JÜLICH
FORSCHUNGSZENTRUM

Session 6: Introduction to advanced tools

October 9th, 2017 | Wouter Klijn

Overview

- Versioning (GIT)
- Tests
  - Types
  - How to start testing
  - Unittests
- Debugging
  - pdb
- Interactive Development Invironments

Git: Why

- Storage (backup) of source code file
- Who changed what when
- Undo / redo
- Facilitates working on multiple versions of a software
- Merge of changes from multiple developers

https://www.slideshare.net/phpcodemonkey/introduction-to-version-control-presentation

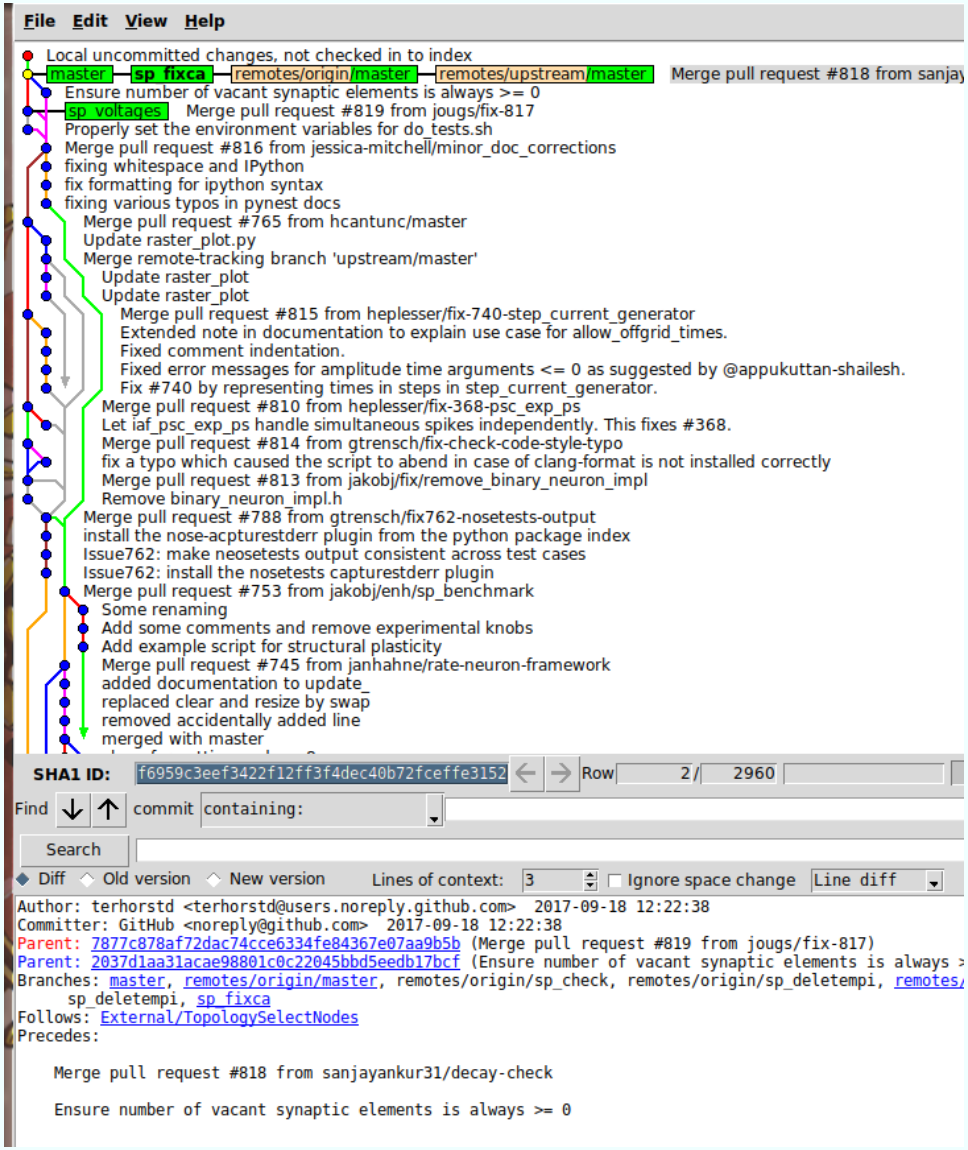
Git

- 70s software interface
- Command line with ‘intuitive‘ arguments
- Graphical user interfaces: Tortoise git (Windows), GitKraken (Linux)
- Integration in mature IDE’s: PyCharm, Visual studio, Eclipse

