Ch03-4-UnittestingFunctions

September 10, 2025

1 Unit testing Code

1.1 Topics

- manual testing
- autmatic testing
- unit testing
- `mypy` test runner and reporting

1.2 Manual testing

- testing the whole programming manually by executing the program
- there are many types of manual testing
- important but time consuming because of the need to manually document the errors, problems
- difficult to reproduce the errors

1.3 Unit testing

- Unit testing is a software testing technique where individual units or components of a software application are tested in isolation to ensure that they function correctly
- a unit is the smallest testable part of an application, such as a function, method, or class
- the primary goal of unit testing is to validate that each unit of the software performs as expected

1.3.1 Some key aspects of unit testing

Isolation

- Each unit test focuses on a single piece of functionality, testing it independently of other parts of the system

Automation

- Unit tests are typically automated, allowing them to be run frequently and consistently, often as part of a continuous integration (CI) process

Repeatability

- Unit tests should produce the same results every time they are run, regardless of the environment or external factors

Fast Execution

- Unit tests are usually designed to be fast, enabling rapid feedback for developers

Code Coverage

- Good unit testing aims to cover as much of the code as possible, ensuring that different paths, edge cases, and scenarios are tested

1.4 Unit testing fruitful functions

- functions can be testing automatically as well as manually
- assert statement can be used to automatically test **fruitful** functions
- each assertion must be True or must pass in order to continue to the next
- if assertion fails, throws AssertionError exception and program halts

[38]:
```
# Examples of assert statments
# == comparison operator that lets you compare two values
# More on comparison operators in later chapter
assert True == True
```

[39]:
```
assert 10 != '10'
```

[40]:
```
assert True == False
print('this will not be printed')
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[40], line 1
----> 1 assert True == False
      2 print('this will not be printed')

AssertionError:
```

[41]:
```
assert 'a' == 'A'
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[41], line 1
----> 1 assert 'a' == 'A'

AssertionError:
```

[42]:
```
# Auto testing or asserting add function
assert add(2, 3) == 5
assert add(10, -5) == 5
# assert add(100, 2000.99) == ?
```

[43]:
```
# Unit test multiply function
# Write some sample test cases for multiply function using assert statement
```

1.5 pytest

- https://docs.pytest.org/en/stable/
- Pytest is a popular testing framework for Python
- allows developers to write simple and scalable test cases
- helps you find and run all the test cases providing a complete report of the test results
- pytest is a third party library/framework that you must install using pip (aka pip installs package) from a Terminal/Command Line
- install and check the version of pytest

```
$ pip install -U pytest
$ pytest --version
```

[4]:
```
! pip install -U pytest
```

```
Requirement already satisfied: pytest in
/opt/anaconda3/envs/py/lib/python3.10/site-packages (8.3.2)
Requirement already satisfied: iniconfig in
/opt/anaconda3/envs/py/lib/python3.10/site-packages (from pytest) (1.1.1)
Requirement already satisfied: packaging in
/opt/anaconda3/envs/py/lib/python3.10/site-packages (from pytest) (22.0)
Requirement already satisfied: pluggy<2,>=1.5 in
/opt/anaconda3/envs/py/lib/python3.10/site-packages (from pytest) (1.5.0)
Requirement already satisfied: exceptiongroup>=1.0.0rc8 in
/opt/anaconda3/envs/py/lib/python3.10/site-packages (from pytest) (1.1.3)
Requirement already satisfied: tomli>=1 in
/opt/anaconda3/envs/py/lib/python3.10/site-packages (from pytest) (2.0.1)
```

[5]:
```
! pytest --version
```

```
pytest 8.3.2
```

1.5.1 Run Pytest

- run the following commands from a Terminal/Command Prompt

```
$ cd <project folder>
$ pytest -v test_python_file.py
```

[6]:
```
%pwd
```

[6]:
```
'/Users/rbasnet/projects/Python-Fundamentals'
```

[7]:
```
%cd demos/function_unittest/
```

```
/Users/rbasnet/projects/Python-Fundamentals/demos/function_unittest
```

[10]:
```
! pytest -v add.py
```

```
============================= test session starts
==============================
platform darwin -- Python 3.10.8, pytest-8.3.2, pluggy-1.5.0 --
/opt/anaconda3/envs/py/bin/python
cachedir: .pytest_cache
hypothesis profile 'default' ->
database=DirectoryBasedExampleDatabase('/Users/rbasnet/projects/Python-
Fundamentals/demos/function_unittest/.hypothesis/examples')
rootdir: /Users/rbasnet/projects/Python-Fundamentals/demos/function_unittest
plugins: anyio-4.0.0, cov-4.1.0, hypothesis-6.62.1
collected 3 items


add.py::test_add PASSED
[ 33%]
add.py::test_add2 PASSED
[ 66%]
add.py::test_add3 PASSED
[100%]

============================== 3 passed in 1.25s
==============================
```

1.6 Exercises

1.6.1 exercise 1

Write a function that takes two numbers; subtracts the second from the first and returns the difference. Write two test cases.

[51]:
```
# Solution to exercise 1
def sub(num1, num2):
    return num1 - num2
```

[52]:
```
def test_sub():
    assert sub(100, 50) == 50
    assert sub(80, 45.5) == 34.5
    print('all test cases passed for sub()')
```

[53]:
```
test_sub()
```

```
all test cases passed for sub()
```

1.6.2 exercise 2

Write a function that converts seconds to hours, minutes and seconds. Function then returns the values in **HH:MM:SS** format (e.g., 01:09:10)

[54]:
```
def get_time(seconds):
    pass
```

[55]:
```
# Here are some tests that should pass:
def test_get_time():
    assert get_time(3600) == '1:0:0'
    assert get_time(3661) == '1:1:1'
    assert get_time(3666) == '1:1:6'
    assert get_time(36610) == '10:10:10'
    print('all test cases passed for get_time()')
```

[56]:
```
test_get_time()
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[56], line 1
----> 1 test_get_time()

Cell In[55], line 3, in test_get_time()
      2 def test_get_time():
----> 3     assert get_time(3600) == '1:0:0'
      4     assert get_time(3661) == '1:1:1'
      5     assert get_time(3666) == '1:1:6'

AssertionError:
```

1.6.3 exercise 3

Write a function called hypotenuse that returns the length of the hypotenuse of a right triangle given the lengths of the two legs as parameters.

[57]:
```
def hypotenuse(leg1, leg2):
    pass
```

[58]:
```
def test_hypotenuse():
    assert hypotenuse(3, 4) == 5.0
    assert hypotenuse(12, 5) == 13.0
    assert hypotenuse(24, 7) == 25.0
    assert hypotenuse(9, 12) == 15.0
    print('all test cases passed hypotenuse()')
```

[59]:
```
test_hypotenuse()
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[59], line 1
----> 1 test_hypotenuse()
```

```
Cell In[58], line 2, in test_hypotenuse()
      1 def test_hypotenuse():
----> 2     assert hypotenuse(3, 4) == 5.0
      3     assert hypotenuse(12, 5) == 13.0
      4     assert hypotenuse(24, 7) == 25.0

AssertionError:
```

1.6.4 exercise 4

Write a function $slope(x1, y1, x2, y2)$ that returns the slope of the line through the points $(x1, y1)$ and $(x2, y2)$. Be sure your implementation of slope can pass the test cases provided in **test_slope( )**.

Then use a call to slope in a new function named intercept(x1, y1, x2, y2) that returns the y-intercept of the line through the points $(x1, y1)$ and $(x2, y2)$

[60]:
```
def slope(x1, y1, x2, y2):
    pass
```

[61]:
```
def test_slope():
    assert slope(5, 3, 4, 2) == 1.0
    assert slope(1, 2, 3, 2) == 0.0
    assert slope(1, 2, 3, 3) == 0.5
    assert slope(2, 4, 1, 2) == 2.0
    print('all test cases passed for slope()')
```

[62]:
```
test_slope()
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[62], line 1
----> 1 test_slope()

Cell In[61], line 2, in test_slope()
      1 def test_slope():
----> 2     assert slope(5, 3, 4, 2) == 1.0
      3     assert slope(1, 2, 3, 2) == 0.0
      4     assert slope(1, 2, 3, 3) == 0.5

AssertionError:
```

[63]:
```
def intercept(x1, y1, x2, y2):
    pass
```

[64]:
```
def test_intercept():
    assert intercept(1, 6, 3, 12) == 3.0
    assert intercept(6, 1, 1, 6) == 7.0
    assert intercept(4, 6, 12, 8) == 5.0
    print('all test cases passed for intercept()')
```

[65]:
```
test_intercept()
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[65], line 1
----> 1 test_intercept()

Cell In[64], line 2, in test_intercept()
      1 def test_intercept():
----> 2     assert intercept(1, 6, 3, 12) == 3.0
      3     assert intercept(6, 1, 1, 6) == 7.0
      4     assert intercept(4, 6, 12, 8) == 5.0

AssertionError:
```

[ ]: