​

Python Testing

In development we use an approach called TDD, Test Driven Development.

Here we create our tests before we write our code.

1. Doc Test:

Something that is unique to Python.

We write a doc string just when a function starts to say what the function does.

The tests are written similar to how we right code in a python shell, using three chevron symbol.

— Running doctest:

python -m dockets test.py

-m tells python to load the doc test module. doc test module runs the doc tests written in the test.py.

If nothing comes back on running the file, then its good. noting was failed.

Disadvantage:

can’t be reused as they are bound to the code they are written on.

all of there comparison is done through string comparison, so it can be tricky to compare floats etc.

2. UnitTest:Testing one particular aspect (method, function, etc) is called unit Testing.

Its a package in Python to create unit tests for our tests.

Unit tests help us to write a new python file for running our tests.

This has an advantage that we can execute it a normal python file unlike DocTest.

The methods in the unitest library should start with ‘test’ else they won’t run.

We normally import the ‘unittest’ module and create a class that extends ‘unittest.TestCase’ class

Sample Code:

import unittest

class SomeTest(unittest.TestCase):

def setUp(self):

# adding some initial set up.

def test\_five\_plus\_five(self): # The name of the method should start with test.

assert 5+5 = 10

def test\_one\_plus\_one(self):

assert not 1+1 =3

if \_\_name\_\_ == ‘\_\_main\_\_’:

unittest.main()

We can run this code by:

python -m unittest file\_name.py

python file\_name.py

——————————

All testing libraries work on the concept of assertions.

An assertion tests a condition in our code that must be met.

Quantitive Assertions:

assertEqual(a, b) - passes if a and b are equal.

assertNotEqual(a, b) - passes if a and b are not equal.

assertGreater(a, b) - passes if a is greater than b

assertGreaterEqual(a,b)

assertLessEqual(a, b)

assertLess(a, b) - passes if a is less than b

assertIn()

assertNotIn()

assertIsInstance(thisThing, isAnInstnaceofThisClass)

Testing that our exceptions get raised:

assertRaises Assertion is used in this case.

How to use assertRaises:

its used a bit differently when compared to other assertions:

with self.assertRaises(ValueError): # the type of exception that we need to confirm will be raised.

someMethod()

This test will pass only if the exception that we want is raised!.

Similar to ‘assertRaises’ we have ‘assertWarns’ and ‘assertLogs’ assertions. We can use these to check if warning or Log entries are created by our code.

Using Coverage:

This is library in Python that is used to tell if we have covered every aspect of our code in a test.

Install it using : pip install coverage

‘coverage run file\_name.py’ will simply run a tests

‘coverage report’ will give an overall report of % of areas that we have covered.

‘coverage report -m’ gives us the line of codes that we missed to include in our test.

‘coverage html’ will give a simple html version of our coverage.