Assignment 7 Q1 & Q2-Ying Sun

November 26, 2018

0.0.1 1. Unit Testing in Python

Problem 1

In [1]:
```
# mistake function -- save as sf.py

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)):
        if n % i == 0: return i
    return n
```

In [ ]:
```
# testing sf.py -- save as test_sf.py

import sf

def test_smallest_factor():
    assert sf.smallest_factor(1) == 1, "failed on 1"
    assert sf.smallest_factor(2) == 2, "failed on 2"
    assert sf.smallest_factor(3) == 3, "failed on 3"
    assert sf.smallest_factor(4) == 2, "failed on 4"
    assert sf.smallest_factor(5) == 5, "failed on 5"
    assert sf.smallest_factor(6) == 2, "failed on 6"
    assert sf.smallest_factor(7) == 7, "failed on 7"
    assert sf.smallest_factor(11) == 11, "failed on 11"
    assert sf.smallest_factor(100) == 2, "failed on 100"
```

In [1]:
```
from IPython.display import Image
Image("/Users/yingsun/persp-analysis_A18/Assignments/A7/Q1.png")
```

Out[1]:

```
YING-SUNdeMacBook-Pro:A7 yingsun$ py.test
============================= test session starts ==============================
platform darwin -- Python 3.6.6, pytest-4.0.0, py-1.7.0, pluggy-0.8.0
rootdir: /Users/yingsun/persp-analysis_A18/Assignments/A7, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_sf.py F                                                             [100%]

=================================== FAILURES ===================================
_____________________________ test_smallest_factor _____________________________

    def test_smallest_factor():
        assert sf.smallest_factor(1) == 1, "failed on 1"
        assert sf.smallest_factor(2) == 2, "failed on 2"
        assert sf.smallest_factor(3) == 3, "failed on 3"
>       assert sf.smallest_factor(4) == 2, "failed on 4"
E       AssertionError: failed on 4
E       assert 4 == 2
E        +  where 4 = <function smallest_factor at 0x110503400>(4)
E        +    where <function smallest_factor at 0x110503400> = sf.smallest_factor

test_sf.py:9: AssertionError
========================== 1 failed in 0.08 seconds ============================
YING-SUNdeMacBook-Pro:A7 yingsun$
```

These tests mainly check some normal cases and corner cases: when n = 1, it checks the situation which 1 is not included in range (2, int(n** o.5)); when n = 4, it checks the situation that the upper bound is equal to lower bound (both = 2). According to the test result, it fails on the fourth test. Because for the range function, the upper bound is not included and not be iterated. So the mistake of smallest_factor function is the upper bound of range function and we can correct it.

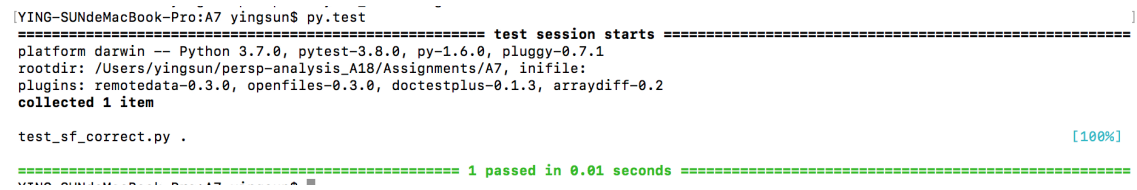
In [ ]:
```
# correct function -- save as sf_correct.py

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
```

In [ ]:
```
# testing sf_correct.py --save as sf_correct.py

import sf_correct

def test_smallest_factor():
    assert sf.smallest_factor(1) == 1, "failed on 1"
    assert sf.smallest_factor(2) == 2, "failed on 2"
    assert sf.smallest_factor(3) == 3, "failed on 3"
    assert sf.smallest_factor(4) == 2, "failed on 4"
    assert sf.smallest_factor(5) == 5, "failed on 5"
    assert sf.smallest_factor(6) == 2, "failed on 6"
    assert sf.smallest_factor(7) == 7, "failed on 7"
```

```
        assert sf.smallest_factor(11) == 11, "failed on 11"
        assert sf.smallest_factor(100) == 2, "failed on 100"
```

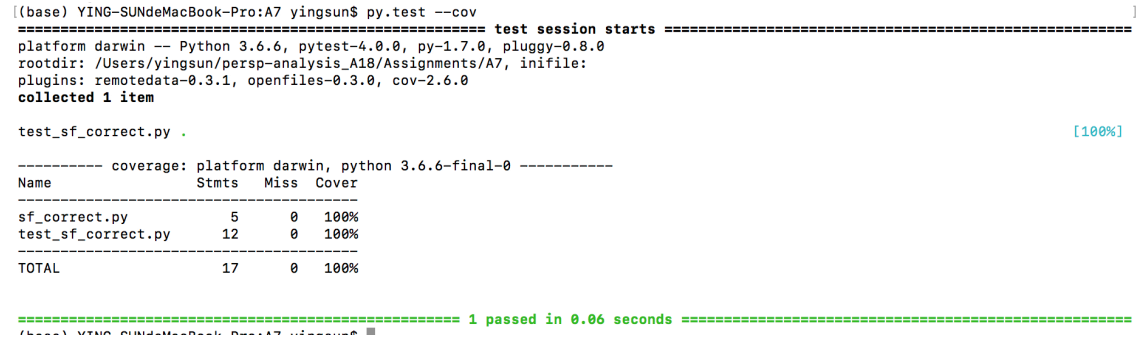
In [6]:
```
Image("/Users/yingsun/persp-analysis_A18/Assignments/A7/Q1_1.png")
```

