Assignment #7

Sanja Miklin

11/26/2018

1. Unit Testing in Python

Problem 1

The problem I the function seems to stem from python's range(), which generates values up to, but not including, the stop value. This means that the function doesn't test all possibilities for the smallest factor of n. This is a problem when a number is a square or, more generally, when the smallest factor of a number is equal to its rounded-down square root.

This is the test that I ended up writing:

```
import factor

def test_factor():
    assert factor.smallest_factor(1) == 1, "failed on n=1"
    assert factor.smallest_factor(2) == 2, "failed on n=2"
    assert factor.smallest_factor(7) == 7, "failed on prime numbers"
    assert factor.smallest_factor(9) == 3, "failed on number that's a square"
    assert factor.smallest_factor(8) == 2, "failed on number with the smallest
factor which is equal to its rounded-down square root"
```

Running the test, I get the following output:

```
sanja-miklins-macbook41:Problem_1 suikai$ py.test
============================= test session starts ==============================
platform darwin -- Python 3.7.1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: /Users/suikai/Desktop/Programming/GitHub/persp-
analysis_A18/Assignments/A7/Problem_1, inifile:
collected 1 item

test_factor.py F                                                         [100%]

=================================== FAILURES ===================================
_____________________________________ test_factor ______________________________________

    def test_factor():
        assert factor.smallest_factor(1) == 1, "failed on n=1"
        assert factor.smallest_factor(2) == 2, "failed on n=2"
        assert factor.smallest_factor(7) == 7, "failed on prime numbers"
>       assert factor.smallest_factor(9) == 3, "failed on number that's a square"
E       AssertionError: failed on number that's a square
E       assert 9 == 3
E        +  where 9 = <function smallest_factor at 0x1072c5ea0>(9)
E        +    where <function smallest_factor at 0x1072c5ea0> =
factor.smallest_factor

test_factor.py:7: AssertionError
========================== 1 failed in 0.04 seconds ===========================
```

I then modified the function as follows:

```
def smallest_factor(n):
    """Return the smallest prime factor of the positive integer n."""
    if n == 1: return 1
        for i in range(2, int(n**.5)+1):
            if n % i == 0: return i
        return n
```

After this, as expected, the py.test returns the following:

```
sanja-miklins-macbook41:Problem_1.2 suikai$ py.test
============================= test session starts ==============================
platform darwin -- Python 3.7.1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: /Users/suikai/Desktop/Programming/GitHub/persp-
analysis_A18/Assignments/A7/Problem_1.2, inifile:
collected 1 item

test_factor2.py .                                                        [100%]

========================== 1 passed in 0.05 seconds ===========================
```

Problem 2

First, I check the coverage of the test from Problem 1. As expected, the coverage is 100%.

```
sanja-miklins-macbook41:Problem_1.2 suikai$ py.test --cov
============================= test session starts ==============================
platform darwin -- Python 3.7.1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: /Users/suikai/Desktop/Programming/GitHub/persp-
analysis_A18/Assignments/A7/Problem_1.2, inifile:
plugins: cov-2.6.0
collected 1 item

test_factor2.py .                                                        [100%]

---------- coverage: platform darwin, python 3.7.1-final-0 -----------
Name              Stmts   Miss  Cover
-----------------------------------
factor2.py            5      0   100%
test_factor2.py       7      0   100%
-----------------------------------
TOTAL                12      0   100%

========================== 1 passed in 0.03 seconds ===========================
```

I wrote a test module for the months_length() function as follows:

```
import months

def test_months():
    assert months.month_length("September", leap_year=False) == 30, "Failed on a 30
day month"
```

```
    assert months.month_length("January", leap_year=False) == 31, "Failed on a 31
day month"
    assert months.month_length("February", leap_year=False) == 28, "Failed on a
non-leap February"
    assert months.month_length("February", leap_year=True) == 29, "Failed on a leap
February"
```

And here is the py.test output. As everything is OK with the function, it passes the test.

```
sanja-miklins-macbook41:Problem_2 suikai$ py.test
============================= test session starts ==============================
platform darwin -- Python 3.7.1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: /Users/suikai/Desktop/Programming/GitHub/persp-
analysis_A18/Assignments/A7/Problem_2, inifile:
plugins: cov-2.6.0
collected 1 item

test_months.py .                                                         [100%]

=========================== 1 passed in 0.04 seconds ===========================
```

Here is the coverage-test output. As I included both the leap year and the regular year condition for February in my test module, the coverage is 100%

```
sanja-miklins-macbook41:Problem_2 suikai$ py.test --cov
============================= test session starts ==============================
platform darwin -- Python 3.7.1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: /Users/suikai/Desktop/Programming/GitHub/persp-
analysis_A18/Assignments/A7/Problem_2, inifile:
plugins: cov-2.6.0
collected 1 item

test_months.py .                                                         [100%]

---------- coverage: platform darwin, python 3.7.1-final-0 -----------
Name             Stmts   Miss  Cover
------------------------------------
months.py           10      1    90%
test_months.py       6      0   100%
------------------------------------
TOTAL               16      1    94%

=========================== 1 passed in 0.03 seconds ===========================
```

Problem 3

I wrote the test module for the operate() function as follows:

```
import pytest
import operation

def test_operation():
    assert operation.operate(4,2,'+') == 6, "addition"
    assert operation.operate(4,2,'-') == 2, "subtraction"
```

```
    assert operation.operate(4,2,'*') == 8, "multiplication"
    assert operation.operate(4,2,'/') == 2, "integer division"
    assert operation.operate(5,2,'/') == 2.5, "float division"
    with pytest.raises(ZeroDivisionError) as excinfo:
        operation.operate(a=4, b=0, oper = '/')
    assert excinfo.value.args[0] == "division by zero is undefined"
    with pytest.raises(ValueError) as excinfo:
        operation.operate(a=4, b=0, oper = 'n')
    assert excinfo.value.args[0] == "oper must be one of '+', '/', '-', or '*'"
```

Using this module, I run py.test –cov, and py.test --cov -report with the following outputs. As expected, the function passes, and the coverage is 100%.

```
sanja-miklins-macbook41:Problem_3 suikai$ py.test --cov
============================= test session starts ==============================
platform darwin -- Python 3.7.1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: /Users/suikai/Desktop/Programming/GitHub/persp-
analysis_A18/Assignments/A7/Problem_3, inifile:
plugins: cov-2.6.0
collected 1 item

test_operation.py .                                                      [100%]

---------- coverage: platform darwin, python 3.7.1-final-0 -----------
Name                Stmts   Miss  Cover
---------------------------------------
operation.py           14      1    93%
test_operation.py      14      0   100%
---------------------------------------
TOTAL                  28      1    96%

========================== 1 passed in 0.03 seconds ===========================
```

Coverage report: 96%

Module ↓	*statements*	*missing*	*excluded*	*coverage*
operation.py	14	1	0	93%
test_operation.py	14	0	0	100%
Total	**28**	**1**	**0**	**96%**

coverage.py v4.5.2, created at 2018-11-25 20:08

2. Test driven development

I don't use Python so I wrote the simples function I could think of, which is the following:

```
import numpy

def get_r(K, L, alpha, Z, delta):
    r = alpha*Z*(L/K)**(1-alpha)-delta
    return r
```

I hoped that Python would adapt whether the input is scalar or vector. The function passed the test, so I guess it worked!

```
sanja-miklins-macbook41:Question_2 suikai$ py.test --cov
============================= test session starts ==============================
platform darwin -- Python 3.7.1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: /Users/suikai/Desktop/Programming/GitHub/persp-
analysis_A18/Assignments/A7/Question_2, inifile:
plugins: cov-2.6.0
collected 244 items

test_r.py ..............................................................  [ 25%]
..........................................................................  [ 54%]
..........................................................................  [ 84%]
.......................................                                     [100%]

---------- coverage: platform darwin, python 3.7.1-final-0 -----------
Name        Stmts   Miss  Cover
-------------------------------
get_r.py        4      0   100%
test_r.py      29      0   100%
-------------------------------
TOTAL          33      0   100%


========================== 244 passed in 0.79 seconds ==========================
```

3. Watts (2014)

In his paper, Watts (2014) argues that sociological theories rely on common sense far more than is acknowledged—whether within the field or outside of it. As a result, he claims, sociological explanations often lack scientific validity and, although empathetically satisfying, cannot provide reliable or useful predictions.

Watts illustrates his argument with the case of 'rational choice theory', introduced from economics into sociology and political science in the 1960s. The theory has been heavily critiqued with regards to 1) the assumptions it makes about the actors in questions, which were frequently seen as invalid or implausible (e.g. focus on utility-maximization, perfect reasoning and understanding of future states etc.), and 2) its predictions which often do not align with actual outcomes. In response to these critiques, the theory—or at least the way it is mobilized by researchers—has expanded over time to encompass different kinds of actors and circumstances. However, in this, and that is where Watts's critique lies, the advocates of the theory have moved away from scientific rigor—including an "emphasis on prediction and deduction"—and have shifted towards "empathetic explanations" and "an emphasis on understandability and sense-making."

Ultimately, Watts sees the pitfall of using commonsense theories of action is that their "everyday validity lends them an aura of universal validity," while they might in fact be false, or at least less valid than they appear to be. He argues that scientist studying people will almost inevitably fall into the trap

of building these theories not only because they can "mentally simulate" the perspectives of their subjects (unlike physicists who do not imagine what it is like to be an electron), but because they continuously do it, as humans themselves, in their everyday life. This 'mental simulation,' however, does not lead to what Watts believes are the goals of social science—unveiling of generalizable causal mechanisms and formulation of accurate predictions.

According to Watts, in order to address this epistemological problem that he presents, sociologists have the "confront the difference between empathetic and causal explanation and thereby to produce more scientifically rigorous if not more satisfying explanations." The solution he proposes, somewhat predictably, is to rely more on experimental method, whether using field, natural or laboratory experiments, or to take advantage of 'large N' observational studies and apply to them the counterfactual model of causal inference. These approaches, he argues, would not only be able to produce predictive accuracy, but would also be able to establish causality, therefore contributing to our understanding of underlying mechanisms.

Watts's solution, then, heavily relies on methods and does little to productively engage with theory. While he does acknowledge that rationalizations about human behavior can be useful as sources of "plausible hypotheses," he also makes a comment about how "in social science, this rule [of specifying a hypothesis in advance of the test] is rarely enforced and almost certainly is routinely violated in practice." However, in this, he seems to ignore the fact that theory is ultimately what drives experimental questions and design—sociologist decide what to look at based on what think might matter. Furthermore, it is the theoretical frameworks that actually enable us to quantify many variables and outcomes in ways that make Watts's methods of choice possible. Finally, theory is what bridges findings from different analyses and models to make human action as a whole more comprehensible, and can be leveraged for predictions in completely new contexts (in which the actors, humans, are arguably some sort of a constant. Therefore, as long as sociologist are clear about the assumptions and simplifications they are making, theoretical models are valuable—if not essential—to causal inference and prediction.