Unit testing

A method by which individual units of source code.  *In computer programming, unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine if they are fit for use.*

unittest module

In Python we have unittest module to help us.

Factorial code

In this example we will write a file *factorial.py*.

**import** **sys**

**def** fact(n):

*"""*

*Factorial function*

*:arg n: Number*

*:returns: factorial of n*

*"""*

**if** n == 0:

**return** 1

**return** n \* fact(n -1)

**def** div(n):

*"""*

*Just divide*

*"""*

res = 10 / n

**return** res

**def** main(n):

res = fact(n)

print(res)

**if** \_\_name\_\_ == '\_\_main\_\_':

**if** len(sys.argv) > 1:

main(int(sys.argv[1]))

Output

$ python factorial.py 5

Which function to test ?

As you can see fact(n) is function which is doing all calculations, so we should test that at least.

Our first test case

Now we will write our first test case.

**import** **unittest**

**from** **factorial** **import** fact

**class** **TestFactorial**(unittest.TestCase):

*"""*

*Our basic test class*

*"""*

**def** test\_fact(self):

*"""*

*The actual test.*

*Any method which starts with ``test\_`` will considered as a test case.*

*"""*

res = fact(5)

self.assertEqual(res, 120)

**if** \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

Running the test:

$ python factorial\_test.py

.

----------------------------------------------------------------------

Ran 1 test in 0.000s

OK

Description

We are importing unittest module first and then the required functions which we want to test.

A testcase is created by subclassing unittest.TestCase.

Now open the test file and change *120* to *121* and see what happens :)

Different assert statements

| **Method** | **Checks that** | **New in** |
| --- | --- | --- |
| *assertEqual(a, b)* | a == b |  |
| *assertNotEqual(a, b)* | a != b |  |
| *assertTrue(x)* | bool(x) is True |  |
| *assertFalse(x)* | bool(x) is False |  |
| *assertIs(a, b)* | a is b | 2.7 |
| *assertIsNot(a, b)* | a is not b | 2.7 |
| *assertIsNone(x)* | x is None | 2.7 |
| *assertIsNotNone(x)* | x is not None | 2.7 |
| *assertIn(a, b)* | a in b | 2.7 |
| *assertNotIn(a, b)* | a not in b | 2.7 |
| *assertIsInstance(a, b)* | isinstance(a, b) | 2.7 |
| *assertNotIsInstance(a, b)* | not isinstance(a, b) | 2.7 |

Testing exceptions

If we call div(0) in factorial.py , we can see if raises an exception.

We can also test these exceptions, like:

self.assertRaises(ZeroDivisionError, div, 0)

Full code

**import** **unittest**

**from** **factorial** **import** fact, div

**class** **TestFactorial**(unittest.TestCase):

*"""*

*Our basic test class*

*"""*

**def** test\_fact(self):

*"""*

*The actual test.*

*Any method which starts with ``test\_`` will considered as a test case.*

*"""*

res = fact(5)

self.assertEqual(res, 120)

**def** test\_error(self):

*"""*

*To test exception raise due to run time error*

*"""*

self.assertRaises(ZeroDivisionError, div, 0)

**if** \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**import** unittest

**def** fun**(**x**):**

**return** x + 1

**class** MyTest**(**unittest.TestCase**):**

**def** test**(**self**):**

self.assertEqual**(**fun**(**3**),** 4**)**