Session 6: Introduction to advanced tools

October 9th, 2017 | Wouter Kijn
Overview

• Versioning (GIT)

• Tests
  • Types
  • How to start testing
  • Unit tests

• Debugging
  • `pdb`

• Interactive Development Environments
Git: Why

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

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

- Command line with ‘intuitive‘ arguments

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

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

09/10/2017
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
  2. Run individual function an test the correct output eg:
     - assertEquals
     - assertTrue
     - assertEqualsThrown
  3. Delete used resources

http://pythontesting.net/framework/unittest/unittest-introduction/
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_success(self):
        self.assertEqual(function(12), 12)

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

- Debug print statements

- Use binary search to find the problem. If you know your program this is often the fastest

- If the program is big, not yours, it’s a hard problem:

  python –m pdb program.py
Debugging: pdb

- Starts your program and halts at the first statement.

<table>
<thead>
<tr>
<th>Command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Execute the next command</td>
</tr>
<tr>
<td>enter</td>
<td>Repeat the last command</td>
</tr>
<tr>
<td>q</td>
<td>Hard exit (with a signal / exception)</td>
</tr>
<tr>
<td>p</td>
<td>Print the value of the variable</td>
</tr>
<tr>
<td>&lt;var&gt;,&lt;var&gt;</td>
<td>Continue with program (until trace_point)</td>
</tr>
<tr>
<td>c</td>
<td>Step into a function</td>
</tr>
<tr>
<td>r</td>
<td>Continue till end of function</td>
</tr>
<tr>
<td>list</td>
<td>Print surrounding code, include (n1, n2)</td>
</tr>
<tr>
<td>&lt;n1,n2&gt;</td>
<td><a href="https://pythonconquerstheuniverse.wordpress.com/2009/09/10/debugging-in-python/">Link</a></td>
</tr>
</tbody>
</table>
Debugging: pdb cont.

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

09/10/2017
Debugging: pdb advanced

Interactive development:
- `pdb.run('statement to evaluated')`
- Postmortem:
  - `pdb.pm()`

- For more in-depth information:
  - [https://pymotw.com/2/pdb/](https://pymotw.com/2/pdb/)
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 with 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
IDE: Spyder

https://www.marsja.se/rstudio-like-python-ides-rodeo-spyder/
IDE: Visual Studio
IDE: Eclipse

Thank you for your attention

References and further reading: