row(reactive_sine, widgets)
sine_panel
```

**Deploy panels**

The above panels all work in a notebook cell, but unlike `ipywidgets` and other approaches, Panel apps work on standalone servers as well. The above app can, for example, be started as a separate web server with:

```python
sine_panel.show()
```

```
Row
  [0] ParamFunction(function)
  [1] Column
      [0] Markdown(str)
          [0] FloatSlider(end=5, name='frequency', start=1.0, value=1)
          [2] FloatSlider(end=10, name='amplitude', start=0.1, value=1)
```

This will start the Bokeh server and open a browser window with the application.

Or you can just indicate what you want to see on the website. `servable()`, and then the shell command to start a server with this object `pipenv run panel serve --show example.ipynb`:

```python
sine_panel.servable();
```

The semicolon prevents another copy of the sine field from being displayed here in the notebook.
Declarative Panels

The above compositional approach is very flexible, but it links domain-specific code (the parts about sine waves) to the widget display code. This is common in prototypical projects, but in projects where the code is going to be used in many different contexts, parts of the code that relate to the underlying domains (i.e., the application or research area) should be separated from those that are tied to certain display technologies (such as Jupyter notebooks or web servers).

For such uses, Panel supports objects that have been declared with the separate Param library. This offers a possibility to independently record and declare the parameters of your objects (code, parameters, application and dashboard technology). For example, the above code can be captured in an object that declares the ranges and values of all parameters as well as the generation of the diagram independently of the panel library or any other type of interaction with the object:

```python
import param
class Sine(param.Parameterized):
    amplitude = param.Number(default=1, bounds=(0, None), softbounds=(0, 5))
    frequency = param.Number(default=2, bounds=(0, 10))
    n = param.Integer(default=200, bounds=(1, 200))

def view(self):
    return sine(self.frequency, self.amplitude, self.n)
sine_obj = Sine()
```

The Sine class and sine_obj instance are not dependent on Panel, Jupyter or any other GUI or web toolkit – they simply declare facts about a particular domain (e.g., that sine waves take frequency and amplitude parameters and that the amplitude is a number greater or equals zero). That information is then enough for Panel to create an editable and viewable representation for this object without having to specify anything that depends on the domain-specific details contained in the Sine class and the sine_obj -Instance are not dependent on Panel, Jupyter or any other GUI or web toolkit. They simply declare facts about a certain range (e.g., that sine waves take frequency and amplitude parameters, and that the amplitude is a number greater than or equal to zero). That information is enough for Panel to create an editable and viewable representation for this object without having to specify anything that depends on the domain-specific details contained outside of sine_obj:

```python
pn.Row(sine_obj.param, sine_obj.view)
```

In order to support a certain domain, you can create hierarchies of such classes, in which all parameters and functions are summarised that you need for different object families. Both parameters and code are adopted in the classes, regardless of a specific GUI library or even the existence of a GUI at all. This approach makes it convenient to maintain a large code base that can be fully viewed and edited with Panel, in a way that can be maintained and customised over time.
Linking plots and actions between panels

The above approaches each work with a variety of displayable objects, including images, equations, tables, and charts. In each case, the panel provides interactive functionality using widgets and updates the objects displayed accordingly, making very few assumptions about what is actually displayed. Panel also supports a broader and more dynamic interactivity in which the displayed object itself is interactive, e.g. JavaScript-based diagrams of Bokeh and Plotly.

For example, if we replace the matplotlib wrapper that came with pandas with the Bokeh wrapper hvPlot, we automatically get interactive plots that allow zooming, panning and hovering:

```
import hvplot.pandas

def hvplot(df, **kwargs):
    return df.hvplot()

pn.interact(sine, view_fn=hvplot)
```

These interactive actions can be combined with more complex interactions in a plot (e.g. tap, hover) to make it easier to explore data and uncover connections. For example, we can use HoloViews to create a more comprehensive version of the hvPlot example that is dynamically updated to show the position on the circle as we hover over the sine curve:

```
import holoviews as hv

tap = hv.streams.PointerX(x=0)

def hvplot2(df, frequency, **kwargs):
    plot = df.hvplot(width=500, padding=(0, 0.1))
    tap.source = plot

    def unit_circle(x):
        cx = np.cos(x*frequency)
        sx = np.sin(x*frequency)
        circle = hv.Path([ hv.Ellipse(0,0,2), [(0,0), (1,0)], [(0,-1), (0,1)]]).
        triangle = hv.Path([ [(0,0), (cx,sx)], [(0,0), (cx,0)], [(cx,0), (cx,sx)]]).
        labels = hv.Labels([(cx/2, 0, '%.2f' % cx), (cx, sx/2., '%.2f' % sx))]
        labels = labels.opts(padding=0.1, xaxis=None, yaxis=None, text_baseline='bottom
    ...
```

(continues on next page)
The `interact` function (`panel.interact`) automatically creates controls for interactively browsing code and data.

```
[1]: import panel as pn

from panel.interact import interact, interactive, fixed, interact_manual
from panel import widgets

pn.extension()
```

Interactions

At the simplest level, `interact` controls are automatically generated for function arguments, and the function is then called with those arguments when you interactively edit the controls. To use `interact` you need to define a function that you want to examine. Here is a function that returns the only argument: `x`.

```
[2]: def f(x):
    return x
```

If you pass this function as the first argument together with an integer keyword argument `x=10` to `interact` a slider is generated and bound to the function parameters.
If you move the slider, the function is called, which outputs the current value of `x`. If you pass `True` or `False` `interact` generates a check box:

```python
interact(f, x=True)
```

When you pass a string, `interact` generates a text area.

```python
interact(f, x='Hi Pythonistas!')
```

`interact` can also be used as a `Decorator`. In this way you can define a function as well as determine the type of interaction. As the following example shows, `interact` works also with functions that have multiple arguments.

```python
@interact(x=True, y=1.0)
def g(x, y):
    return (x, y)
```

**Layout of interactive widgets**

The `interact` function returns a panel that contains the widgets and the display output. By indexing these panels we can lay out the objects exactly how we want:

```python
import numpy as np, pandas as pd, matplotlib.pyplot as plt
%matplotlib inline

def mplplot(df, **kwargs):
    fig = df.plot().get_figure()
    ...
```

(continues on next page)
plt.close(fig)
return fig

def sine(frequency=1.0, amplitude=1.0, n=200, view_fn=mplplot):
    xs = np.arange(n)/n*20.0
    ys = amplitude*np.sin(frequency*xs)
    df = pd.DataFrame(dict(y=ys), index=xs)
    return view_fn(df, frequency=frequency, amplitude=amplitude, n=n)

[8]: i = pn.interact(sine, n=(5,100))
      pn.Row(i[1][0], pn.Column("<br><b>Sine waves", i[0][0], i[0][1]))

[8]: Row
    [0] Matplotlib(Figure, name='interactive00056')
    [1] Column
      [0] Markdown(str)
      [1] FloatSlider(end=3.0, name='frequency', start=-1.0, value=1.0)
      [2] FloatSlider(end=3.0, name='amplitude', start=-1.0, value=1.0)

[9]: layout = interact(f, x=10)
      pn.Column("**A custom interact layout**", pn.Row(layout[0], layout[1]))

[9]: Column
    [0] Markdown(str)
    [1] Row
      [0] Column
        [0] IntSlider(end=15, name='p', start=-5, value=10)
        [1] Row
          [0] Str(tuple, name='interactive00075')

**Set arguments with fixed**

There may be times when you want to examine a function using interact, but want to set one or more of its arguments to specific values. This can be achieved using the fixed function:

[10]: def h(p, q):
      return (p, q)

[11]: interact(h, p=5, q=fixed(20))

[11]: Column
      [0] Column
        [0] IntSlider(end=15, name='p', start=-5, value=5)
        [1] Row
          [0] Str(tuple, name='interactive00093')
**Widget abbreviations**

If you pass certain values, `interact` uses automatically the appropriate widget, e.g. a checkbox for `True` or `IntSlider` for integer values. So you don’t have to explicitly specify the appropriate widget:

```
[12]: interact(f, x=widgets.FloatSlider(start=0.0,end=10.0,step=0.01,value=3.0))
[12]: Column
    [0] Column
        [0] FloatSlider(end=10.0, start=0.0, step=0.01, value=3.0)
    [1] Row
        [0] Str(float, name='interactive00106')
```

```
[13]: interact(f, x=(0.0,10.0,0.01,3.0))
[13]: Column
    [0] Column
        [0] FloatSlider(end=10.0, name='x', start=0.0, step=0.01, value=3.0)
    [1] Row
        [0] Str(float, name='interactive00117')
```

This example shows how the keyword arguments are processed by `interact`:

1. If the keyword argument is an instance of `Widget` with a `value` attribute, this widget is used. Any widget with a `value` attribute can be used, including custom ones.

2. Otherwise, the value is treated as a `Widget Abbreviation` that is converted to a widget before use.

The following table gives an overview of the various `Widget Abbreviations`:

<table>
<thead>
<tr>
<th>Keyword argument</th>
<th>Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>True</code> or <code>False</code></td>
<td>Checkbox</td>
</tr>
<tr>
<td>'Hi Pythonistas!'</td>
<td>Text</td>
</tr>
<tr>
<td>Integer value as <code>min,max,step,value</code></td>
<td><code>IntSlider</code></td>
</tr>
<tr>
<td>Floating-point <code>min,max,step,value</code></td>
<td><code>FloatSlider</code></td>
</tr>
<tr>
<td><code>['apple','pear']</code> or <code>{one:1,two:2}</code></td>
<td><code>Dropdown</code></td>
</tr>
</tbody>
</table>

**Widgets**

`Panel` offers a wide range of widgets for precise control of parameter values. The widget classes use a consistent API that allows broad categories of widgets to be treated as interchangeable. For example, to select a value from a list of options, you can use `SelectWidget`, a `RadioButtonGroup` widget, or an equivalent widget interchangeably.

Like all other components in `Panel`, `Widget` objects can also synchronise their state both in the notebook and on the bokeh server:

```
[1]: import panel as pn
    pn.extension()
```

```
Data type cannot be displayed: application/javascript, application/vnd.holoviews_load.v0+json
```

```
Data type cannot be displayed: application/javascript, application/vnd.holoviews_load.v0+json
```

9.1. Dashboards 469
Jupyter Tutorial, Release 0.8.0

[2]: widget = pn.widgets.TextInput(name='A widget', value='A string')
widget

[2]: TextInput(name='A widget', value='A string')

If you change the text value, the corresponding parameter is automatically updated:

[3]: widget.value

[3]: 'A string'

Updating the parameter value also updates the widget:

[4]: widget.value = 'Another string'

Callbacks and links

In order to notice a parameter change, we can call a function `widget.param.watch` with the parameter to be observed:

[5]: from __future__ import print_function

widget.param.watch(print, 'value')

[5]: Watcher(inst=TextInput(name='A widget', value='Another string'), cls=<class 'panel.widgets.input.TextInput'>, fn=<built-in function print>, mode='args', onlychanged=True, parameter_names=('value',), what='value')

If we change now `widget.value`, the resulting event is output.

[6]: widget.value = 'A'

Event(what='value', name='value', obj=TextInput(name='A widget', value='A'), cls=TextInput(name='A widget', value='A'), old='Another string', new='A', type='changed')

PanelWidgets, in combination with objects, enable the easy creation of interactive dashboards and visualisations. For more information on defining callbacks and links between widgets and other components, see the User Guide.

Widgets

To put several widgets together, they can be added to a Row-, Column- or Tabs panel. For more information on the layout of widgets and control panels, see the customization user guide.

[7]: slider = pn.widgets.FloatSlider(name='Another widget', width=200)
pn.Column(widget, slider, width=200)

[7]: Column(width=200)

[0] TextInput(name='A widget', value='A')

[1] FloatSlider(name='Another widget', width=200)
Widget categories

The supported widgets can be divided into different categories based on their compatible APIs.

Option selection

With option selection widgets you can select one or more values from a list or a dictionary. All widgets of this type have options and value parameters.

<table>
<thead>
<tr>
<th>Options</th>
<th>Widget</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual</td>
<td>AutocompleteInput</td>
<td>selects one value from an automatically completed text field</td>
</tr>
<tr>
<td>values</td>
<td>DiscretePlayer</td>
<td>displays controls of mediaplayer, that allow you to play and step through</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the options available</td>
</tr>
<tr>
<td></td>
<td>DiscreteSlider</td>
<td>selects a value with a slider</td>
</tr>
<tr>
<td></td>
<td>RadioButtonGroup</td>
<td>selects a value from a series of mutually exclusive toggle keys</td>
</tr>
<tr>
<td></td>
<td>RadioBoxGroup</td>
<td>selects a value from a series of mutually exclusive check boxes</td>
</tr>
<tr>
<td></td>
<td>Select</td>
<td>selects a value from a drop-down menu</td>
</tr>
<tr>
<td>Multiple</td>
<td>CheckBoxGroup</td>
<td>select values by activating the corresponding check boxes</td>
</tr>
<tr>
<td>values</td>
<td>CheckButtonGroup</td>
<td>select values by toggling the corresponding buttons</td>
</tr>
<tr>
<td></td>
<td>CrossSelector</td>
<td>select values by moving items between two lists</td>
</tr>
<tr>
<td></td>
<td>MultiSelect</td>
<td>select values by marking them in a list</td>
</tr>
</tbody>
</table>

Type-based selectors

Type-based selectors offer the possibility of choosing a value according to its type. All selectors have a value. In addition to the type, the widgets in this category can also have other forms of validation, for example the upper and lower limits of sliders.
<table>
<thead>
<tr>
<th>Types</th>
<th>Widget</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Individual values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Numerically</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntSlider</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td>FloatSlider</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td><strong>Boolean values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checkbox</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td><strong>Date</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DatetimeInput</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td>DatePicker</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td>DateSlider</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TextInput</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ColorPicker</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td>FileInput</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td>LiteralInput</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td><strong>Areas</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Numerically</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntRangeSlider</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td>RangeSlider</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td><strong>Dates</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DateRangeSlider</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audio</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
<tr>
<td>Button</td>
<td></td>
<td>allows the selection of a single value type</td>
</tr>
</tbody>
</table>
Parameterisation

Panel supports the use of parameters and dependencies between parameters, expressed in a simple way by `param`, to encapsulate dashboards as declarative, stand-alone classes.

Parameters are Python attributes that have been extended using the `param` library to support types, ranges, and documentation. This is just the information you need to automatically create widgets for each parameter.

Parameters and widgets

For this purpose, some parameterised classes with different parameters are declared first:

```python
[1]: import param
    import datetime as dt

class BaseClass(param.Parameterized):
    x = param.Parameter(default=3.14,doc="X position")
    y = param.Parameter(default="Not editable",constant=True)
    string_value = param.String(default="str",doc="A string")
    num_int = param.Integer(50000,bounds=(-200,100000))
    unbounded_int = param.Integer(23)
    float_with_hard_bounds = param.Number(8.2,bounds=(7.5,10))
    float_with_soft_bounds = param.Number(0.5,bounds=(0,None),softbounds=(0,2))
    unbounded_float = param.Number(30.01)
    hidden_parameter = param.Number(2.718)
    integer_range = param.Range(default=(3,7),bounds=(0, 10))
    float_range = param.Range(default=(0,1.57),bounds=(0, 3.145))
    dictionary = param.Dict(default={"a":2, "b":9})

class Example(BaseClass):
    """An example Parameterized class""
    timestamps = []
    boolean = param.Boolean(True, doc="A sample Boolean parameter")
    color = param.Color(default='#FFFFFF')
    date = param.Date(dt.datetime(2017, 1, 1),
                      bounds=(dt.datetime(2017, 1, 1), dt.datetime(2017, 2, 1)))
    select_string = param.ObjectSelector(default="yellow",objects=['red','yellow','green'])
    select_fn = param.ObjectSelector(default=list,objects=[list,set,dict],precedence=0.5)
    int_list = param.ListSelector(default=[3,5], objects=[1,3,5,7,9],
                                   precedence=0.5)
    single_file = param.FileSelector(path='../*/*.py*',precedence=0.5)
    multiple_files = param.MultiFileSelector(path='../*/*.py?',precedence=0.5)
    record_timestamp = param.Action(lambda x: x.timestamps.append(dt.datetime.now()),
                                     doc="""Record timestamp.""",precedence=0.7)

Example.num_int

[1]: 50000
```
As you can see, the declaration of parameters only depends on the separate `param` library. Parameters are a simple idea with a few properties critical to creating clean, usable code:

- The `param` library is written in pure Python with no dependencies, which makes it easy to include in any code without tying it to a specific GUI or widgets library, or to Jupyter notebooks.
- Parameter declarations focus on semantic information that is relevant to your domain. In this way, you avoid contaminating domain-specific code with anything that binds it to a specific display or interaction with it.
- Parameters can be defined wherever they make sense in your inheritance hierarchy, and you can document them once, enter them and limit them to a certain area. All these properties are inherited from any base class. For example, all parameters work the same here, regardless of whether they were declared in `BaseClass` or `Example`. This makes it easier to provide this metadata once and prevents it from being duplicated anywhere in the code where areas or types need to be checked or documentation saved.

If you then decide to use these parameterised classes in a notebook or web server environment, you can easily display and edit the parameter values as an optional additional step with `import panel`:

```
[2]: import panel as pn

pn.extension()

base = BaseClass()

Row(Example.param, base.param)
```

If you decide to use these parameterised classes in a notebook or web server environment, you can easily display and edit the parameter values as an optional additional step with `import panel`:

```
[2]: import panel as pn

pn.extension()

base = BaseClass()

Row(Example.param, base.param)
```

(continues on next page)
As you can see, Panel does not need to have knowledge of your domain-specific application, nor of the names of your parameters. It simply shows widgets for all parameters that have been defined for this object. By using Param with Panel, an almost complete separation between your domain-specific code and your display code is achieved, which considerably simplifies the maintenance of both over a longer period of time. Here even the msg behavior of the buttons was declared declaratively as an action that can be called regardless of whether it is used in a GUI or in another context.

Interaction with the above widgets is only supported in the notebook and on the bokeh server. However, you can also export static renderings of the widgets to a file or a website.

If you edit values in this way, you have to run the notebook cell by cell by default. When you get to the cell above, edit the values as you wish and execute the following cells, in which these parameter values are referred to, your interactively selected settings are used:

```
[3]: Example.unbounded_int
[3]: 23

[4]: Example.num_int
[4]: 50000
```

To work around this and automatically update all widgets generated from the parameter, you can pass the param object:

```
[5]: pn.Row(Example.param.float_range, Example.param.num_int)
[5]: Row
[ 0] RangeSlider(end=3.145, name='Float range', value=(0, 1.57))
[ 1] IntSlider(end=100000, name='Num int', start=-200, value=50000)
```

### Custom widgets

In the previous section we saw how parameters can be automatically converted into widgets. This is possible because the Panel internally manages an assignment between parameter types and widget types. However, sometimes the standard widget doesn’t provide the most convenient user interface, and we want to give Panel an explicit hint on how a parameter should be rendered. This is possible with the widgets argument for the Param panel. With the widgets keyword we can declare an association between the parameter name and the desired widget type.

As an example we can assign a RadioButtonGroup and a DiscretePlayer to a String and a Number selector.
It is also possible to pass arguments to the widget to customise it. Instead of passing the widget, pass a dictionary with the options you want. Uses the `type` keyword to map the widget:

```
[7]: pn.Param(CustomExample.param, widgets={
    'select_string': {'type': pn.widgets.RadioButtonGroup, 'button_type': 'primary'},
    'select_number': pn.widgets.DiscretePlayer}
)
```

```
[7]: Param(ParameterizedMetaclass, widgets={'select_string': {'button...})
```

### Parameter dependencies

Declaring parameters is usually just the beginning of a workflow. In most applications, these parameters are then linked to a computation. To express the relationship between a computation and the parameters on which it depends, the `param.depends` decorator for parameterized methods can be used. This decorator gives panels and other param-based libraries (e.g. HoloViews) an indication that the method should be recalculated if a parameter is changed.

As a simple example with no additional dependencies, let’s write a small class that returns an ASCII representation of a sine wave that depends on `phase` and `frequency` parameters. When we pass the `.view` method to a panel, the view is automatically recalculated and updated as soon as one or more of the parameters change:

```
[8]: import numpy as np

class Sine(param.Parameterized):
    phase = param.Number(default=0, bounds=(0, np.pi))
    frequency = param.Number(default=1, bounds=(0.1, 2))

    @param.depends('phase', 'frequency')
    def view(self):
        y = np.sin(np.linspace(0, np.pi*3, 40)*self.frequency+self.phase)
        y = ((y-y.min())/y.ptp())*20
        array = np.array([list((' '*int(round(d))-1) + '*').ljust(20)) for d in y])
        return pn.pane.Str('
'.join([r.join([r for r in array.T]), height=325,width=500])

sine = Sine(name='ASCII Sine Wave')
pn.Row(sine.param, sine.view)
```

```
[0] Column(margin=5, name='Param00161', width=300)
```

(continues on next page)
The parameterised and annotated view method can return any type provided by the Pane-Objects panel. This makes it easy to link parameters and their associated widgets to a plot or other output. Parameterised classes can therefore be a very useful pattern for encapsulating part of a computational workflow with an associated visualisation and for declaratively expressing the dependencies between the parameters and the computation.

By default, a Param area (Pane) shows widgets for all parameters with a precedence value above the value pn.Param.display_threshold, so you can use precedence to automatically hide parameters. You can also explicitly choose which parameters should contain widgets in a certain area by passing an parameters argument. For example, this code outputs a phase widget, keeping sine.frequency the initial value 1:

```python
[9]: pn.Row(pn.panel(sine.param, parameters=['phase']), sine.view)
```

Another common pattern is linking the values of one parameter to another parameter, for example when there are dependencies between parameters. In the following example we define two parameters, one for the continent and one for the country. Since we would like the selection of valid countries to change when we change continent, let's define a method to do this for us. To connect the two, we express the dependency using the param.depends decorator and then use watch=True to ensure that the method is executed when the continent is changed.

We also define a view method that returns an HTML iframe showing the country using Google Maps.

```python
[10]: class GoogleMapViewer(param.Parameterized):
    continent = param.ObjectSelector(default='Asia', objects=['Africa', 'Asia', 'Europe'])
    country = param.ObjectSelector(default='China', objects=['China', 'Thailand', 'Japan'])
    _countries = {'Africa': ['Ghana', 'Togo', 'South Africa', 'Tanzania'],
                  'Asia': ['China', 'Thailand', 'Japan'],
                  'Europe': ['Austria', 'Bulgaria', 'Greece', 'Portugal', 'Switzerland']}

    @param.depends('continent', watch=True)
    def _update_countries(self):
        countries = self._countries[self.continent]
        self.param['country'].objects = countries
        self.country = countries[0]

    @param.depends('country')
    def view(self):
        iframe = ""
        iframe += '<iframe width="800" height="400" src="https://maps.google.com/maps?q='
        iframe += '{country}&z=6&output=embed"
        ```
```
```
Whenever the continent changes, the _update_countries method for changing the displayed country list is now executed, which in turn triggers an update of the view method.

```
[10]: Row
[0] Column(margin=5, name='Param00201', width=300)
[0] StaticText(value='<b>Google Map V...)
[1] Select(name='Continent', options=OrderedDict([('Africa', ...]), value='Asia')
[2] Select(name='Country', options=OrderedDict([('China', ...]), value='China')
[1] ParamMethod(method)
```

```
from bokeh.plotting import figure

class Shape(param.Parameterized):
    radius = param.Number(default=1, bounds=(0, 1))

    def __init__(self, **params):
        super(Shape, self).__init__(**params)
        self.figure = figure(x_range=(-1, 1), y_range=(-1, 1))
        self.renderer = self.figure.line(*self._get_coords())

    def _get_coords(self):
        return [], []

    def view(self):
        return self.figure

class Circle(Shape):
    n = param.Integer(default=100, precedence=-1)

    def __init__(self, **params):
        super(Circle, self).__init__(**params)

    def _get_coords(self):
        angles = np.linspace(0, 2*np.pi, self.n+1)
        return (self.radius*np.sin(angles),
                self.radius*np.cos(angles))

    @param.depends('radius', watch=True)
    def update(self):
        xs, ys = self._get_coords()
        self.renderer.data_source.data.update({'x': xs, 'y': ys})

class NGon(Circle):
```

(continues on next page)
Parameter sub-objects

Parameterized objects often have parameter values that are Parameterized objects themselves and form a tree-like structure. With the control panel you can not only edit the parameters of the main object, but also access sub-objects.

Let’s first define a hierarchy of Shape classes that will draw a bokeh plot of the selected Shape:

```python
from bokeh.plotting import figure

class Shape(param.Parameterized):
    radius = param.Number(default=1, bounds=(0, 1))

    def __init__(self, **params):
        super(Shape, self).__init__(**params)
        self.figure = figure(x_range=(-1, 1), y_range=(-1, 1))
        self.renderer = self.figure.line(*self._get_coords())

    def _get_coords(self):
        return [], []

    def view(self):
        return self.figure

class Circle(Shape):
    n = param.Integer(default=100, precedence=-1)

    def __init__(self, **params):
        super(Circle, self).__init__(**params)

    def _get_coords(self):
        angles = np.linspace(0, 2*np.pi, self.n+1)
        return (self.radius*np.sin(angles),
                self.radius*np.cos(angles))

    @param.depends('radius', watch=True)
    def update(self):
        xs, ys = self._get_coords()
        self.renderer.data_source.data.update({'x': xs, 'y': ys})

class NGon(Circle):
    n = param.Integer(default=3, bounds=(3, 10), precedence=1)
```

(continues on next page)
Now that we have multiple Shape classes we can create instances of them and create a ShapeViewer to choose between. We can also declare two methods with parameter dependencies that update the plot and the plot title. It should be noted that the param.depends decorator can not only depend on parameters on the object itself, but can also be expressed on certain parameters on the subobject, for example shape.radius or with shape.param on parameters of the subobject.

```python
@param.depends('radius', 'n', watch=True)
def update(self):
    xs, ys = self._get_coords()
    self.renderer.data_source.data.update({'
x': xs, 'y': ys})
```

Let's start with the standard view, which has a toggle button to expand the sub-object:

```python
shapes = [NGon(), Circle()]

class ShapeViewer(param.Parameterized):
    shape = param.ObjectSelector(default=shapes[0], objects=shapes)

    @param.depends('shape')
def view(self):
        return self.shape.view()

    @param.depends('shape', 'shape.radius')
def title(self):
        return '## %s (radius=%.1f)' % (type(self.shape).__name__, self.shape.radius)

    def panel(self):
        return pn.Column(self.title, self.view)
```

480 Chapter 9. Create web applications
Alternatively, we can offer a completely separate `expand_layout` instance for a param area, which with the `expand` and `expand_button` option always remains expanded. This allows us to separate the main widgets and the sub-object’s widgets:

```python
[15]: viewer = ShapeViewer()
    expand_layout = pn.Column()

    pn.Row(
        pn.Column(
            pn.panel(viewer.param, expand_button=False, expand=True, expand_layout=expand_layout),
            "#### Subobject parameters:",
            expand_layout),
        viewer.panel())
```

### Styling

Panel objects build on `param`, which allows them to be specified by parameters so that users can flexibly edit to control the output displayed. In addition to the parameters specific to each component and component class, all components define a common set of parameters to control the size and style of the rendered view.

```python
[1]: import panel as pn

    pn.extension()
```

Data type cannot be displayed: application/javascript, application/vnd.holoviews_load.v0+json

Data type cannot be displayed: application/javascript, application/vnd.holoviews_load.v0+json
**Styling components**

`css_classes`

The `css_classes` parameter enables a panel component to be assigned to one or more CSS classes. CSS can be specified directly in the notebook or as a reference to an external CSS file by passing it to the Panel extension with `raw_css` or `css_files` as a list. Outside a notebook, in an external module or library, we can attach configuration parameters with `pn.config.raw_css` and `pn.config.js_files`.

To demonstrate this usage, let’s define a CSS class named `widget-box`:

```python
[2]: css = '''
    .widget-box {
        background: #f0f0f0;
        border-radius: 5px;
        border: 1px black solid;
    }
'''

pn.extension(raw_css=[css])
```

If we just want to give the component a background, we can define one as a hex string:

```python
[4]: pn.Column(background='#f0f0f0', width=100, height=100)
```

`background`

If we just want to give the component a background, we can define one as a hex string:

```python
[4]: Column(background='#f0f0f0', height=100, sizing_mode='fixed', width=100)
```
Certain components, especially markup-related panes, provide a `style` parameter that can be used to define CSS styles that are applied to the HTML container of the window content, e.g. the Markdown pane:

```python
[5]: pn.pane.Markdown('# Cusy: DevOps', style={'font-family': 'sans-serif'})
[5]: Markdown(str, style={'font-family': '...'})
```

### Component size and layout

The size of the components and their spacing are also controlled by a number of parameters that are shared by all components.

#### margin

The `margin` parameter can be used to create space around an element, which is defined as the number of pixels in the order top, right, bottom and left, e.g.

```python
[6]: pn.Row(
    pn.Column(pn.widgets.Button(name='Selector', margin=(20, 16, 20, 26)), background='˓→#f0f0f0'),
    pn.Column(pn.widgets.Button(name='Widget', margin=(20, 16, 20, 0)), background='˓→#f0f0f0'),
    pn.Column(pn.widgets.Button(name='Description', margin=(20, 26, 20, 0)), background='˓→#f0f0f0'))
```

### Absolute dimensioning with width and height

By default, all components use either automatic or absolute resizing. Panels generally take up as much space as the components they contain, and text- or image-based panels adjust to the size of their content. To set a fixed size for a component, it is usually sufficient to set a width or height. In certain cases, however, `sizing_mode='fixed'` must be specified explicitly.

```python
[7]: pn.Row(
    pn.pane.Markdown('\>CUSY_', style={'color': "white", 'font-weight': "300", 'background': "black", 'width': "100px", 'height': "100px", 'padding': "10px"}),
    pn.pane.GIF('../../ipywidgets/smiley.gif', width=100),
    pn.widgets.FloatSlider(width=100))
```

```python
[7]: Row
[0] Markdown(str, style={'color': 'white', ...})
[1] GIF(str, width=100)
[2] FloatSlider(width=100)
```
sizing_mode

sizing_mode can have the following values:

- **fixed**: The component is not responsive. The original width and height are retained regardless of subsequent events that resize the browser window. This is the default behavior and just uses the specified width and height.

- **stretch_width**: The component resizes to stretch it to the available width without maintaining the aspect ratio. The height of the component depends on the type of component and can be fixed or tied to the content of the component.

- **stretch_height**: The component is resized appropriately to fit the available height, but without maintaining the aspect ratio. The width of the component depends on the type of component and can be fixed or tied to the content of the component.

- **stretch_both**: The component is responsive, regardless of width or height, and occupies all available horizontal and vertical space, even if this changes the aspect ratio of the component.

- **scale_height**: The component is resized appropriately to stretch it to the available height while maintaining the original or provided aspect ratio.

- **scale_width**: The component is resized appropriately to stretch it to the available width while maintaining the original or provided aspect ratio.

- **scale_both**: The component is resized to the available width and height, while maintaining the original or provided aspect ratio.

```python
[8]: pn.pane.Str(background='f0f0f0', height=100, sizing_mode='stretch_width')
```

```python
[8]: Str(Empty, background='f0f0f0', height=100, sizing_mode='stretch_width')
```

```python
[9]: pn.Column(pn.pane.Str(background='f0f0f0', sizing_mode='stretch_height'), height=100)
```

```python
[9]: Column(height=100)
    [0] Str(Empty, background='f0f0f0', sizing_mode='stretch_height')
```

```python
[10]: pn.Column(pn.pane.Str(background='f0f0f0', sizing_mode='stretch_both'), height=100)
```

```python
[10]: Column(height=100)
    [0] Str(Empty, background='f0f0f0', sizing_mode='stretch_both')
```

```python
[11]: pn.Column(pn.pane.GIF('.../ipywidgets/smiley.gif', sizing_mode='scale_both'),
              background='f0f0f0')
```

```python
[11]: Column(background='f0f0f0')
    [0] GIF(str, sizing_mode='scale_both')
```

**Spacer**

Spacer are a very versatile component that can be used to easily create fixed or responsive distances between objects. Like all other components, Spacer support both absolute and responsive mode:

```python
[12]: pn.Row(1, pn.Spacer(width=200), 2, pn.Spacer(width=100), 3, pn.Spacer(width=50), 4, pn.
      Spacer(width=25), 5)
```
VSpacer and HSpacer ensure an attractive vertical or horizontal distance. With these components we can place objects equidistantly on a layout and shrink the empty area when the browser is resized:

```python
[13]: Row
  [0] Str(int)
  [1] Spacer(width=200)
  [2] Str(int)
  [3] Spacer(width=100)
  [4] Str(int)
  [5] Spacer(width=50)
  [6] Str(int)
  [8] Str(int)
```

**Deploy and export**

One of the main design goals for Panel was to enable a seamless transition between interactively prototyping a dashboard and deploying it as a standalone server app. This notebook shows how to interactively display panels, embed static output, save a snapshot, and serve it as a separate web server app.

**Configure output**

Panel objects are automatically displayed in a notebook and use Jupyter Comms to support communication between the rendered app and the Jupyter kernel. The display of a panel object in the notebook is simple: it only has to load the `panel.extension` first in order to initialise the required JavaScript in the notebook context.

```python
[1]: import panel as pn
pn.extension()
```

Data type cannot be displayed: application/javascript, application/vnd.holoviews_load.v0+json

Data type cannot be displayed: application/javascript, application/vnd.holoviews_load.v0+json

9.1. Dashboards
Optional dependencies

In order to be able to use certain components such as Vega, LaTeX and Plotly-Plots, the corresponding Javascript components must also be loaded. To do this, you can simply include them as part of the call to `pn.extension`:

```
[2]: pn.extension('vega', 'katex')
```

Initialise JS and CSS

Additional CSS and Javascript can also be specified with `css_files`, `js_files` and `raw_css`. `js_files` should be specified as a dictionary mapping from the exported JS module name to the URL with the JS components, while `css_files` can be defined as a list:

```
[3]: pn.extension(js_files={'deck': 'https://unpkg.com/deck.gl@~5.2.0/deckgl.min.js'}, css_files=['https://api.tiles.mapbox.com/mapbox-gl-js/v0.44.1/mapbox-gl.css'])
```

With this `raw_css` argument you can define a list of strings with CSS that should be published as part of the notebook and the app.

Providing keyword arguments with `extension` is equivalent to specifying with `pn.config`. `pn.config` is the preferred approach to add Javascript and CSS files outside of a notebook:

```
[4]: pn.config.js_files = {'deck': 'https://unpkg.com/deck.gl@~5.2.0/deckgl.min.js'}
pn.config.css_files = ['https://api.tiles.mapbox.com/mapbox-gl-js/v0.44.1/mapbox-gl.css']
```

Display in the notebook

Once `extension` is loaded, panel objects that are placed at the end of a cell are displayed:

```
[5]: pane = pn.panel('<marquee>Here is some custom HTML</marquee>')</n```

486 Chapter 9. Create web applications
The display function

To avoid having to put a panel in the last row of a notebook cell, you can use the IPython display function:

```
[6]: def display_marquee(text):
    display(pn.panel('<marquee>{text}</marquee>'.format(text=text)))

display_marquee('This Panel was displayed from within a function')
```

Inline apps

Finally, it is also possible to display a panel object as a bokeh server app in the notebook. To do this, call the .app method in the panel object and enter the URL of your notebook server:

```
[7]: pane.app('localhost:8888')
```

The app is now executed in an instance of the Bokeh server that is separate from the Jupyter notebook kernel, so that you can quickly test whether the entire functionality of your app works both in the notebook and in the server context.

Display in an interactive Python window (REPL)

If you work via the command line, extensive displays are not automatically displayed inline, as is the case in a notebook. However, you can interact with your panel components if you start a Bokeh server instance and use the show method to open a separate browser window. The method has the following arguments:

- **port**: int, (optional): allows a specific port to be specified (default=0 select any open port)
- **websocket_origin**: str or list(str) (optional): A list of hosts that can connect to the websocket. This is necessary when a server app is embedded in an external website. If not specified, localhost is used.
- **threaded**: boolean (optional, default=False): True starts the server in a separate thread and allows you to interact with the app.

The show call returns either a Bokeh server instance (threaded=False) or an StoppableThread instance (threaded=True), both provide a stop method for stopping the server instance.

Starting a server from the command line

Panel (and Bokeh) provide a CLI command to deploy a Python script, app directory, or Jupyter notebook with a Bokeh or Panel app. To start a server using the CLI, simply do the following:

```
$ pipenv run panel serve app.ipynb
```

To turn a notebook into a deployable app, simply attach to one or more panel objects .servable(), which adds the app to bokeh's curdoc. This makes it easy to create dashboards interactively in a notebook and then seamlessly provide them to the Bokeh server.
**Session status**

- `panel.state` exposes some of the internal Bokeh server components to users.
- `panel.state.curdoc` allows access to the current `bokeh.document`.

**Embed**

Panel generally needs either the Jupyter kernel or a Bokeh server running in the background to enable interactive behavior. However, for simple apps it is also possible to capture the entire widget status so that the app can be used entirely from Javascript. To demonstrate this, let’s create a simple app that simply takes a slider value, multiplies that by 5, and then displays the result:

```python
[8]: slider = pn.widgets.IntSlider(name='Integer to Scientific Notation Converter', start=0, end=10)
@pn.depends(slider.param.value)
def callback(value):
    return '%d = %e' % (value, value)
row = pn.Row(slider, callback)
```

If you try the above widget, you will find that it only has three different status 0, 5 and 10. This is because embedding attempts to limit the number of options for non-discrete or semi-discrete widgets to a maximum of three values by default. This can be changed with the `max_opts` argument of the `embed` method. The full options for the `embed` method are:

- `max_states`: Maximum number of states to be embedded
- `max_opts`: Maximum number of states for a single widget
- `json`: Specifies whether the data should be exported to json files
- `save_path`: Path to save JSON files
- `load_path`: Path or URL from which the JSON files are loaded (as `save_path` unless otherwise specified)

As you can easily imagine, a combinatorial explosion of the statuses can quickly occur with several widgets, so that the output is limited to around 1000 statuses by default. For larger apps, the status can also be exported to JSON files. For example, if you want to make the app available on a website, specify `save_path` where the JSON file should be saved and `load_path` where the JS code should search for the files.
Save

If you don’t need an actual server or just want to export a static snapshot of a panel app, you can use the save method which can be used to export the app to a standalone HTML or PNG file.

By default, the generated HTML file depends on loading the JavaScript code for BokehJS from the online CDN repository to reduce the file size. If you need to work in a networked or non-networked environment, you can choose to use INLINE resources instead of CDN:

```python
from bokeh.resources import INLINE
pane.save('deploy-panel.html', resources=INLINE)
```

To export the png file you also need Selenium and PhantomJS:

```bash
$ pipenv install selenium
Installing selenium...
...
$ npm install -g phantomjs-prebuilt
...
Done. Phantomjs binary available at /usr/local/lib/node_modules/phantomjs-prebuilt/lib/
  ...phantom/bin/phantomjs
+ phantomjs-prebuilt@2.1.16
added 81 packages from 76 contributors in 31.121s
```

In addition, you can use the save method together with the embed option to embed the app status in the app or to save it in JSON files, which can be deployed together with the exported HTML code. You have the following options:

- **resources**: bokeh.resources, e.g. CDN or INLINE
- **embed**: Boolean value, whether the status should be saved in the file or not.
- **max_states**: The maximum number of states to be embedded
- **max_opts**: The maximum number of states for a single widget
- **embed_json**: Boolean value as to whether the data should be exported as a JSON file (default=True).

Pipelines

In parameterisation is described how classes are created, which declare the parameters and link calculations or visualisations. In this section you will learn how you can connect several such panels with a pipeline to express complex workflows in which the output of one stage is fed into the next stage.

```python
import param
import panel as pn
pn.extension('katex')
```
While we saw earlier how methods are linked to the `param.depends` decorator, pipelines use a different decorator and a convention for displaying the objects. The `param.output` decorator provides a way to annotate the methods of a class by declaring its output. Pipelines use this information to determine what outputs are available to be fed into the next stage of the workflow. In the following example, the class `Stage1` has two parameters (`a` and `b`) and an output `c`. The decorator’s signature allows a number of different ways to declare the outputs:

- `param.output()`: If output is declared with no arguments, the method returns output that inherits the name of the method and does not make any specific type declarations.
- `param.output(param.Number)`: When declaring an output with a specific parameter or a Python type, the output is declared with a specific type.
- `param.output(c=param.Number)`: If an output is declared with a keyword argument, you can overwrite the method name as the name of the output and declare the type.

It is also possible to declare several parameters as keywords or as tuples:

- `param.output(c=param.Number, d=param.String)`
- `param.output(('c', param.Number), ('d', param.String))`

In the example below, the output is simply the result of multiplying the two inputs (`a` and `b`) that produce the output `c`. In addition, we declare a `view` method that returns a LaTeX pane. Finally, a `panel` method returns a Panel object that render both the parameters and the view.

```python
[3]: class Stage1(param.Parameterized):
    a = param.Number(default=5, bounds=(0, 10))
    b = param.Number(default=5, bounds=(0, 10))
    @param.output(('c', param.Number), ('d', param.Number))
    def output(self):
        return self.a * self.b, self.a ** self.b
    @param.depends('a', 'b')
    def view(self):
        c, d = self.output()
        return pn.pane.LaTeX('\$a \times b = \{c\n^{{{a}}}^{\{b\}} = \{d\}$'.format(a=self.a, b=self.b, c=c, d=d))
    def panel(self):
        return pn.Row(self.param, self.view)
stage1 = Stage1()
stage1.panel()
```

In summary, we followed a few conventions to create this stage of our pipeline:
1. Declare a parameterised class with some input parameters
2. Declare and name one or more output methods
3. Declare a panel method that returns a View of the object that the pipeline can render.

Now that the object has been instantiated, we can also ask it about its outputs:

```
[4]: stage1.param.outputs()

[4]: {'c': (<param.Number at 0x11ab652c0>,
    <bound method Stage1.output of Stage1(a=5, b=5, name='Stage101510'>,
    0),
    'd': (<param.Number at 0x11ab65530>,
    <bound method Stage1.output of Stage1(a=5, b=5, name='Stage101510'>,
    1)})
```

We can see that Stage1 declared an output with the name c of the type Number that can be accessed using the output method. Now let's add stage1 with add_stage to our pipeline:

```
[5]: pipeline.add_stage('Stage 1', stage1)
```

For a pipeline, however, we still need at least one stage2 that processes the result of stage1. Therefore a parameter c should be declared from the result of stage1. As a further parameter, we define exp and a view method again, which depends on the two parameters and the panel method.

```
[6]: class Stage2(param.Parameterized):
    
c = param.Number(default=5, precedence=-1, bounds=(0, None))
    exp = param.Number(default=0.1, bounds=(0, 3))

    @param.depends('c', 'exp')
    def view(self):
        return pn.pane.LaTeX('${%s}^{{%s}}={%.3f}$' % (self.c, self.exp, self.c**self.exp))

    def panel(self):
        return pn.Row(self.param, self.view)

stage2 = Stage2(c=stage1.output()[0])
stage2.panel()
```

Also, we now add stage2 to the pipeline object:

```
[7]: pipeline.add_stage('Stage 2', stage2)
```

We now have a two-stage pipeline where the output c is passed from stage1 to stage2. Now we can display the pipeline with pipeline.layout:

```
[8]: pipeline.layout
```
The rendering of the pipeline shows a small diagram with the available workflow stages and the *Previous* and *Next* buttons to switch between the individual phases. This enables navigation even in more complex workflows with many more phases.

Above we instantiated each level individually. However, if the pipeline is to be deployed as a server app, the stages can also be declared as part of the constructor:

```python
[9]: stages = [
    ('Stage 1', Stage1),
    ('Stage 2', Stage2)
]

pipeline = pn.pipeline.Pipeline(stages)
```

The pipeline stages can either be `Parameterized` instances or `Parameterized` classes. With instances, however, you have to make sure that the update of the parameters of the class also updates the current status of the class.
Templates

If you want to provide a panel app or a dashboard as a bokeh application, it is rendered in a standard template that refers to the JS and CSS resources as well as the actual panel object. If you want to adapt the layout of the provided app or if you want to provide several separate panels in one app, the Template component of Panel allows you to adapt this standard template.

Such a template is defined with Jinja, whereby you can extend or even completely replace the standard template. Here is an example:

```html
<!DOCTYPE html>
<html lang="en">
{% block head %}
<head>
  {%- block inner_head %}
  <meta charset="utf-8">
  <title>{% block title %}{% e if title else "Panel App" %}{% endblock %}</title>
  {%- block preamble %}{% endblock %}
  {%- block resources %}
    {%- block css_resources %}
      {{ bokeh_css | indent(8) if bokeh_css }}
    {%- endblock %}
    {%- block js_resources %}
      {{ bokeh_js | indent(8) if bokeh_js }}
    {%- endblock %}
  {%- endblock %}
  {%- block postamble %}{% endblock %}
{%- endblock %}
</head>
{% endblock %}
{% block body %}
{% block inner_body %}
  {%- block contents %}
    {%- for doc in docs %}
      {%- embed(doc) if doc.elementid %}
    {%- endfor %}
    {%- for root in doc.roots %}
      {%- embed(root) %}
    {%- endfor %}
  {%- endblock %}
  {%- plot_script %}
{%- endblock %}
</body>
{% endblock %}
</html>
```

The template defines a number of user-defined blocks that can be supplemented or overwritten by `extends:`

9.1. Dashboards 493
Use custom templates

[1]:
```
import panel as pn
import holoviews as hv

pn.extension()
```

Once we have Panel loaded, we can start defining a custom template. It is usually easy to customise an existing template by overwriting certain blocks. With `{% extends base %}` we declare that we are only expanding an existing template and not defining a new one.

In the following case, we are expanding the `postamble` block of the header to load an additional resource and the `contents` block to redefine the arrangement of the components:

[2]:
```
{%- extends base %}

<!-- head -->
{%- block postamble %}
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
{%- endblock %}

<!-- body -->
{%- block contents %}
<h1>Custom template for multiple apps</h1>
<p>This is a Panel app with a custom template allowing us to compose multiple Panel objects into a single HTML document.</p>
<br>
<div class="container">
  <div class="row">
    <div class="col-sm">
      {{ embed(roots.A) }}
    </div>
    <div class="col-sm">
      {{ embed(roots.B) }}
    </div>
  </div>
</div>
{%- endblock %}
```

Using the `embed` macro, we have defined two different `roots` in the template. In order to be able to render the template, we must first create the `pn.Template` object with the HTML template and then integrate the two `roots` objects.

[3]:
```
tmpl = pn.Template(template)
```

(continues on next page)
tmpl.add_panel('A', hv.Curve([1, 2, 3]))
tmpl.add_panel('B', hv.Curve([1, 2, 3]))
tmpl.servable()

[3]: Template
    [A] HoloViews(Curve)
    [B] HoloViews(Curve)

A button is rendered in the notebook with which you can start a local server to check whether the output meets your expectations.

If the template is larger, it is often easier to create it in a separate file. You can use the Jinja2 template loading mechanism by defining an environment together with a loader:

[4]: from jinja2 import Environment, FileSystemLoader
    env = Environment(loader=FileSystemLoader('.'))
    jinja_template = env.get_template('sample_template.html')
    tmpl = pn.Template(jinja_template)
    tmpl.add_panel('A', hv.Curve([1, 2, 3]))
    tmpl.add_panel('B', hv.Curve([1, 2, 3]))
    tmpl

[4]: Template
    [A] HoloViews(Curve)
    [B] HoloViews(Curve)

Voilà

Voilà was developed by QuantStack.

Features

- Voilà supports interactive Jupyter widgets, including round trips to the kernel. Custom widgets like bqplot, ipyleaflet, ipyvolume, ipympl, ipysheet, plotly, ipywebrtc etc. are also supported.
- Voilà does not allow arbitrary code execution by dashboard users.
- Voilà is based on Jupyter standard protocols and file formats and works with any Jupyter-Kernel: C++, Python, Julia. This makes it a language-independent dashboard system.
- Voilà is expandable. It contains a flexible Template system for creating extensive layouts.
Execution model

An important aspect of this execution model is that the frontend cannot specify which code is executed by the backend. Unless otherwise specified with the option `--strip-sources=False`, the source code of the rendered notebook does not even reach the frontend. The Voilà instance of `jupyter_server` does not allow execution requests by default.

Warning: The current version of Voilà does not respond to the first GET request until all cells have been executed. This can take longer. However, work is being done to enable progressive rendering, see `feat: progressive cell rendering`.

See also:
- Voilà Gallery
- And voilà!

Installation and use

Installation

voila can be installed with:

```bash
$ pipenv run voila docs/jupyter/ipywidgets/examples.ipynb
pipenv install voila
Installing voila...
...Successfuly installed jupyter-server-0.1.1 jupyterlab-pygments-0.1.0 voila-0.1.10
...```
Start

... as a stand-alone application

You can check the installation, e.g. with:

```
$ pipenv run voila pipenv run voila docs/workspace/jupyter/ipywidgets/examples.ipynb
...
[Voila] Voila is running at:
http://localhost:8866/
```

Your standard browser should open and display the *ipywidgets* examples from our tutorial:

### Beispiele

*IPython* enthält eine Architektur für interaktive Widgets, die Python-Code, der im Kernel ausgeführt wird, und JavaScript/HTML/CSS, die im Browser ausgeführt werden, zusammenfügt. Mit diesen Widgets können Benutzer ihren Code und ihre Daten interaktiv untersuchen.

#### Interact-Funktion

*ipywidgets.interact* erstellt automatisch User-Interface(UI)-Controls, um Code und Daten interaktiv zu erkunden.

Im einfachsten Fall generiert *interact* automatisch Steuerelemente für Funktionsargumente und ruft dann die Funktion mit diesen Argumenten auf, wenn Sie die Steuerelemente interaktiv bearbeiten. Im folgenden eine Funktion, die ihr einziges Argument `x` ausgibt.

#### Slider

Wenn ihr eine Funktion mit einem ganzzahligen keyword argument (e.g. `x=10`) angebt, wird ein Schieberegler generiert und an den Funktionsparameter gebunden:

![Slider](image)

#### Checkbox

Wenn ihr `True` oder `False` angebt, generiert *interact* eine Checkbox:

![Checkbox](image)

Alternatively, you can also display a directory with all the notebooks it contains:

```
$ pipenv run voila docs/workspace/jupyter/ipywidgets
...
```

<table>
<thead>
<tr>
<th>Select items to open with voila.</th>
</tr>
</thead>
<tbody>
<tr>
<td>libs</td>
</tr>
<tr>
<td>examples.ipynb</td>
</tr>
<tr>
<td>networkx.ipynb</td>
</tr>
<tr>
<td>custom-widget.ipynb</td>
</tr>
<tr>
<td>widget-list.ipynb</td>
</tr>
<tr>
<td>widget-events.ipynb</td>
</tr>
</tbody>
</table>

It is also possible to display the source code with:

9.1. Dashboards
Note: Note that the code is only displayed. Voilà does not allow users to edit or run the code.

\[\text{Beispiele}\]

IPython enthält eine Architektur für interaktive Widgets, die Python-Code, der im Kernel ausgeführt wird, und JavaScript/HTML/CSS, die im Browser ausgeführt werden, zusammenfügt. Mit diesen Widgets können Benutzer ihren Code und ihre Daten interaktiv untersuchen.

**Interact-Funktion**

\[
\text{ipywidgets.interact}\text{ erstellt automatisch User-Interface(UI)-Controls, um Code und Daten interaktiv zu erkunden.}
\]

In [1]:

```
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

Im einfachsten Fall generiert \texttt{interact} automatisch Steuerelemente für Funktionsargumente und ruft dann die Funktion mit diesen Argumenten auf, wenn Sie die Steuerelemente interaktiv bearbeiten. Im folgenden eine Funktion, die ihr einziges Argument \texttt{x} ausgibt.

In [2]:

```
def f(x):
    return x
```

Slider

Wenn Sie eine Funktion mit einem ganzzahligen \texttt{keyword argument} \texttt{(x=10)} angeben, wird ein Schieberegler generiert und an den Funktionsparameter gebunden:

In [3]:

```
interact(f, x=10);
```

\begin{verbatim}
  x  10
  10
\end{verbatim}

Checkbox

Wenn Sie \texttt{True} oder \texttt{False} angeben, generiert \texttt{interact} eine Checkbox:

In [4]:

```
interact(f, x=True);
```

\begin{verbatim}
   x
  True
\end{verbatim}

Textbereich

... as an extension of the Jupyter server

Alternativ können Sie \texttt{voilà} als einen Teil des Jupyter-Server starten:

\[\text{\$ pipenv run jupyter notebook} \]

... Then you can call up voilà, e.g. under the URL \texttt{http://localhost:8888/voila}.
**Templating**

**Voilà gridstack**

`gridstack.js` is a jQuery plugin for widget layouts. This enables multi-column drag and drop grids and customizable layouts suitable for Bootstrap v3. It also works with `knockout.js` and touch devices.

The Gridstack Voilà template uses the metadata of the notebook cells to design the notebook’s layout. It is supposed to support the entire specification for the outdated Jupyter Dashboards.

<table>
<thead>
<tr>
<th>Voila + Gridstack.js demo</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Gridstack Voilà template" /></td>
</tr>
</tbody>
</table>

**Decorator**

`interact` kann auch als Decorator verwendet werden. Auf diese Weise können Sie eine Funktion definieren und in einer einzigen Einstellung damit interagieren. Wie das folgende Beispiel zeigt, funktioniert `interact` auch mit Funktionen, die mehrere Argumente haben:

**Textbereich**

Wenn ihr einen String übergebt, generiert `interact` einen Textbereich:

**Checkbox**

Wenn ihr `True` oder `False` angebt, generiert `interact` eine Checkbox.

**voila-vuetify**

`voila-vuetify` is a template for using Voilà with the Material Design Component Framework Vuetify.js.
Installation

```bash
$ pipenv install bqplot ipyvuetify voila-vuetify==voila-vuetify 0.0.1a8
```

Usage

To use voila-vuetify in a notebook, you first have to import ipyvuetify:

```python
import ipyvuetify as v
```

Then you can create a layout, e.g. with:

```python
v.Tabs(_metadata={'mount_id': 'content-main'}, children=[
    v.Tab(children=['Tab1']),
    v.Tab(children=['Tab2']),
    v.TabItem(children=[
        v.Layout(row=True, wrap=True, align_center=True, children=[
            v.Flex(xs12=True, lg6=True, xl4=True, children=[
                fig, slider
            ]),
            v.Flex(xs12=True, lg6=True, xl4=True, children=[
                figHist2, sliderHist2
            ]),
            v.Flex(xs12=True, xl4=True, children=[
                fig2
            ])
        ]),
        v.TabItem(children=[
            v.Container(children=['Lorem ipsum'])
        ])
    ])
]
```

You can use `bqplot_vuetify_example.ipynb` with:

```bash
$ pipenv run voila --template vuetify-default bqplot_vuetify_example.ipynb
```

Then your standard browser will open the URL http://localhost:8866/ and show you the plots in Responsive Material Design.

Example for Voilà-vuetify with the monitor resolution of a laptop MDPI screen:
Example for Voilà-vuetify with the monitor resolution of an iPhone X:
voila-debug

voila-debug is a template for displaying debug information when working on Voilà applications.
**Installation**

```
$ pipenv install voila-debug
```

**Usage**

You can use the template `debug.ipynb` with:

```
$ pipenv run voila --template=debug --VoilaExporter.template_file=debug.tpl
```

This will open your default browser with the URL `localhost:8866`.

Then you can take a closer look at how it works at `http://localhost:8866/voila/render/docs/jupyter/dashboards/voila/debug.ipynb`.

In addition to an example widget, it contains a code cell for exiting the kernel:

```python
import os
def kill_kernel(change):
    os._exit(0)

button = widgets.Button(description="Kill Kernel")
button.on_click(kill_kernel)
button
```

**voila-reveal**

`voila-reveal` is a template for slideshows based on RevealJS.

**Installation**

```
$ pipenv install voila-reveal
```
Usage

You can use the template with:

```
$ pipenv run voila --template=reveal reveal.ipynb
```

Additional options can be used to override the default settings, e.g. to change the default value for transition Fade to Zoom with:

```
$ pipenv run voila --template=reveal --VoilaConfiguration.resources="{'reveal': {
    'transition': 'zoom'}}" reveal.ipynb
```

If configuration options are to be saved permanently, a conf.json file can be created in share/jupyter/voila/templates/reveal/

```
{
    "traitlet_configuration": {
        "resources": {
            "reveal": {
                "scroll": false,
                "theme": "simple",
                "transition": "zoom"
            }
        }
    }
}
```

You can then turn your notebook into a slideshow in View → Cell Toolbar → Slideshow. In a cell toolbar you can choose between

- **Slide** left to right
- **Sub-Slide** top to bottom
- **Fragment** stops inside a slide

**Notes** Speaker notes opened in a new window when the presenter press the t key

If you want to publish your slideshow on binder, you must write the following tag in the metadata of the notebook in Edit → Edit Notebook Metadata:

```
"rise": {
    "autolaunch": true
}
```

You can also use the chalkboard reveal plugin in the metadata of the notebook:

```
"rise": {
    "enable_chalkboard": true
}
```
Create your own templates

A Voilà template is a folder that is located in the virtual environment at `share/jupyter/voila/templates` and for example, contains the following:

```
/Users/veit/.local/share/virtualenvs/jupyter-tutorial--q5BvmfG/share/jupyter/voila/…templates/mytheme
├── conf.json
├── nbconvert_templates
│   └── voila.tpl
├── static
│   ├── mytheme.js
│   └── mytheme.css
└── templates
    ├── 404.html
    ├── browser-open.html
    ├── error.html
    └── page.html
    └── tree.html
```

**conf.json** Configuration file that e.g. refers to the basic template:

```
{"base_template": "default"}
```

**nbconvert_templates** Custom templates for nbconvert `nbconvert`.

**static** Directory for static files.

**templates** Custom tornado templates.

**bqplot_vuetify_example.ipynb**

**Import**

```
[1]: import ipyvuetify as v
```

**First histogram plot**

```
[2]: import ipywidgets as widgets
    import numpy as np
    from bqplot import pyplot as plt
    import bqplot

    n = 200
    x = np.linspace(0.0, 10.0, n)
    y = np.cumsum(np.random.randn(n)*10).astype(int)

    fig = plt.figure( title='Histogram')
    np.random.seed(0)
```
hist = plt.hist(y, bins=25)
hist.scales['sample'].min = float(y.min())
hist.scales['sample'].max = float(y.max())
fig.layout.width = 'auto'
fig.layout.height = 'auto'
fig.layout.min_height = '300px' # so it shows nicely in the notebook
fig

Slider

[3]: slider = v.Slider(thumb_label='always', class_="px-4", v_model=30)
widgets.link((slider, 'v_model'), (hist, 'bins'))
slider

Line chart

[4]: fig2 = plt.figure( title='Line Chart')
np.random.seed(0)
p = plt.plot(x, y)

fig2.layout.width = 'auto'
fig2.layout.height = 'auto'
fig2.layout.min_height = '300px' # so it shows nicely in the notebook
fig2

Add BrushIntervalSelector

[5]: brushintsel = bqplot.interacts.BrushIntervalSelector(scale=p.scales['x'])

def update_range(*args):
    if brushintsel.selected is not None and brushintsel.selected.shape == (2,):
        mask = (x > brushintsel.selected[0]) & (x < brushintsel.selected[1])
        hist.sample = y[mask]

brushintsel.observe(update_range, 'selected')
fig2.interaction = brushintsel
Second histogram plot

```python
[6]: n2 = 200

x2 = np.linspace(0.0, 10.0, n2)
y2 = np.cumsum(np.random.randn(n2)*10).astype(int)

figHist2 = plt.figure( title= 'Histogram 2')
np.random.seed(0)
hist2 = plt.hist(y2, bins=25)
hist2.scales[ 'sample' ].min = float(y2.min())
hist2.scales[ 'sample' ].max = float(y2.max())
figHist2.layout.width = 'auto'
figHist2.layout.height = 'auto'
figHist2.layout.min_height = '300px' # so it shows nicely in the notebook

sliderHist2 = v.Slider( _metadata={ 'mount_id': 'histogram_bins2' }, thumb_label= 'always',
                      class_ = 'px-4', v_model=5)
from traitlets import link
link((sliderHist2, 'v_model'), (hist2, 'bins'))

display(figHist2)
display(sliderHist2)
```

Set up voila vuetify layout

The Voila vuetify template does not show the output of the Jupyter Notebook, only the widget with the mount_id metadata.

```python
[7]: v.Tabs( _metadata={ 'mount_id': 'content-main' }, children=[
    v.Tab(children=[ 'Tab1' ]),
    v.Tab(children=[ 'Tab2' ]),
    v.TabItem(children=[
        v.Layout(row= True, wrap= True, align_center= True, children=[
            v.Flex(xs12= True, lg6= True, xl4= True, children=[
                fig, slider
            ]),
            v.Flex(xs12= True, lg6= True, xl4= True, children=[
                figHist2, sliderHist2
            ]),
            v.Flex(xs12= True, xl4= True, children=[
                fig2
            ])
        ])
    ])
```

(continues on next page)
### debug.ipynb

<table>
<thead>
<tr>
<th>Code</th>
</tr>
</thead>
</table>
| ```python | import ipywidgets as widgets  
| | slider = widgets.FloatSlider(description='x')  
| | text = widgets.FloatText(disabled=True, description='$x^2$')  
| | def compute(*ignore):  
| | text.value = str(slider.value**2)  
| | slider.observe(compute, 'value')  
| | slider.value = 14  
| | widgets.VBox([slider, text])  
| ``` |
| VBox(children=(FloatSlider(value=14.0, description='x'), FloatText(value=196.0, description='$x^2$'), disabled=...)

<table>
<thead>
<tr>
<th>Code</th>
</tr>
</thead>
</table>
| ```python | import os  
| | def kill_kernel(change):  
| | os._exit(0)  
| | button = widgets.Button(description="Kill Kernel")  
| | button.on_click(kill_kernel)  
| | button  
| ``` |
| Button(description='Kill Kernel', style=ButtonStyle())

---

Chapter 9. Create web applications
INDEX

A
ACID, 219
autoclass (directive), 377
autoexception (directive), 377
automodule (directive), 377

B
BASE, 219
branch, 344
built distribution, 432

C
CAP theorem, 219
Cassandra, 219
close, 344
Column Family, 220
conda, 432
Consistency, 220
Consistent hash function, 220
CouchDB, 220

D
devpi, 433
distribution package, 433

E
egg, 433
environment, 381
event variable
   python-version, 448
Eventually Consistent, 220

F
fork, 344

G
Git, 344
git commit, 344
GitLab, 344
Graph model, 220
Graph partitioning, 221
Graph traversal, 220

H
HBase, 221
HEAD, 344
Hypertable, 221
import package, 433

K
Key/value pair, 221

L
Locking, 221

M
MapReduce, 221
Merge request, 344
module, 433
MongoDB, 221
MVCC - Multiversion Concurrency Control, 221

N
Notebook cell, 8
Notebook kernel, 8

P
Paxos, 221
pip, 433
Pipenv, 433
Pipfile, 433
Pipfile.lock, 433
Property graph model (PGM), 221
pypi.org, 433
Python Package Index (PyPI), 433
python-version, 448

R
Redis, 221
release, 433
remote, 344
Riak, 221

S
Semantic integrity, 222
setuptools, 433
source directory, 381
source distribution (sdist), 433
Spack, 433

T
Test Case, 434
Test Fixture, 434
Test Runner, 434
Test Suite, 434

V
Vector clock, 222
virtualenv, 433

W
Warehouse, 434
wheel, 434

X
XPATH, 222
XQuery, 222
XSLT, 222