Out[6]:



In [9]:
```
Image("/Users/yingsun/persp-analysis_A18/Assignments/A7/Q1_2.png")
```

Out[9]:



After fixing the problem, the correct function can pass all the tests with 100% coverage.

Problem 2

In [ ]:
```
# prblem 2 function -- save as ml.py

    def month_length(month, leap_year=False):
        """Return the number of days in the given month."""
        if month in {"September", "April", "June", "November"}:
            return 30
        elif month in {"January", "March", "May", "July",
                               "August", "October", "December"}:
            return 31
        if month == "February":
            if not leap_year:
                return 28
            else:
```

```
                return 29
        else:
            return None
```

In [ ]:
```
# testing ml.py -- save as test_ml.py

import ml

def test_month_length():
    assert ml.month_length("January") == 31, "failed on January"
    assert ml.month_length("February") == 28, "failed on February"
    assert ml.month_length("February", leap_year=True) == 29, \
    "failed on February, leap_year"
    assert ml.month_length("March") == 31, "failed on March"
    assert ml.month_length("April") == 30, "failed on April"
    assert ml.month_length("May") == 31, "failed on May"
    assert ml.month_length("June") == 30, "failed on June"
    assert ml.month_length("July") == 31, "failed on July"
    assert ml.month_length("August") == 31, "failed on August"
    assert ml.month_length("September") == 30, "failed on September"
    assert ml.month_length("October") == 31, "failed on October"
    assert ml.month_length("November") == 30, "failed on November"
    assert ml.month_length("December") == 31, "failed on December"
    assert ml.month_length("Month") == None, "failed on invalid input"
```

These tests above include all the possible situations: the months which have 31 days; the months which have 30 days; the month which has 28 days; the month which has 29 days and invalid input.

In [10]:
```
Image("/Users/yingsun/persp-analysis_A18/Assignments/A7/Q2.png")
```

Out[10]:

According to the result, the month_length function can pass all the tests with 100% coverage.

Problem 3

In [ ]:
```
# operate function --save as opera.py

def operate(a, b, oper):
    """Apply an arithmetic operation to a and b."""
    if type(oper) is not str:
        raise TypeError("oper must be a string")
    elif oper == '+':
        return a + b
    elif oper == '-':
        return a - b
    elif oper == '*':
        return a * b
    elif oper == '/':
        if b == 0:
            raise ZeroDivisionError("division by zero is undefined")
        return a / b
    raise ValueError("oper must be one of '+', '/', '-', or '*'")
```

In [ ]:
```
# testing opera.py -- save as test_opera.py

import opera
import pytest

def test_operate():
    assert opera.operate(6, 8, '+')== 14, "failed on '+'"
    assert opera.operate(6, 8, '-')== -2, "failed on'-'"
    assert opera.operate(6, 8, '*')== 48, "failed on '*'"
    assert opera.operate(6, 8, '/')== 3/4, "failed on '/'"

    with pytest.raises(ZeroDivisionError) as err1:
        opera.operate(6, 0, '/')
    assert err1.value.args[0] == "division by zero is undefined"

    with pytest.raises(TypeError) as err2:
        opera.operate(6, 0, 0)
    assert err2.value.args[0] == "oper must be a string"

    with pytest.raises(ValueError) as err3:
        opera.operate(6, 0, '!=')
    assert err3.value.args[0] == "oper must be one of '+', '/', '-', or '*'"
```

The test includes all the possible operations and typical errors.

In [11]:
```
Image("/Users/yingsun/persp-analysis_A18/Assignments/A7/Q3.png")
```

Out[11]:

```
(base) YING-SUNdeMacBook-Pro:A7 yingsun$ py.test --cov
============================================================ test session starts ============================================================
platform darwin -- Python 3.6.6, pytest-4.0.0, py-1.7.0, pluggy-0.8.0
rootdir: /Users/yingsun/persp-analysis_A18/Assignments/A7, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.0, cov-2.6.0
collected 1 item

test_opera.py .                                                                                                                      [100%]

---------- coverage: platform darwin, python 3.6.6-final-0 -----------
Name            Stmts   Miss  Cover
-------------------------------------
opera.py           14      0   100%
test_opera.py      16      0   100%
-------------------------------------
TOTAL              30      0   100%


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

In [2]: Image("/Users/yingsun/persp-analysis_A18/Assignments/A7/Q3_1.png")

Out[2]:

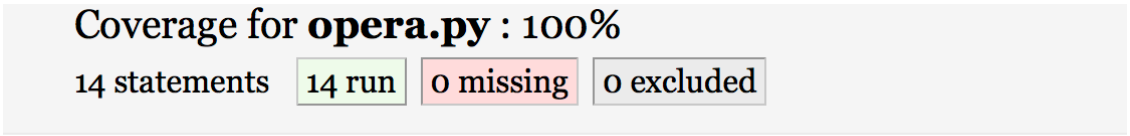
Coverage for **test_opera.py** : 100%

16 statements | 16 run | 0 missing | 0 excluded

