
Pyodide Documentation

Release 0.14.0

Mozilla?

Aug 22, 2019

1	Using Pyodide	3
2	Developing Pyodide	13
3	Indices and tables	27

Important: From your browser, you can [try Pyodide in an Iodide notebook](#). The [Iodide documentation site](#) provides additional user and developer documentation.

The Python scientific stack, compiled to WebAssembly.

Note: Pyodide bundles support for the following packages: numpy, scipy, and many other libraries in the Python scientific stack.

To use additional packages from PyPI, try the experimental feature, [Installing packages from PyPI](#) and try to *pip install* the package.

To create a Pyodide package to support and share libraries for new applications, try [Creating a Pyodide package](#).

Pyodide may be used in several ways, including in an Iodide notebook, directly from JavaScript, or to execute Python scripts asynchronously in a web worker. Although still experimental, additional packages may be installed from PyPI to be used with Pyodide.

1.1 Using Pyodide from Iodide

This document describes using Pyodide inside Iodide. For information about using Pyodide directly from Javascript, see [Using Pyodide from Javascript](#).

1.1.1 Running basic Python

Create a Python chunk, by inserting a line like this:

```
%% py
```

Type some Python code into the chunk, and press Shift+Enter to evaluate it. If the last clause in the cell is an expression, that expression is evaluated, converted to Javascript and displayed in the console like all other output in Javascript. See [type conversions](#) for more information about how data types are converted between Python and Javascript.

```
%% py
import sys
sys.version
```

1.1.2 Loading packages

Only the Python standard library and `six` are available after importing Pyodide. Other available libraries, such as `numpy` and `matplotlib` are loaded on demand.

If you just want to use the versions of those libraries included with Pyodide, all you need to do is import and start using them:

```
%% py
import numpy as np
np.arange(10)
```

For most uses, that is all you need to know.

However, if you want to use your own custom package or load a package from another provider, you'll need to use the `pyodide.loadPackage` function from a Javascript chunk. For example, to load a special distribution of Numpy from `custom.com`:

```
%% js
pyodide.loadPackage('https://custom.com/numpy.js')
```

After doing that, the `numpy` you import from a Python chunk will be this special version of Numpy.

1.1.3 Using a local build of Pyodide with Iodide

You may want to build a local copy of Pyodide with some changes and test it inside of Iodide.

By default, Iodide will use a copy of Pyodide deployed to Netlify. However, it will use locally-installed copy of Pyodide if `USE_LOCAL_PYODIDE` is set.

Set that environment variable in your shell:

```
export USE_LOCAL_PYODIDE=1
```

Then follow the building and running instructions for Iodide as usual.

Next, build Pyodide using the regular instructions in `./README.md`. Copy the contents of Pyodide's build directory to your Iodide checkout's `build/pyodide` directory:

```
mkdir $IODIDE_CHECKOUT/build/pyodide
cp $PYODIDE_CHECKOUT/build/* $IODIDE_CHECKOUT/build/pyodide
```

1.2 Using Pyodide from Javascript

This document describes using Pyodide directly from Javascript. For information about using Pyodide from Iodide, see [Using Pyodide from Iodide](#).

1.2.1 Startup

Include `pyodide.js` in your project.

The recommended way to include Pyodide in your project is to download a release from [here](#) and include the contents in your distribution, and import the `pyodide.js` file there from a `<script>` tag.

For prototyping purposes, you may also use the following CDN URL, though doing so is not recommended, since it isn't versioned and could change or be unstable at any time:

```
https://pyodide.cdn.iodide.io/pyodide.js
```

This file has a single `Promise` object which bootstraps the Python environment: `languagePluginLoader`. Since this must happen asynchronously, it is a `Promise`, which you must call `then` on to complete initialization. When the promise resolves, `pyodide` will have installed a namespace in global scope: `pyodide`.

```
languagePluginLoader.then(() => {
  // pyodide is now ready to use...
  console.log(pyodide.runPython('import sys\nsys.version'));
});
```

1.2.2 Running Python code

Python code is run using the `pyodide.runPython` function. It takes as input a string of Python code. If the code ends in an expression, it returns the result of the expression, converted to Javascript objects (See [type conversions](#)).

```
pyodide.runPython('import sys\nsys.version');
```

1.2.3 Loading packages

Only the Python standard library and `six` are available after importing Pyodide. To use other libraries, you'll need to load their package using `pyodide.loadPackage`. This downloads the file data over the network (as a `.data` and `.js` index file) and installs the files in the virtual filesystem.

Packages can be loaded by name, for those included in the official pyodide repository (e.g. `pyodide.loadPackage('numpy')`). It is also possible to load packages from custom URLs (e.g. `pyodide.loadPackage('https://foo/bar/numpy.js')`), in which case the URL must end with `<package-name>.js`.

When you request a package from the official repository, all of that package's dependencies are also loaded. Dependency resolution is not yet implemented when loading packages from custom URLs.

Multiple packages can also be loaded in a single call,

```
pyodide.loadPackage(['cycler', 'pytz'])
```

`pyodide.loadPackage` returns a Promise.

```
pyodide.loadPackage('matplotlib').then(() => {
  // matplotlib is now available
});
```

1.2.4 Complete example

TODO

1.3 Using Pyodide from a web worker

This document describes how to use pyodide to execute python scripts asynchronously in a web worker.

1.3.1 Startup

Setup your project to serve `webworker.js`. You should also serve `pyodide.js`, and all its associated `.asm.js`, `.data`, `.json`, and `.wasm` files as well, though this is not strictly required if `pyodide.js` is pointing to a site serving current versions of these files.

Update the `webworker.js` sample so that it has a valid URL for `pyodide.js`, and sets `self.languagePluginUrl` to the location of the supporting files.

In your application code create a web worker, and add listeners for `onerror` and `onmessage`.

Call `postMessage` on your web worker, passing an object with the key `python` containing the script to execute as a string. You may pass other keys in the data object. By default the web worker assigns these to its global scope so that they may be imported from `python`. The results are returned as the `results` key, or if an error was encountered, it is returned in the `error` key.

For example:

```
var pyodideWorker = new Worker('./webworker.js')

pyodideWorker.onerror = (e) => {
  console.log(`Error in pyodideWorker at ${e.filename}, Line: ${e.lineno}, ${e.
↪message}`)
}

pyodideWorker.onmessage = (e) => {
  const {results, error} = e.data
  if (results) {
    console.log('pyodideWorker return results: ', results)
  } else if (error) {
    console.log('pyodideWorker error: ', error)
  }
}

const data = {
  A_rank: [0.8, 0.4, 1.2, 3.7, 2.6, 5.8],
  python:
    'import statistics\n' +
    'from js import A_rank\n' +
    'statistics.stdev(A_rank)'
}

pyodideWorker.postMessage(data)
```

1.3.2 Loading packages

Packages referenced from your python script will be automatically downloaded the first time they are encountered. Please note that some of the larger packages such as `numpy` or `pandas` may take several seconds to load. Subsequent uses of these packages will not incur the download overhead of the first run, but there is still some time required for the `import` in python itself.

If you would like to pre-load some packages, or the automatic package loading does not work for you for some reason, you may modify the `webworker.js` source to load some specific packages as described in [Using Pyodide directly from Javascript](#).

For example, to always load packages `numpy` and `pytz`, you would insert the line `self.pyodide.loadPackage(['numpy', 'pytz']).then(() => {` as shown below:

```
self.languagePluginUrl = 'http://localhost:8000/'
importScripts('./pyodide.js')

var onmessage = function(e) { // eslint-disable-line no-unused-vars
  languagePluginLoader.then(() => {
```

(continues on next page)

(continued from previous page)

```

self.pyodide.loadPackage(['numpy', 'pytz']).then(() => {
  const data = e.data;
  const keys = Object.keys(data);
  for (let key of keys) {
    if (key !== 'python') {
      // Keys other than python must be arguments for the python script.
      // Set them on self, so that `from js import key` works.
      self[key] = data[key];
    }
  }
  self.pyodide.runPythonAsync(data.python, () => {})
  .then((results) => { self.postMessage({results}); })
  .catch((err) => {
    // if you prefer messages with the error
    self.postMessage({error : err.message});
    // if you prefer onerror events
    // setTimeout(() => { throw err; });
  });
});
});
}

```

1.3.3 Caveats

Using a web worker is advantageous because the python code is run in a separate thread from your main UI, and hence does not impact your application's responsiveness. There are some limitations, however. At present, Pyodide does not support sharing the Python interpreter and packages between multiple web workers or with your main thread. Since web workers are each in their own virtual machine, you also cannot share globals between a web worker and your main thread. Finally, although the web worker is separate from your main thread, the web worker is itself single threaded, so only one python script will execute at a time.

1.4 Installing packages from PyPI

Pyodide has experimental support for installing pure Python wheels from PyPI.

IMPORTANT: Since the packages hosted at `files.pythonhosted.org` don't support CORS requests, we use a CORS proxy at `cors-anywhere.herokuapp.com` to get package contents. This makes a man-in-the-middle attack on the package contents possible. However, this threat is minimized by the fact that the integrity of each package is checked using a hash obtained directly from `pypi.org`. We hope to have this improved in the future, but for now, understand the risks and don't use any sensitive data with the packages installed using this method.

For use in Iodide:

```

%% py
import micropip
micropip.install('snowballstemmer')

# Iodide implicitly waits for the promise to resolve when the packages have finished
# installing...

%% py
import snowballstemmer

```

(continues on next page)

(continued from previous page)

```
snowballstemmer.stemmer('english')
stemmer.stemWords('go goes going gone'.split())
```

For use outside of Iodide (just Python), you can use the `then` method on the Promise that `micropip.install` returns to do work once the packages have finished loading:

```
def do_work(*args):
    import snowballstemmer
    snowballstemmer.stemmer('english')
    stemmer.stemWords('go goes going gone'.split())

import micropip
micropip.install('snowballstemmer').then(do_work)
```

1.5 API Reference

pyodide version 0.1.0

Backward compatibility of the API is not guaranteed at this point.

1.5.1 Python API

pyodide.open_url(url)

Fetches a given *url* and returns a `io.StringIO` to access its contents.

Parameters

name	type	description
<i>url</i>	str	the URL to open

Returns

A `io.StringIO` object with the URL contents./

pyodide.eval_code(code, ns)

Runs a string of code. The last part of the string may be an expression, in which case, its value is returned.

This function may be overridden to change how `pyodide.runPython` interprets code, for example to perform some preprocessing on the Python code first.

Parameters

name	type	description
<i>code</i>	str	the code to evaluate
<i>ns</i>	dict	evaluation name space

Returns

Either the resulting object or `None`.

pyodide.as_nested_list(obj)

Converts Javascript nested arrays to Python nested lists. This conversion can not be performed automatically, because Javascript Arrays and Objects can be combined in ways that are ambiguous.

Parameters

name	type	description
<i>obj</i>	JS Object	The object to convert

Returns

The object as nested Python lists.

1.5.2 Javascript API

pyodide.loadPackage(names)

Load a package or a list of packages over the network.

This makes the files for the package available in the virtual filesystem. The package needs to be imported from Python before it can be used.

Parameters

name	type	description
<i>names</i>	{String, Array}	package name, or URL. Can be either a single element, or an array.
<i>messageCallback</i>	function	A callback, called with progress messages. (optional)

Returns

Loading is asynchronous, therefore, this returns a Promise.

pyodide.loadedPackages

Array with loaded packages.

Use `Object.keys(pyodide.loadedPackages)` to access the names of the loaded packages, and `pyodide.loadedPackages[package_name]` to access install location for a particular `package_name`.

pyodide.pyimport(name)

Access a Python object from Javascript. The object must be in the global Python namespace.

For example, to access the `foo` Python object from Javascript:

```
var foo = pyodide.pyimport('foo')
```

Parameters

name	type	description
<i>names</i>	String	Python variable name

Returns

name	type	description
<i>object</i>	<i>any</i>	If one of the basic types (string, number, boolean, array, object), the Python object is converted to Javascript and returned. For other types, a Proxy object to the Python object is returned.

pyodide.globals

An object whose attributes are members of the Python global namespace. This is a more convenient alternative to `pyodide.pyimport`.

For example, to access the `foo` Python object from Javascript:

```
pyodide.globals.foo
```

pyodide.repr(obj)

Gets the Python's string representation of an object.

This is equivalent to calling `repr(obj)` in Python.

Parameters

name	type	description
<i>obj</i>	<i>any</i>	Input object

Returns

name	type	description
<i>str_repr</i>	String	String representation of the input object

pyodide.runPython(code)

Runs a string of code. The last part of the string may be an expression, in which case, its value is returned.

Parameters

name	type	description
<i>code</i>	String	Python code to evaluate

Returns

name	type	description
<i>jsresult</i>	<i>any</i>	Result, converted to Javascript

pyodide.runPythonAsync(code, messageCallback)

Runs Python code, possibly asynchronously loading any known packages that the code chunk imports.

For example, given the following code chunk

```
import numpy as np
x = np.array([1, 2, 3])
```

pyodide will first call `pyodide.loadPackage(['numpy'])`, and then run the code chunk, returning the result. Since package fetching must happen asynchronously, this function returns a `Promise` which resolves to the output. For example, to use:

```
pyodide.runPythonAsync(code, messageCallback)
  .then((output) => handleOutput(output))
```

Parameters

name	type	description
<code>code</code>	String	Python code to evaluate
<code>messageCallback</code>	function	Callback given status messages (optional)

Returns

name	type	description
<code>result</code>	Promise	Resolves to the result of the code chunk

pyodide.version()

Returns the pyodide version.

