Problem1

November 25, 2018

0.1 Problem 1.

0.1.1 Problem 1.1

For this problem, I suppose that the range part of the code has a little problem. Considering that in python, the range function doesn't include the upper bound of the range, thus, in this smallest_factor function, it is better to use int(n^0.5)+1 rather than int(n^0.5) to make sure we have include all possible circumstances. To verify my suspicion, let's do following test:

```
In [7]: # introduce the function. Save this function as problem1.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 [11]: # Then, write the test for this function.Save this test as test_problem1.py
         import problem1
         def test_problem1():
             assert problem1.smallest_factor(2) == 2, 'failed on 2'
             assert problem1.smallest_factor(3) == 3, 'failed on 3'
             assert problem1.smallest_factor(4) == 2, 'failed on 4'
             assert problem1.smallest_factor(9) == 3, 'failed on 9'
             assert problem1.smallest_factor(13) == 13, 'failed on 13'
```

```
In [15]: ! py.test
```

```
============================= test session starts =============================
platform win32 -- Python 3.7.1rc1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: F:\\\\\Perspective\Assignment 7, inifile:
collected 1 item

test_problem1.py F                                                       [100%]

================================== FAILURES ===================================
_______________________________ test_problem1 ________________________________

    def test_problem1():
        assert problem1.smallest_factor(2) == 2, 'failed on 2'
        assert problem1.smallest_factor(3) == 3, 'failed on 3'
>       assert problem1.smallest_factor(4) == 2, 'failed on 4'
E       AssertionError: failed on 4
E       assert 4 == 2
```

```
E        +  where 4 = <function smallest_factor at 0x00000156668D8400>(4)
E        +    where <function smallest_factor at 0x00000156668D8400> =
problem1.smallest_factor

test_problem1.py:5: AssertionError
========================== 1 failed in 0.05 seconds ==========================
```

From the above test, we could see that when we test n as 4, the upper bound of the range is the same as the lower bound. Then, there is an error with the range function. What's more, there is also be an error for n as 9, where it won't test 3 as the smallest factor. Therefore, this function needs some modification.

```
In [16]: # The correct version of this function.
         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 [19]: ! py.test

============================= test session starts =============================
platform win32 -- Python 3.7.1rc1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: F:\\\\\Perspective\Assignment 7, inifile:
plugins: cov-2.6.0
collected 1 item

test_problem1.py .                                                       [100%]

========================== 1 passed in 0.02 seconds ===========================
```

We could see that it passed all the test in the test_problem1.py now.

0.1.2 Problem 1.2

To start with, we shall test the coverage of my former test.

```
In [20]: ! py.test --cov

============================= test session starts =============================
platform win32 -- Python 3.7.1rc1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: F:\\\\\Perspective\Assignment 7, inifile:
plugins: cov-2.6.0
collected 1 item

test_problem1.py .                                                       [100%]

--------- coverage: platform win32, python 3.7.1-candidate-1 ---------
Name               Stmts   Miss  Cover
```

```
---------------------------------------
problem1.py              5       0   100%
test_problem1.py         7       0   100%
---------------------------------------
TOTAL                   12       0   100%


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

Considering that it covers all codes of the smallest_factor function, it doesn't need supplement. We shall move to the test of the month_length function.

```
In [21]: # introduce the function. Save this function as problem2.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 [22]: # Then, write the test for this function.Save this test as test_problem2.py
         import problem2
         def test_problem2():
             assert problem2.month_length("February", leap_year = False) == 28, 'failed on
         February in common year'
             assert problem2.month_length("February", leap_year = True) == 29, 'failed on
         February in leap year'
             assert problem2.month_length("March", leap_year = False) == 31, 'failed on March in
         common year'
             assert problem2.month_length("March", leap_year = True) == 31, 'failed on March in
         leap year'
             assert problem2.month_length("April", leap_year = False) == 30, 'failed on April in
         common year'
             assert problem2.month_length("April", leap_year = True) == 30, 'failed on April in
         leap year'

In [23]: !py.test --cov
```

```
============================= test session starts =============================
platform win32 -- Python 3.7.1rc1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: F:\\\\\Perspective\Assignment 7, inifile:
plugins: cov-2.6.0
collected 2 items

test_problem1.py .                                                       [ 50%]
test_problem2.py .                                                       [100%]

--------- coverage: platform win32, python 3.7.1-candidate-1 ---------
Name               Stmts   Miss  Cover
---------------------------------------
```

```
problem1.py             5      0   100%
problem2.py            10      1    90%
test_problem1.py        7      0   100%
test_problem2.py        8      0   100%
-----------------------------------------
TOTAL                  30      1    97%


========================== 2 passed in 0.07 seconds ==========================
```

From the result, we could see that this function is correct.

0.1.3 Problem 1.3

```
In [24]: # introduce the function. Save this function as problem3.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 [25]: #Then, write the test for this function.Save this test as test_problem2.py
         import pytest
         import problem3
         def test_problem3():
             assert problem3.operate(2,7,"+") == 9, 'failed on add function'
             assert problem3.operate(2,7,"-") == -5, 'failed on subtract function'
             assert problem3.operate(2,7,"*") == 14, 'failed on multiply function'
             assert problem3.operate(4,2,"/") == 2, 'failed on division function'
             pytest.raises(TypeError, problem3.operate, a=2, b=7, oper=2.7)
             pytest.raises(ZeroDivisionError, problem3.operate, a=2, b=0, oper='/')
             pytest.raises(ValueError, problem3.operate, a=2, b=7, oper="^")

In [27]: !py.test --cov
```

```
============================= test session starts =============================
platform win32 -- Python 3.7.1rc1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: F:\\\\\Perspective\Assignment 7, inifile:
plugins: cov-2.6.0
collected 3 items

test_problem1.py .                                                       [ 33%]
test_problem2.py .                                                       [ 66%]
test_problem3.py .                                                       [100%]

--------- coverage: platform win32, python 3.7.1-candidate-1 ---------
```

```
Name                Stmts   Miss  Cover
-------------------------------------
problem1.py             5      0   100%
problem2.py            10      1    90%
problem3.py            14      0   100%
test_problem1.py        7      0   100%
test_problem2.py        8      0   100%
test_problem3.py       10      0   100%
-------------------------------------
TOTAL                  54      1    98%


========================== 3 passed in 0.08 seconds ==========================
```

```
In [28]: !py.test --cov-report html --cov

============================= test session starts =============================
platform win32 -- Python 3.7.1rc1, pytest-4.0.1, py-1.7.0, pluggy-0.8.0
rootdir: F:\\\\\Perspective\Assignment 7, inifile:
plugins: cov-2.6.0
collected 3 items

test_problem1.py .                                                       [ 33%]
test_problem2.py .                                                       [ 66%]
test_problem3.py .                                                       [100%]

--------- coverage: platform win32, python 3.7.1-candidate-1 ---------
Coverage HTML written to dir htmlcov


========================== 3 passed in 0.11 seconds ==========================
```

From the result, we could see that this function is correct and the test covers all codes.