LABORATORIO 1

Primero escribiremos algunas funciones y luego nos centraremos en escribir las pruebas. Primero, abra una carpeta en su favorito Editor de código. Y crea un archivo llamado `utils.py`. Pegue el siguiente código en el archivo.

```
import math


def is_prime(n):
    if n < 0:
        return 'Negative numbers are not allowed'

    if n <= 1:
        return False

    if n == 2:
        return True

    if n % 2 == 0:
        return False

    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True


def cubic(a):
    return a * a * a


def say_hello(name):
    return "Hello, " + name
```

Copy

Tenemos tres funciones diferentes en el `utils.py` expediente. Ahora, tenemos que probar cada función con diferentes casos de prueba. Escribamos las pruebas para la primera función. `is_prime`.

#1. Crea un archivo llamado `test_utils.py` en la carpeta de muestra como `utils.py`.

#2. Importar el `utils` y `unittest` módulo.

#3. Crea una clase con el nombre `TestUtils` extensión `unittest.TestCase` clase. El nombre de la clase puede ser cualquier cosa. Intente darle a la clase un nombre significativo.

#4. Dentro de la clase, escribe un método llamado `test_is_prime` que acepta `self` como argumento

#5. Escriba diferentes casos de prueba con argumentos al `is_prime` y compare la salida con la salida esperada.

#6. Caso de prueba de ejemplo `self.assertFalse(utils.is_prime(1))`.

#7. Esperamos el resultado del `is_prime(1)` será falso en el caso anterior.

#8. Al igual que en el caso anterior, realizaremos diferentes casos de prueba en función de la función que estamos probando.

Veamos las pruebas.

```
import unittest

import utils


class TestUtils(unittest.TestCase):
    def test_is_prime(self):
        self.assertFalse(utils.is_prime(4))
        self.assertTrue(utils.is_prime(2))
        self.assertTrue(utils.is_prime(3))
        self.assertFalse(utils.is_prime(8))
        self.assertFalse(utils.is_prime(10))
        self.assertTrue(utils.is_prime(7))
        self.assertEqual(utils.is_prime(-3),
                         "Negative numbers are not allowed")


if __name__ == '__main__':
    unittest.main()
```

Estamos invocando el `main` método de `unittest` el módulo para ejecutar las pruebas usando `python filename.py` El comando. Ahora, ejecute las pruebas.

Verá una salida similar a la siguiente.

```
$ python test_utils.py
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Ahora, intente escribir también los casos de prueba para otras funciones. Piense en diferentes casos para las funciones y escriba pruebas para ellos. Eche un vistazo a las siguientes pruebas que se agregan a la clase anterior.

```
...


class TestUtils(unittest.TestCase):
    def test_is_prime(self):
```

```
    ...

  def test_cubic(self):
    self.assertEqual(utils.cubic(2), 8)
    self.assertEqual(utils.cubic(-2), -8)
    self.assertNotEqual(utils.cubic(2), 4)
    self.assertNotEqual(utils.cubic(-3), 27)

  def test_say_hello(self):
    self.assertEqual(utils.say_hello("Geekflare"), "Hello, Geekflare")
    self.assertEqual(utils.say_hello("Chandan"), "Hello, Chandan")
    self.assertNotEqual(utils.say_hello("Chandan"), "Hi, Chandan")
    self.assertNotEqual(utils.say_hello("Hafeez"), "Hi, Hafeez")


...
```

Hemos aprendido a escribir pruebas unitarias utilizando el módulo unittest. Ahora es el momento de ver diferentes formas de ejecutar las pruebas.

Ya hemos visto una forma de ejecutar los casos de prueba en la sección anterior. Veamos las otras dos formas de ejecutar las pruebas usando el módulo unittest.

#1. Usando el nombre del archivo y el módulo unittest.

En este método, usaremos el módulo unittest y el nombre del archivo para ejecutar las pruebas. El comando para ejecutar las pruebas es `python -m unittest filename.py`. En nuestro caso, el comando para ejecutar las pruebas es `python -m unittest test_utils.py`.

#2. Usando el método de descubrimiento

Usaremos el `discover` método del módulo unittest para detectar automáticamente todos los archivos de prueba y ejecutarlos. Para detectar automáticamente los archivos de prueba, debemos nombrarlos comenzando con la palabra clave `test`.

El comando para ejecutar las pruebas usando el método de descubrimiento es `python -m unittest discover`. El comando detectará todos los archivos cuyos nombres comiencen con `test` y ejecutarlos.