https://xkcd.com/1597/

Git

- clone
- checkout
- add
- commit
- fetch
- pull
- push
- remote
- branch



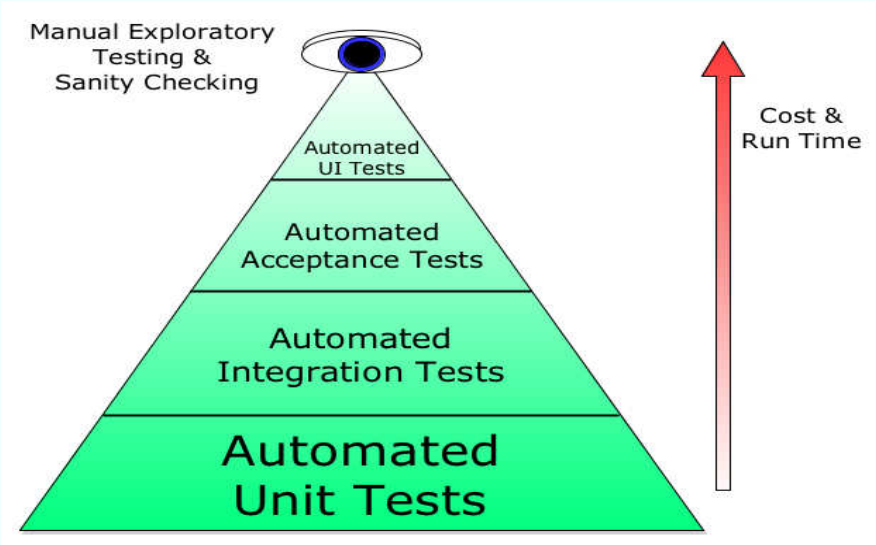
Testing

- Automatic programs or checklist assessing the correction functioning of software.
- Prevent introduction of errors when adding features.
- But also:
  - Tests as documentation
  - Leads to better design: loose coupling
  - In larger projects, improved development speed (mostly due to reduction in bugs to be solved)

Testing pyramid

- Major types of tests:
  - Manual testing
  - Data driven delta testing / regression testing
  - Component testing
  - Unit testing

The concepts are fuzzy and there is overlap and different names for the same thing



http://willhamill.com/2013/08/12/automated-testing-and-the-evils-of-ice-cream

How to start testing?

- Writing down the **manual tests** you already do
  - Doubles as documentation
- Create an **data driven delta** test
  - Create test data
  - Forces you to think about ‘user’ interactions
  - Doubles as introductory how-to
- Pick a single important **component** and disconnect it from the rest.
  - And continue doing this till you end up with:
- **Unit test** for small parts of the code that do one and only one thing.

Python: unittest

- Based on the xunit standard
- Setup -> test -> teardown
    1. Create files, etc. needed to run the component
    1. Run individual function an test the correct output eg:
        - assertEqual
        - assertTrue
        - assertExceptionThrown
    1. Delete used resources

http://pythontesting.net/framework/unittest/unittest-introduction/

Python: unittest

```
import unittest

def function(parameter):
    return parameter

class TestSomething(unittest.TestCase):

    def setUp(self):
        pass

    def test_fail(self):
        self.assertEqual(function(13), 12)

    def test_succes(self):
        self.assertEqual(function(12), 12)

    def tearDown(self):
        pass

if __name__ == '__main__':
    unittest.main()
```

```
wouter@WKLIJNWORK:/mnt/c/work$ python3 unittester.py
F.
======================================================================
FAIL: test_something_fail (__main__.TestSomething)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "unittester.py", line 15, in test_fail
    self.assertEqual(function(13), 12)
AssertionError: 13 != 12


----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

Debugging

- Debug print statements
- Use binairy search to find the problem. If you know your program this is often the fastest
- If the program is big, or not your own, it's a hard problem:

```
python –m pdb program.py
```

Debugging: pdb

Command	action
n	Execute the next command
enter	Repeat the last command
q	Hard exit (with a signal / exception)
p <var>,<var>	Print the value of the variable
c	Continue with program (until trace_point)
s	Step into a function
r	Continue till end of function
list <n1,n2>	Print surrounding code, include (n1, n2)

https://pythonconquerstheuniverse.wordpress.com/2009/09/10/debugging-in-python/

Debugging: pdb cont.

- PDB starts your program and halts at the first statement.
- For large programs you can add trace points:

```
import pdb
pdb.set_trace()
```

- Execution will drop into debugging mode

When doing interactive development:

- pdb.run('statement to evaluated')

https://pymotw.com/2/pdb/

Debugging: pdb advanced

Interactive development:

- pdb.run('statement to evaluated')

- Postmortem:
  - pdb.pm()
    "Debugging of the sys.last_backtrace"
  - Could be use in combination with except

- For more in-depth information:
  - https://pymotw.com/3/pdb/index.html

https://pymotw.com/2/pdb/

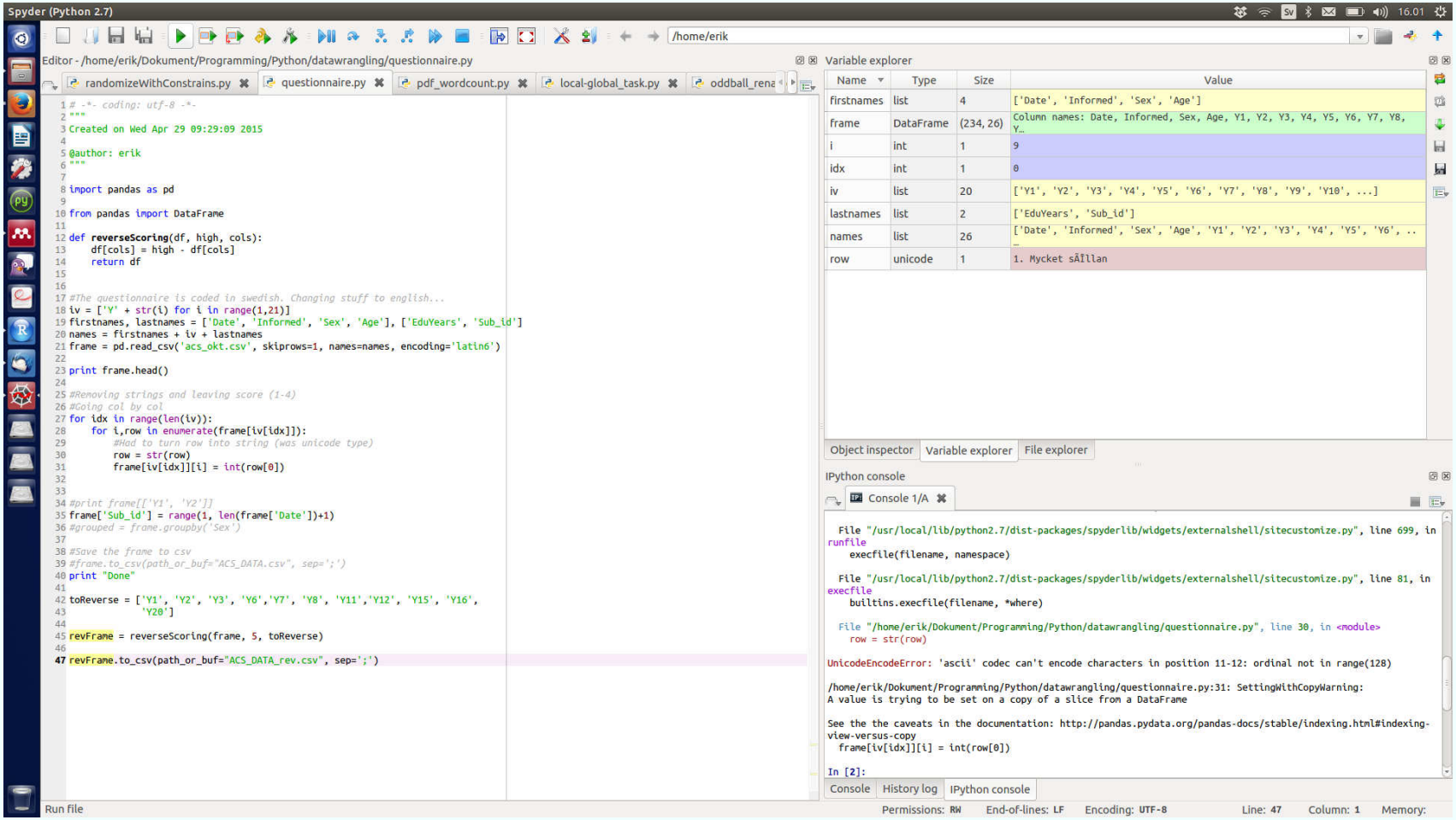
IDE

- The biggest difference between python and Matlab is the Integrated Development Environment (IDE)
- Python is typically interacted with via code or console.
- Selecting an IDE is an ‘important’ choice.
  - It takes time to get use to a IDE
  - Operating system
  - Features

IDE

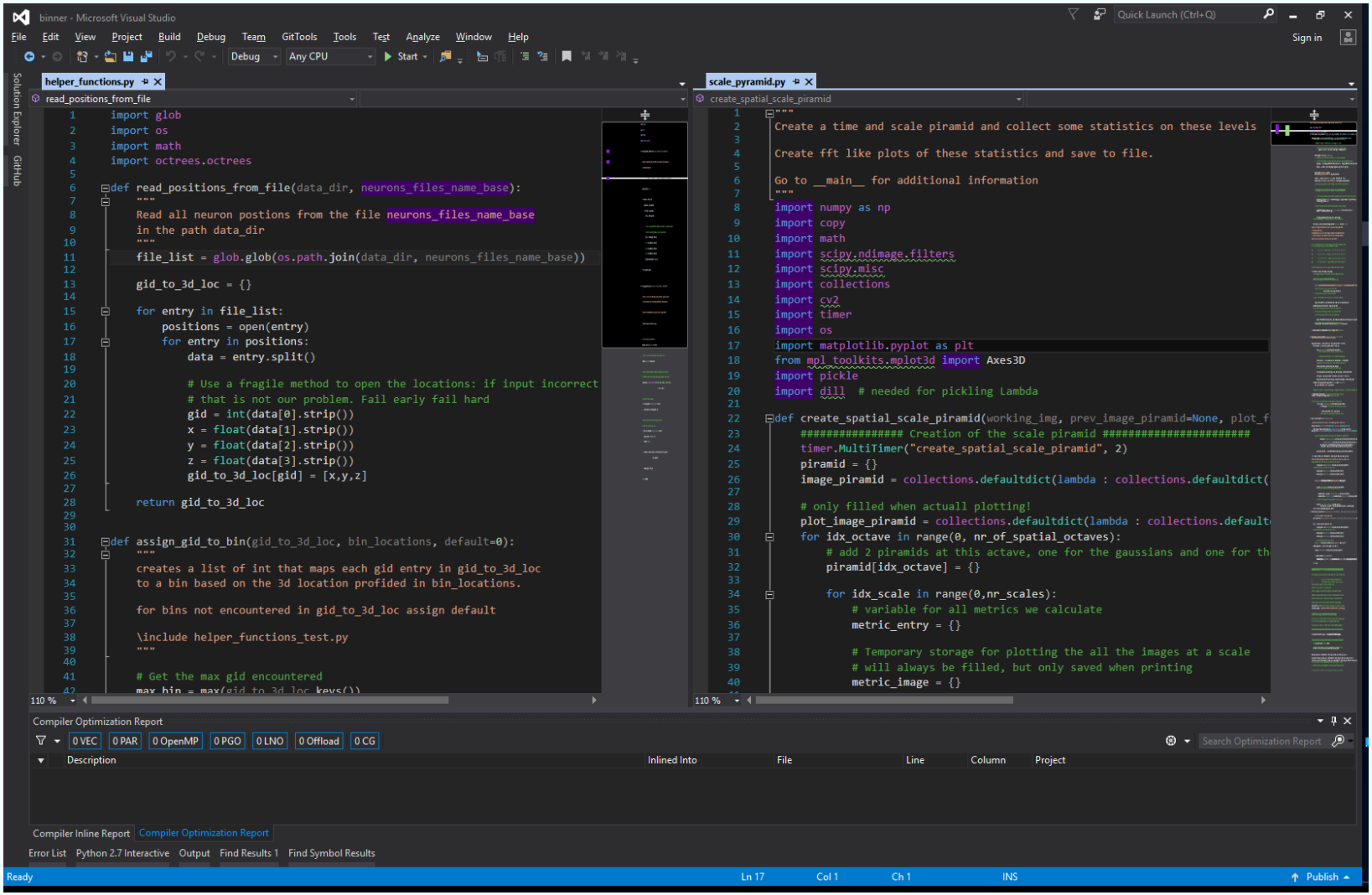
- Spyder: MATLAB like interface
  - Available on most operating systems
  - Python centric
- Visual Studio: python development tools
  - Windows
  - Prepared for later C++ development (Cython)
- Eclipse JAVA based but supports most languages
  - Available on most operating systems
  - Prepared for later C++ development
- PyCharm. Python centric IDE

IDE: Spyder

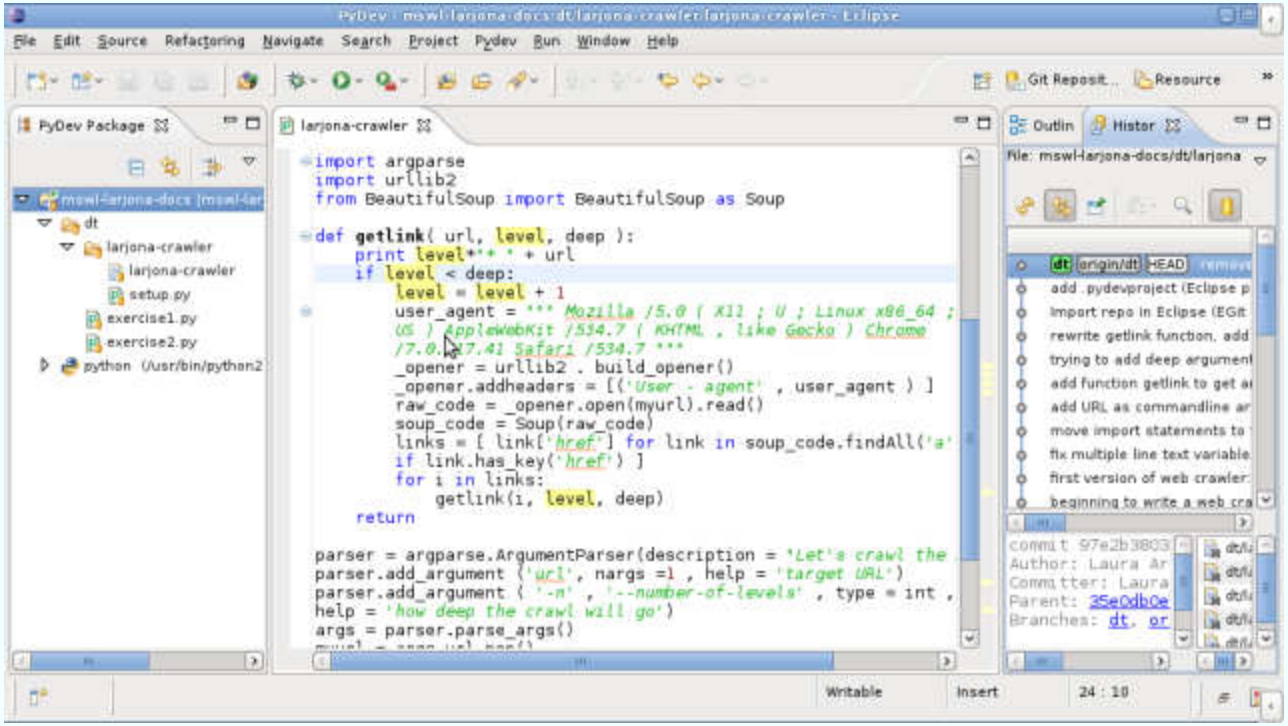


https://www.marsja.se/rstudio-like-python-ides-rodeo-spyder/

IDE: Visual Studio



IDE: Eclipse



https://larjona.wordpress.com/2011/09/27/first-steps-with-python-and-eclipse-ide/

Thank you for your attention




References and further reading: