1 Demo Video 3
2 Inspiration 5
3 Installing 7
4 Full Content 9
  4.1 Commands .................................................. 9
    4.1.1 Displaying data ...................................... 9
    4.1.2 Using the osg viewer in an external app ......... 10
    4.1.3 Adding files to monitor for change ............... 10
  4.2 Jarvis emacs bindings .................................. 10
    4.2.1 Basic functions ..................................... 10
    4.2.2 Code interaction .................................... 11
    4.2.3 Creating new commands .............................. 11
    4.2.4 Misc .................................................. 11
Jarvis is a Python coding companion. Point it to a python function, and it will execute it. As soon as you change and save your code, Jarvis will detect it, and will rerun the function.

If an exception is raised, it will be displayed in the error panel.

If you insert some debugging statements in your code, they will be displayed in the debug panel.

Last, but not least, if you are using OpenSceneGraph Python bindings, you will be able to output an OSG tree to the Jarvis interface. This way, you can instantly see the new 3D scene your code is generating.
Demo Video

You can have a better description and a demo video on the Jarvis Front Page.
Jarvis was inspired by works of Bret Victor, especially his talk Inventing on Principle.

The central idea is that the feedback loop when you are coding should be the shortest possible, so you can see the effect of your code changes instantly, or almost. Jarvis implements a (small) subset of these ideas.

Those ideas are also used in the Light Table KickStarter project.
1. install qt, pyqt, osg, osgswig
2. install pymacs an jinja2 if you want to use emacs bindings
3. pip install jarvis
4. jarvis -filename_function my_python_file.py:main
5. Enjoy!
4.1 Commands

Jarvis commands are available through the jarvis.commands module.

```python
from jarvis.commands import debug, debug_xml

def main():
a = "Hello World !"
    debug(a)
```

4.1.1 Displaying data

These functions are used to display data in the Jarvis panels. All displayed strings are prefixed with a timestamp.

- `debug(*args)`: Display args separated by a blank space
- `debug_dir(object, filt)`: Display keys in dir(object) that contain the string “filt”, or all if filt is None
- `debug_xml(args)`: Display the n first arguments like debug would do, and the last one as an xml object
- `debug osg(osgdata)`: Display the osg tree “osgdata” in the osg panel
- `debug osg_set_loop_time(loop_time)`: Set the looping time for the osg view
- `error(*args)`: Display an error in the error panel. (If an exception is raised in your code, it will be displayed in any case)
- `testunit_result(result)`: Print an error in the error panel if any of the unittest result has an error or a failure

Here is a typical use of this function, given a unittest class called TestMyModule:

```python
import unittest

class TestMyModule(unittest.TestCase):
    ...
    def main():
        filt = "test_"
```
```python
suite = unittest.TestLoader().loadTestsFromTestCase(TestMyModule)
suite = filter(lambda x: str(x).startswith(filt), suite)
suite = unittest.TestLoader().suiteClass(suite)
result = unittest.TextTestRunner(verbosity=2).run(suite)
# Show the test unit in jarvis
testunit_result(result)
```

```python
reset_start_time():
    Used to reset the timer used to time debug displays
```

4.1.2 Using the osg viewer in an external app

```python
get osg_viewer():
    Return the osg viewer that is displayed in the osg panel
```

4.1.3 Adding files to monitor for change

```python
add_watch_file()
    Add a file to be watched by Jarvis. As soon as the file changes, the full code is reexecuted.
# Misc replace_this
```

4.2 Jarvis emacs bindings

Those bindings are available through Pymacs. You will have to install it if you want to use the jarvis commands in emacs. You will have to install jinja2 too.

Once Pymacs is installed, you will have to copy or require jarvis.el file in your own ~/.emacs init file. The typical content of this file is:

```emacs
;;; Initialize Jarvis
(pymacs-load "jarvis.emacs" "j-"
(global-set-key (kbd "C-x g") "j-goto-error"
(global-set-key (kbd "C-x i") "j-inspect-vars"
```

The first line tell pymacs to load jarvis.emacs and then to prefix the jarvis command with “j-” . You can change that, but I will assume that you are using this prefix below.

So, all the commands below will appear prefixed by j-, you will be able to see them using by typing “M-x j- TAB” in emacs once everything is installed correctly.

4.2.1 Basic functions

```python
launch:  Launch jarvis. It will start with the last known entry point.
```

```python
test_this  Tell Jarvis to load the current file, and then to run the main function.
```

```python
test_filename_function_set  The argument to this function should looks like file:function . Tell Jarvis to load the file, and then to run the function in this file.
```
4.2.2 Code interaction

inspect_vars  Request local variables in the current file at the current line. They will be displayed in a *
jarvis_inspect*
buffer.

goto_error  Go to the last error that was displayed in the error panel.
paste_debug_buffer  Paste the content of the debug panel at the current cursor position.

4.2.3 Creating new commands

The commands or snippets created by the following two functions will be created in your ~/.jarvis.d directory. They
will appear as new j- prefixed commands. You can modify those commands and test them inside emacs without
restarting jarvis, they are completely dynamic.

new_command  Create a new command. Will ask for a command name, and will write a file with a typical command
code, with the right filename.

new_snippet  Create a new command. Will ask for a command name, and will write a file with a typical command
code, with the right filename.

4.2.4 Misc

test_create  This will create a test file, given the current python file. It will look for a tests directory somewhere, or
will ask for one if not found.
google  Ask for a query then open a browser with google search.
Index

A

add_watch_file() (built-in function), 10