Unit Testing Workshop

Helen V Cook
November 27, 2014

DSB Group Meeting

Overview

- Lecture (30 min)
    - What is unit testing?
    - How it will make your life easier
    - Examples from the retreat
    - What to test, limitations, next steps
- Practice time (1 hr)
    - Write tests for find_level, or your own code (in Python)

Many kinds of testing

Most Specific	Does it return the right thing?	Unit testing
	Do all the parts work together?	
	…	
	Does it do what the requirements said it was supposed to?	
Most Broad	Does it fill the customer's need?	Acceptance testing

Unit testing

==

Test each subroutine

Aww, do I have to?

- Testing is just a waste of time.
- I’m writing throw away code, so I don’t have to.
- I'm not a developer writing a software project.
- I’m too good a programmer to need to write tests.

No, but…

Testing makes
your life easier

Makes your life easier

- Automation
- Ensure it works
- Write better code
- Reuse code
- Saves time

Automation

- Tests are an automated way of verifying the output is what you expect.
- No more manually running your code and manually inspecting the output.
- Will find bugs early, and save you time later.

Be sure it works

- Ensure that your code does what you think it does.
- Ensure new changes don’t break existing code.
- Easier to make big changes when you can verify your changes are correct.
- Finds problems early.

Write better code

- You will write cleaner interfaces.
- You will write shorter functions.
- Testing helps document your code.

Reuse code

- Write one well tested function, then never have to write it again.
- Code never really gets run only once anyway.
- When you revisit old code, the tests tell you what it's expected to do and what it actually does.

Testing saves time

- Stop writing code when the tests pass.
- Less throw away code means less code to write.
- Less manual work means less time spent on testing.
- Integration is now much easier.

How to write tests?

Testing Framework

- Python - unittest
- Perl - Test::More (Test::Simple)
- Javascript - Qunit (there are others)
- Frameworks exist for any language you would use: C++, java, bash, ruby, haskell, pl/SQL, R, LaTeX, ...

Your current workflow (?)

- Jump in and write code
- Run code, manually inspect output, identify bug
- Fix bug, write some more code
- Realize that previously working things no longer work
- Fix code, realize you need to refactor it
- Design the interface

Testing workflow

- Design the interface
- Write a small test
- Write just enough code to fail the test
- Run test (fails)
- Write just enough code to pass the test
- Run test (passes)

Example: PTMcons find level

- Have a file containing a level number (taxid) and a list of member taxids
- Have a list of member taxids as input: the query organism AND (level OR a list of organisms)
- Want to find the smallest level that contains them all.

API Design

- Read the levels file and store contents as a data structure we can query: **parse_levels_file**(levels_file)
- Core function, takes list of taxids and finds the smallest level: **smallest_group**(members, groups)
- Wrap input, error checking: **find_level**(query_org, search_level, search_orgs, levels_file)

t_find_level.py

```
import unittest

from find_level import *

class TestFindLevel (unittest.TestCase):
    levels_file = “data/eggnogv4.levels.txt"

    def test_parse_levels_file(self):
        levels = parse_levels_file(self.levels_file)
        self.assertTrue(levels, 'Levels file opened and read’)
        with self.assertRaises(IOError):
            parse_levels_file("file_does_not_exist")

    def test_smallest_group(self):
        levels = parse_levels_file(self.levels_file)
        self.assertEqual(smallest_group([240176, 486041, 578458], levels), 5338)


suite = unittest.TestLoader().loadTestsFromTestCase(TestFindLevel)
unittest.TextTestRunner(verbosity=2).run(suite)
```

t_find_level.py

```
import unittest

from find_level import *

class TestFindLevel (unittest.TestCase):
    levels_file = “data/eggnogv4.levels.txt"

    def test_parse_levels_file(self):
        levels = parse_levels_file(self.levels_file)
        self.assertTrue(levels, 'Levels file opened and read’)
        with self.assertRaises(IOError):
            parse_levels_file("file_does_not_exist")

    def test_smallest_group(self):
        levels = parse_levels_file(self.levels_file)
        self.assertEqual(smallest_group([240176, 486041, 578458], levels), 5338)


suite = unittest.TestLoader().loadTestsFromTestCase(TestFindLevel)
unittest.TextTestRunner(verbosity=2).run(suite)
```

t_find_level.py

```
import unittest

from find_level import *

class TestFindLevel (unittest.TestCase):
    levels_file = “data/eggnogv4.levels.txt"

    def test_parse_levels_file(self):
        levels = parse_levels_file(self.levels_file)
        self.assertTrue(levels)
        with self.assertRaises(IOError):
            parse_levels_file("file_does_not_exist")

    def test_smallest_group(self):
        levels = parse_levels_file(self.levels_file)
        self.assertEqual(smallest_group([240176, 486041, 578458], levels), 5338)


suite = unittest.TestLoader().loadTestsFromTestCase(TestFindLevel)
unittest.TextTestRunner(verbosity=2).run(suite)
```

find_level.py

```python
def parse_levels_file(levels_file):
    '''
    Parses the eggnog levels file that contains the definitions of orthologous
    groups and creates the levels dictionary, which it returns.

    @param levels_file filename of the eggnog levels file
    @return levels dictionary
    '''
    return False

def smallest_group(members, groups):
    '''
    Given a set of members, find the smallest group that contains them all.
    Members must be values in the groups dictionary.
    An exception will be thrown if a valid level can not be found.

    @param members A list of the members to find the smallest containing group for
    @param groups A dictionary with groups as the keys and members as values
    @return The smallest group containing all members
    '''
    return False
```

Doxygen

- For PTMcons, run `doxygen config.dox` in `ptmcons/doc/`
- Will autogenerate HTML documentation from commented functions in the source code!
- And will warn if parameters are not documented.

find_level.py

```
def parse_levels_file(levels_file):
    '''
    Parses the eggnog levels file that contains the definitions of orthologous
    groups and creates the levels dictionary, which it returns.

    @param levels_file filename of the eggnog levels file
    @return levels dictionary
    '''
    return False

def smallest_group(members, groups):
    '''
    Given a set of members, find the smallest group that contains them all.
    Members must be values in the groups dictionary.
    An exception will be thrown if a valid level can not be found.

    @param members A list of the members to find the smallest containing group for
    @param groups A dictionary with groups as the keys and members as values
    @return The smallest group containing all members
    '''
    return False
```

Run the tests (fail)

```
% python t_find_level.py

======================================================================
FAIL: test_parse_levels_file (__main__.TestFindLevel)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "t_find_level.py", line 10, in test_parse_levels_file
    self.assertTrue(levels)
AssertionError: False is not true


======================================================================
FAIL: test_smallest_group (__main__.TestFindLevel)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "t_find_level.py", line 16, in test_smallest_group
    self.assertEqual(smallest_group([240176, 486041, 578458], levels), 5338)
AssertionError: False != 5338
```

Edit find_level.py

```
def smallest_group(members, groups):
    minsize = float("inf")
    bestgroup = ''

    for g in groups:
        group_contents = get_members(g, groups)
        if set(members) == set(members).intersection(set(group_contents)):
            size = get_size(g, groups)
            if size < minsize:
                minsize = size
                bestgroup = g
    if bestgroup == '':
        raise Exception('Cannot find any valid group for members')
    else:
        return int(bestgroup)


… and so on …
```

Run the tests (pass)

```
% python t_find_level.py

test_parse_levels_file (__main__.TestFindLevel) ...
cannot open file_does_not_exist
ok

test_smallest_group (__main__.TestFindLevel) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.026s

OK
```

Bug fixing workflow

- Find bug in code
- Add tests to cover bug
- Fix bug
- Run tests (pass)

What to test?

- Happy case
  - Good input -> Good output
- Negative cases
  - Bad input -> Errors are handled
- Both are important

Negative tests

- Test common input errors (Null, negative taxid, …)
- Test invalid input (string as input, …)
- Test weird cases (network and disk errors, …)
- …
- But be pragmatic

Limits to unit testing

- Code coverage
- Hard to test nondeterministic and threaded code.
- Hard to test frontend and graphical output.
- Takes time to generate realistic test data, errors.
- Only tests what the tests are designed to test.

Where from here?

- Integration and end to end testing
- Acceptance testing

Exercises

- Understand the code from the lecture.
- Look at the find_level() tests in t_find_level.py
    - Add tests as necessary (remember negative cases).
    - Compare your code to the PTMcons repository.
    - If your tests are missing from there, commit them!
- Or, work on adding tests to your own code.

Resources

- This presentation and example code:
    - http://github.com/bitmask/workshop-on-testing
- Python unittest documentation:
    - http://docs.python.org/2/library/unittest.html
- Doxygen documentation:
    - http://www.stack.nl/~dimitri/doxygen/