```python
import opera
import pytest

def test_operate():
    assert opera.operate(6, 8, '+')== 14, "failed on '+'"
    assert opera.operate(6, 8, '-')== -2, "failed on'-'"
    assert opera.operate(6, 8, '*')== 48, "failed on '*'"
    assert opera.operate(6, 8, '/')== 3/4, "failed on '/'"

    with pytest.raises(ZeroDivisionError) as err1:
        opera.operate(6, 0, '/')
    assert err1.value.args[0] == "division by zero is undefined"

    with pytest.raises(TypeError) as err2:
        opera.operate(6, 0, 0)
    assert err2.value.args[0] == "oper must be a string"

    with pytest.raises(ValueError) as err3:
        opera.operate(6, 0, '!=')
    assert err3.value.args[0] == "oper must be one of '+', '/', '-', or '*'"
```

In [4]: Image("/Users/yingsun/persp-analysis_A18/Assignments/A7/Q3_2.png")

Out[4]:




```
1  def operate(a, b, oper):
2      """Apply an arithmetic operation to a and b."""
3      if type(oper) is not str:
4          raise TypeError("oper must be a string")
5      elif oper == '+':
6          return a + b
7      elif oper == '-':
8          return a - b
9      elif oper == '*':
10         return a * b
11     elif oper == '/':
12         if b == 0:
13             raise ZeroDivisionError("division by zero is undefined")
14         return a / b
15     raise ValueError("oper must be one of '+', '/', '-', or '*'")
```

According to these results, the operate function can pass all the tests with 100% coverage. Besides the results in terminal, we also can use py.test --- cov-report html --cov to generate an HTML file for visualizing the current line coverage. This HTML file is in the folder of htmlcov.

0.0.2 2. Test driven development

(a) and (b)

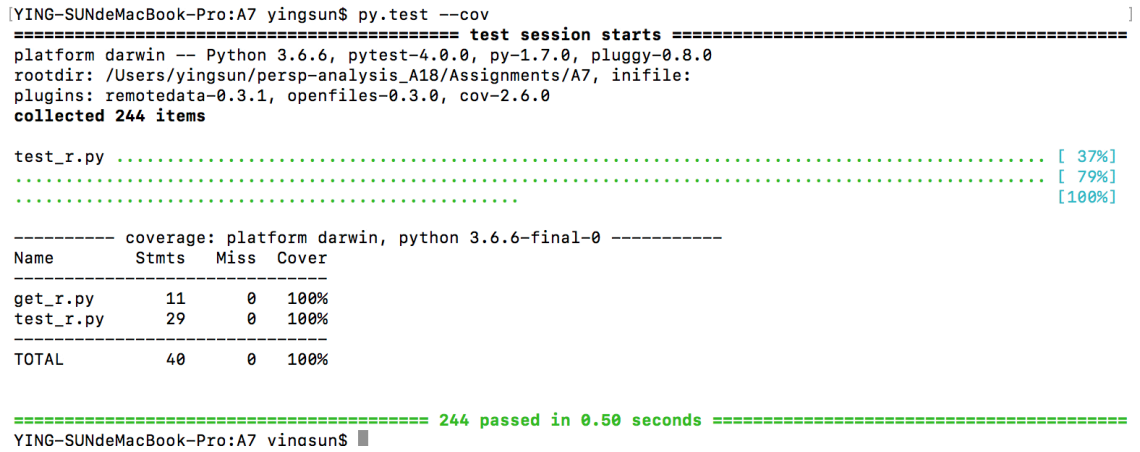
```
In [ ]: # function get_r -- save as get_r.py
        import numpy as np

        def get_r(K, L, alpha, Z, delta):
            '''
            This function generates the interest rate or vector of interest rates
            '''
            assert alpha > 0 and alpha < 1, "alpha should between 0 and 1"
            assert delta >= 0 and delta < 1, "delta should between 0 and 1"
            assert Z > 0, "Z should lager than 0"

            r = alpha * Z * ( (L / K) ** (1 - alpha) ) - delta

            if (type(K) == float) and (type(L)== float):
                assert type(r)==float, "If K and L are both scalars,  \
                the interest rate should be a scalar"

            if not np.isscalar(K) and not np.isscalar(L):
```

```
            assert not np.isscalar(r), "If K and L are both vectors, \
            the interest rate should be a vector"

        return r
```

(c)

```
In [3]: Image("/Users/yingsun/persp-analysis_A18/Assignments/A7/2.png")
```

Out[3]:



According to the result, the get_r function can pass all the tests with 100% coverage.

0.0.3 3. Watts (2014)

See attached pdf