It can be either the exact release version (e.g. `0.1.0`), or the latest release version followed by the number of commits since, and the git hash of the current commit (e.g. `0.1.0-1-bd84646`).

Parameters

None

Returns

name	type	description
<code>version</code>	String	Pyodide version string

The Development section help Pyodide contributors to find information about the development process including making packages to support third party libraries and understanding type conversions between Python and JavaScript. The Project section helps contributors get started and gives additional information about the project's organization.

2.1 Creating a Pyodide package

Pyodide includes a set of automatic tools to make it easier to add new third-party Python libraries to the build.

These tools automate the following steps to build a package:

- Download a source tarball (usually from PyPI)
- Confirm integrity of the package by comparing it to a checksum
- Apply patches, if any, to the source distribution
- Add extra files, if any, to the source distribution
- If the package includes C/C++/Cython extensions:
 - Build the package natively, keeping track of invocations of the native compiler and linker
 - Rebuild the package using emscripten to target WebAssembly
- If the package is pure Python:
 - Run the `setup.py` script to get the built package
- Package the results into an emscripten virtual filesystem package, which comprises:
 - A `.data` file containing the file contents of the whole package, concatenated together
 - A `.js` file which contains metadata about the files and installs them into the virtual filesystem.

Lastly, a `packages.json` file is output containing the dependency tree of all packages, so `pyodide.loadPackage` can load a package's dependencies automatically.

2.1.1 The meta.yaml file

Packages are defined by writing a `meta.yaml` file. The format of these files is based on the `meta.yaml` files used to build [Conda packages](#), though it is much more limited. The most important limitation is that Pyodide assumes there will only be one version of a given library available, whereas Conda allows the user to specify the versions of each package that they want to install. Despite the limitations, keeping the file format as close as possible to conda's should make it easier to use existing conda package definitions as a starting point to create Pyodide packages. In general, however, one should not expect Conda packages to “just work” with Pyodide. (In the longer term, Pyodide may use conda as its packaging system, and this should hopefully ease that transition.)

There is a helper tool that will generate a `meta.yaml` for packages on PyPI that will work for many pure Python packages. This tool will populate the latest version, download link and sha256 hash by querying PyPI. It doesn't currently handle package dependencies. To run it, do:

```
bin/pyodide mkpkg $PACKAGE_NAME
```

The supported keys in the `meta.yaml` file are described below.

package

package/name

The name of the package. It must match the name of the package used when expanding the tarball, which is sometimes different from the name of the package in the Python namespace when installed. It must also match the name of the directory in which the `meta.yaml` file is placed. It can only contain alpha-numeric characters and `-`, `_`.

package/version

The version of the package.

source

source/url

The url of the source tarball.

The tarball may be in any of the formats supported by Python's `shutil.unpack_archive`: `tar`, `gztar`, `bztar`, `xztar`, and `zip`.

source/path

Alternatively to `source/url`, a relative or absolute path can be specified as package source. This is useful for local testing or building packages which are not available online in the required format.

If a path is specified, any provided checksums are ignored.

source/md5

The MD5 checksum of the tarball. It is recommended to use SHA256 instead of MD5. At most one checksum entry should be provided per package.

source/sha256

The SHA256 checksum of the tarball. It is recommended to use SHA256 instead of MD5. At most one checksum entry should be provided per package.

source/patches

A list of patch files to apply after expanding the tarball. These are applied using `patch -p1` from the root of the source tree.

source/extras

Extra files to add to the source tree. This should be a list where each entry is a pair of the form `(src, dst)`. The `src` path is relative to the directory in which the `meta.yaml` file resides. The `dst` path is relative to the root of source tree (the expanded tarball).

build**build/skip_host**

Skip building C extensions for the host environment. Default: `True`.

Setting this to `False` will result in ~2x slower builds for packages that include C extensions. It should only be needed when a package is a build time dependency for other packages. For instance, `numpy` is imported during installation of `matplotlib`, importing `numpy` also imports included C extensions, therefore it is built both for host and target.

build/cflags

Extra arguments to pass to the compiler when building for WebAssembly.

(This key is not in the Conda spec).

build/ldflags

Extra arguments to pass to the linker when building for WebAssembly.

(This key is not in the Conda spec).

build/post

Shell commands to run after building the library. These are run inside of `bash`, and there are two special environment variables defined:

- `$SITEPACKAGES`: The `site-packages` directory into which the package has been installed.
- `$PKGDIR`: The directory in which the `meta.yaml` file resides.

(This key is not in the Conda spec).

requirements

requirements/run

A list of required packages.

(Unlike conda, this only supports package names, not versions).

2.2 Type conversions

Python to Javascript conversions occur:

- when returning the final expression from a `pyodide.runPython` call (evaluating a Python cell in Iodide)
- using `pyodide.pyimport`
- passing arguments to a Javascript function from Python

Javascript to Python conversions occur:

- when using the `from js import ...` syntax
- returning the result of a Javascript function to Python

2.2.1 Basic types

The following basic types are implicitly converted between Javascript and Python. The values are copied and any connection to the original object is lost.

Python	Javascript
int, float	Number
str	String
True	true
False	false
None	undefined, null
list, tuple	Array
dict	Object

2.2.2 Typed arrays

Javascript typed arrays (`Int8Array` and friends) are converted to Python `memoryviews`. This happens with a single binary memory copy (since Python can't access arrays on the Javascript heap), and the data type is preserved. This makes it easy to correctly convert it to a Numpy array using `numpy.asarray`:

```
array = Float32Array([1, 2, 3])
```

```
from js import array
import numpy as np
numpy_array = np.asarray(array)
```

Python `bytes` and `buffer` objects are converted to Javascript as `Uint8ClampedArrays`, without any memory copy at all, and is thus very efficient, but be aware that any changes to the buffer will be reflected in both places.

Numpy arrays are currently converted to Javascript as nested (regular) Arrays. A more efficient method will probably emerge as we decide on an ndarray implementation for Javascript.

2.2.3 Class instances

Any of the types not listed above are shared between languages using proxies that allow methods and some operators to be called on the object from the other language.

Javascript from Python

When passing a Javascript object to Python, an extension type is used to delegate Python operations to the Javascript side. The following operations are currently supported. (More should be possible in the future – work in ongoing to make this more complete):

Python	Javascript
<code>repr(x)</code>	<code>x.toString()</code>
<code>x.foo</code>	<code>x.foo</code>
<code>x.foo = bar</code>	<code>x.foo = bar</code>
<code>del x.foo</code>	<code>delete x.foo</code>
<code>x(...)</code>	<code>x(...)</code>
<code>x.foo(...)</code>	<code>x.foo(...)</code>
<code>X.new(...)</code>	<code>new X(...)</code>
<code>len(x)</code>	<code>x.length</code>
<code>x[foo]</code>	<code>x[foo]</code>
<code>x[foo] = bar</code>	<code>x[foo] = bar</code>
<code>del x[foo]</code>	<code>delete x[foo]</code>
<code>x == y</code>	<code>x == y</code>
<code>x.typeof</code>	<code>typeof x</code>

One important difference between Python objects and Javascript objects is that if you access a missing member in Python, an exception is raised. In Javascript, it returns undefined. Since we can't make any assumptions about whether the Javascript member is missing or simply set to undefined, Python mirrors the Javascript behavior. For example:

```
// Javascript
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }
}
point = new Point(42, 43)
```

```
# python
from js import point
assert point.y == 43
del point.y
assert point.y is None
```

Python from Javascript

When passing a Python object to Javascript, the Javascript [Proxy API](#) is used to delegate Javascript operations to the Python side. In general, the Proxy API is more limited than what can be done with a Python extension, so there are certain operations that are impossible or more cumbersome when using Python from Javascript than vice versa. The most notable limitation is that while Python has distinct ways of accessing attributes and items (`x.foo` and `x[foo]`), Javascript conflates these two concepts. The following operations are currently supported:

Javascript	Python
<code>foo in x</code>	<code>hasattr(x, 'foo')</code>
<code>x.foo</code>	<code>getattr(x, 'foo')</code>
<code>x.foo = bar</code>	<code>setattr(x, 'foo', bar)</code>
<code>delete x.foo</code>	<code>delattr(x, 'foo')</code>
<code>x.ownKeys()</code>	<code>dir(x)</code>
<code>x(...)</code>	<code>x(...)</code>
<code>x.foo(...)</code>	<code>x.foo(...)</code>

An additional limitation is that when passing a Python object to Javascript, there is no way for Javascript to automatically garbage collect that object. Therefore, custom Python objects must be manually destroyed when passed to Javascript, or they will leak. To do this, call `.destroy()` on the object, after which Javascript will no longer have access to the object.

```
var foo = pyodide.pyimport('foo');
foo.call_method();
foo.destroy();
foo.call_method(); // This will raise an exception, since the object has been
                  // destroyed
```

2.2.4 Using Python objects from Javascript

A Python object (in global scope) can be brought over to Javascript using the `pyodide.pyimport` function. It takes a string giving the name of the variable, and returns the object, converted to Javascript (See [type conversions](#)).

```
var sys = pyodide.pyimport('sys');
```

2.2.5 Using Javascript objects from Python

Javascript objects can be accessed from Python using the special `js` module. This module looks up attributes of the global (`window`) namespace on the Javascript side.

```
import js
js.document.title = 'New window title'
```

Performance considerations

Looking up and converting attributes of the `js` module happens dynamically. In most cases, where the value is small or results in a proxy, this is not an issue. However, if the value takes a long time to convert from Javascript to Python, you may want to store it in a Python variable or use the `from js import ...` syntax.

For example, given this large Javascript variable:

```
var x = new Array(1000).fill(0)
```

Use it from Python as follows:

```
import js
x = js.x # conversion happens once here
for i in range(len(x)):
    item = x[i] # we don't pay the conversion price each time here
```

Or alternatively:

```
from js import x # conversion happens once here
for i in range(len(x)):
    item = x[i] # we don't pay the conversion price each time here
```

Important: From your browser, you can try Pyodide in an Iodide notebook. The Iodide documentation site provides additional user and developer documentation.

2.3 About the Project

*****What is Pyodide?**** | *****Try Pyodide***** | *****Getting Started***** | *****Contributing***** | *****License*****

2.4 Pyodide



The Python scientific stack, compiled to WebAssembly.

*****Try Pyodide and Iodide in your browser*****

2.4.1 What is Pyodide?

Pyodide brings the Python runtime to the browser via WebAssembly, along with the Python scientific stack including NumPy, Pandas, Matplotlib, parts of SciPy, and NetworkX. The ‘*packages*’ directory <<https://github.com/iodide-project/pyodide/tree/master/packages>> lists over 35 packages which are currently available.

Pyodide provides transparent conversion of objects between Javascript and Python. When used inside a browser, Python has full access to the Web APIs.

While closely related to the [iodide project](#), a tool for *literate scientific computing and communication for the web*, Pyodide goes beyond running in a notebook environment. To maximize the flexibility of the modern web, **Pyodide** may be used standalone in any context where you want to **run Python inside a web browser**.

2.4.2 Try Pyodide (no installation needed)

For more information, try [the demo](#) and look through the [documentation](#).

2.4.3 Getting Started

Pyodide offers three different ways to get started depending on your needs and technical resources. These include:

- *Download a pre-built version* (the quickest way to get started)
- *Build Pyodide from source* (this method requires installing prerequisites and using `make`. Primarily for Linux users who want to experiment or contribute back to the project.)
- *Use a Docker image* (recommended for Windows and macOS users and for Linux users who prefer a Debian-based Docker image on Docker Hub with the dependencies already installed)

Download a pre-built version

Pre-built versions of Pyodide may be downloaded from this repository's [releases page](#).

Building from source

Building is easiest on Linux. For other platforms, we recommend using the Docker image (described below) to build Pyodide.

Make sure the prerequisites for `emscripten` are installed. Pyodide will build a custom, patched version of `emscripten`, so there is no need to build it yourself prior.

Additional build prerequisites are:

- A working native compiler toolchain, enough to build CPython.
- A native Python 3.7 to run the build scripts.
- PyYAML
- `lessc` to compile less to css.
- `uglifyjs` to minify Javascript builds.
- `ccache` (optional) recommended for much faster rebuilds.

Build using `make`

After installing the build prerequisites, run from the command line:

```
make
```

Using Docker

We provide a Debian-based Docker image on Docker Hub with the dependencies already installed to make it easier to build Pyodide.

1. Install Docker
2. From a git checkout of Pyodide, run `./run_docker`
3. Run `make` to build.

If running `make` deterministically stops at one point in each subsequent try, increasing the maximum RAM usage available to the docker container might help [This is different from the physical RAM capacity inside the system]. Ideally, at least 3 GB of RAM should be available to the docker container to build `pyodide` smoothly. These settings can be changed via Docker Preferences [See [here](#)].

You can edit the files in your source checkout on your host machine, and then repeatedly run `make` inside the Docker environment to test your changes.

2.4.4 Contributing

Please view the [CONTRIBUTING](#) document for tips on filing issues, making changes, and submitting pull requests. The following sections describe how to run tests, run Pyodide benchmarks, and lint the source code.

Testing

Install the following dependencies into the default Python installation:

```
pip install pytest selenium pytest-instafail
```

Install [geckodriver](#) and [chromedriver](#) and check that they are in your `PATH`.

Automated testing

To run the `pytest` suite of tests, type on the command line:

```
pytest test/ packages/
```

Manual interactive testing

To run manual interactive tests, a docker environment and a webserver will be used.

1. Bind port 8000 for testing. To automatically bind port 8000 of the docker environment and the host system, run:
`./run_docker`
2. Now, this can be used to test the `pyodide` builds running within the docker environment using external browser programs on the host system. To do this, run: `./bin/pyodide serve`
3. This serves the `build` directory of the `pyodide` project on port 8000.
 - To serve a different directory, use the `--build_dir` argument followed by the path of the directory.
 - To serve on a different port, use the `--port` argument followed by the desired port number. Make sure that the port passed in `--port` argument is same as the one defined as `DOCKER_PORT` in the `run_docker` script.
4. Once the webserver is running, simple interactive testing can be run by visiting this URL: <http://localhost:8000/console.html>

Benchmarking

To run common benchmarks to understand Pyodide's performance, begin by installing the same prerequisites as for testing. Then run:

```
make benchmark
```

Linting

Python is linted with `flake8`. C and Javascript are linted with `clang-format`.

To lint the code, run:

```
make lint
```

2.4.5 License

Pyodide uses the Mozilla Public License Version 2.0. See the [LICENSE](#) file for more details.

*****What is Pyodide?**** | *****Try Pyodide***** | *****Getting Started***** | *****Contributing***** | *****License***** | *****Back to top*****

2.5 How to Contribute

Thank you for your interest in contributing to PYODIDE! There are many ways to contribute, and we appreciate all of them. Here are some guidelines & pointers for diving into it.

2.5.1 Code of Conduct

PYODIDE has adopted a [Code of Conduct](#) that we expect all contributors and core members to adhere to.

2.5.2 Development

Work on PYODIDE happens on Github. Core members and contributors can make Pull Requests to fix issues and add features, which all go through the same review process. We'll detail how you can start making PRs below.

We'll do our best to keep `master` in a non-breaking state, ideally with tests always passing. The unfortunate reality of software development is sometimes things break. As such, `master` cannot be expected to remain reliable at all times. We recommend using the latest stable version of PYODIDE.

PYODIDE follows semantic versioning (<http://semver.org/>) - major versions for breaking changes (x.0.0), minor versions for new features (0.x.0), and patches for bug fixes (0.0.x).

We keep a file, [CHANGELOG.md](#), outlining changes to PYODIDE in each release. We like to think of the audience for changelogs as non-developers who primarily run the latest stable. So the change log will primarily outline user-visible changes such as new features and deprecations, and will exclude things that might otherwise be inconsequential to the end user experience, such as infrastructure or refactoring.

2.5.3 Bugs & Issues

We use [Github Issues](#) for announcing and discussing bugs and features. Use [this link](#) to report a bug or issue. We provide a template to give you a guide for how to file optimally. If you have the chance, please search the existing issues before reporting a bug. It's possible that someone else has already reported your error. This doesn't always work, and

sometimes it's hard to know what to search for, so consider this extra credit. We won't mind if you accidentally file a duplicate report.

Core contributors are monitoring new issues & comments all the time, and will label & organize issues to align with development priorities.

2.5.4 How to Contribute

Pull requests are the primary mechanism we use to change PYODIDE. GitHub itself has some [great documentation](#) on using the Pull Request feature. We use the “fork and pull” model [described here](#), where contributors push changes to their personal fork and create pull requests to bring those changes into the source repository.

Please make pull requests against the `master` branch.

If you're looking for a way to jump in and contribute, our list of “*good first issues*” [_](https://github.com/iodide-project/pyodide/labels/good%20first%20issue) is a great place to start.

If you'd like to fix a currently-filed issue, please take a look at the comment thread on the issue to ensure no one is already working on it. If no one has claimed the issue, make a comment stating you'd like to tackle it in a PR. If someone has claimed the issue but has not worked on it in a few weeks, make a comment asking if you can take over, and we'll figure it out from there.

We use `py.test`, driving [Selenium](#) as our testing framework. Every PR will automatically run through our tests, and our test framework will alert you on Github if your PR doesn't pass all of them. If your PR fails a test, try to figure out whether or not you can update your code to make the test pass again, or ask for help. As a policy we will not accept a PR that fails any of our tests, and will likely ask you to add tests if your PR adds new functionality. Writing tests can be scary, but they make open-source contributions easier for everyone to assess. Take a moment and look through how we've written our tests, and try to make your tests match. If you are having trouble, we can help you get started on our test-writing journey.

All code submissions should pass `make lint`. Python is checked with the default settings of `flake8`. C and Javascript are checked against the Mozilla style in `clang-format`.

2.5.5 Development Workflow

See `README.md`.

2.5.6 License

All contributions to PYODIDE will be licensed under the [Mozilla Public License 2.0 \(MPL 2.0\)](#). This is considered a “weak copyleft” license. Check out the [tdrLegal](#) entry for more information, as well as Mozilla's [MPL 2.0 FAQ](#) if you need further clarification on what is and isn't permitted.

2.5.7 Get in Touch

- **Gitter:** Pyodide currently shares the [#iodide](#) channel over at [gitter.im](#)

2.6 IODIDE CODE of CONDUCT

2.6.1 CONDUCT

We are committed to providing a friendly, safe and welcoming environment for all, regardless of level of experience, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, ethnicity, age, religion, nationality, or other similar characteristic.

- Please be kind and courteous. There's no need to be mean or rude.
- On IRC & any other IODIDE communication venue, please avoid using overtly sexual nicknames or other nicknames that might detract from a friendly, safe and welcoming environment for all.
- Respect that people have differences of opinion and that every design or implementation choice carries a trade-off and numerous costs. There is seldom a right answer.
- There are many unproductive habits that should be avoided in communication. We borrow the Recurse Center's "social rules": no feigning surprise, no well-actually's, no backseat driving, and no subtle -isms. Read the preceding link for further discussion.
- Please keep unstructured critique to a minimum. If you have solid ideas you want to experiment with, make a fork and see how it works. All feedback should be constructive in nature. If you need more detailed guidance around giving feedback, consult [Digital Ocean's Code of Conduct](#).
- We will exclude you from interaction if you insult, demean or harass anyone. That is not welcome behavior. We interpret the term "harassment" as including the definition in the [Citizen Code of Conduct](#); if you have any lack of clarity about what might be included in that concept, please read their definition. In particular, we don't tolerate behavior that excludes people in socially marginalized groups.
- Private harassment is also unacceptable. No matter who you are, if you feel you have been or are being harassed or made uncomfortable by a community member, please contact one of the channel ops or any of the IODIDE core team members immediately. Whether you're a regular contributor or a newcomer, we care about making this community a safe place for you and we've got your back.
- Likewise any spamming, trolling, flaming, baiting or other attention-stealing behavior is not welcome.

2.6.2 MODERATION

These are the policies for upholding our community's standards of conduct. If you feel that a thread needs moderation, please contact the IODIDE core team.

1. Remarks that violate the IODIDE standards of conduct, including hateful, hurtful, oppressive, or exclusionary remarks, are not allowed. (Cursing is allowed, but never targeting another user, and never in a hateful manner.)
2. Remarks that moderators find inappropriate, whether listed in the code of conduct or not, are also not allowed.
3. Moderators will first respond to such remarks with a warning.
4. If the warning is unheeded, the user will be "kicked," i.e., kicked out of the communication channel to cool off.
5. If the user comes back and continues to make trouble, they will be banned, i.e., indefinitely excluded.
6. Moderators may choose at their discretion to un-ban the user if it was a first offense and they offer the offended party a genuine apology.
7. If a moderator bans someone and you think it was unjustified, please take it up with that moderator, or with a different moderator, in private. Complaints about bans in-channel are not allowed.
8. Moderators are held to a higher standard than other community members. If a moderator creates an inappropriate situation, they should expect less leeway than others.

9. In the IODIDE community we strive to go the extra step to look out for each other. Don't just aim to be technically unimpeachable, try to be your best self. In particular, avoid flirting with offensive or sensitive issues, particularly if they're off-topic; this all too often leads to unnecessary fights, hurt feelings, and damaged trust; worse, it can drive people away from the community entirely.
10. And if someone takes issue with something you said or did, resist the urge to be defensive. Just stop doing what it was they complained about and apologize. Even if you feel you were misinterpreted or unfairly accused, chances are good there was something you could've communicated better — remember that it's your responsibility to make your fellow IODIDE community members comfortable. Everyone wants to get along and we are all here first and foremost because we want to talk about science and cool technology. You will find that people will be eager to assume good intent and forgive as long as you earn their trust.
11. The enforcement policies listed above apply to all official IODIDE venues. If you wish to use this code of conduct for your own project, consider explicitly mentioning your moderation policy or making a copy with your own moderation policy so as to avoid confusion.

Adapted from the the 'Rust Code of Conduct <<https://www.rust-lang.org/en-US/conduct.html>>', with further reference from 'Digital Ocean Code of Conduct <<https://github.com/digitalocean/engineering-code-of-conduct#giving-and-receiving-feedback>>', the 'Recurse Center <<https://www.recurse.com/code-of-conduct>>', the 'Citizen Code of Conduct <<http://citizencodeofconduct.org/>>', and the 'Contributor Covenant <<https://www.contributor-covenant.org/version/1/4/code-of-conduct.html>>'.

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`