Introducción a unittest

Para probar los programas existen las pruebas manuales y las pruebas automáticas. La prueba manual es una técnica de prueba en la que los humanos realizan la prueba manualmente una vez finalizado el desarrollo. La prueba automática es una técnica de prueba en la que se crean programas que realizan pruebas de manera automática y entregan los resultados.

Las pruebas manuales llevan mucho tiempo y son difíciles de realizar. Los desarrolladores escriben código para realizar las pruebas (pruebas automáticas). Existen diferentes tipos de pruebas en las pruebas automáticas. Algunas de ellas son Pruebas unitarias, Pruebas de integración, Pruebas de extremo a extremo, Pruebas de estrés, etc,

El flujo estándar de las pruebas.

- Escribir o Actualizar el código.
- Escribir o Actualizar las pruebas para los diferentes casos de su código.
- Ejecutar las pruebas (manualmente o utilizando un ejecutor de pruebas).
- Ver los resultados de las pruebas. Si hay algún error, corregirlo y repetir los pasos.

A continuación se analizará el tipo de pruebas más esencial y básico llamado pruebas unitarias.

¿Qué son las pruebas unitarias?

Las pruebas unitarias son una técnica para probar un bloque de código pequeño e independiente. El bloque de código pequeño generalmente es una función. La palabra independiente significa que no depende de otras piezas de código del proyecto.

Suponga que se debe comprobar si una cadena es igual a "**`Programador`**" o no. Para ello, se ha escrito una función que toma un argumento y devuelve si es igual a "**`Programador`**" o no.

```
def es_igual_a_programador(cadena):
     return cadena == "Programador"
```

La función anterior no depende de ningún otro código. Por lo tanto, se puede probar de forma independiente entregándole diferentes entradas. La pieza de código independiente se puede utilizar en todo el proyecto.

Importancia de las pruebas unitarias

En general, el código de bloques independientes puede utilizarse en todo el proyecto. Por lo tanto, debe estar bien escrito y probado. Las pruebas unitarias son las que se utilizan para probar este tipo de bloques independientes de código.

¿Qué ocurre si no utilizamos pruebas unitarias en nuestro proyecto?

Suponga que no se prueban los bloques de código pequeños que se utilizan en todo el proyecto. Todas las demás pruebas, como las de integración, las de extremo a extremo, etc., que utilizan los bloques pequeños de código pueden fallar. Esto rompe la aplicación.

¿Qué es unittest en Python?

Python **`unittest`** es un marco (framework) de pruebas integrado para probar código Python.

Tiene un ejecutor de pruebas, que nos permite ejecutar las pruebas. Por lo tanto, se puede utilizar el módulo incorporado **`unittest`** para las pruebas sin utilizar módulos de terceros.

Tenemos que seguir los siguientes pasos para probar el código Python utilizando el módulo unittest.

1. Escriba el código.
2. Importe el módulo **`unittest`**.
3. Cree un archivo que comience con la palabra clave **`test`**. Por ejemplo **`test_primo.py`**. La palabra clave **`test`** se utiliza para identificar los archivos de prueba.
4. Cree una clase que extienda la clase **`unittest.TestCase`**.
5. Escriba métodos (pruebas) dentro de la clase. Cada método contiene diferentes casos de prueba basados en su requerimiento. Se debe nombrar el método comenzando con la palabra clave **`test`**.
6. Ejecute las pruebas. Se puede ejecutar las pruebas de diferentes maneras.

Ejecución usando el comando **`python -m unittest nombre_del_archivo.py`**.

Ejecutando los archivos de prueba como archivos generales de Python con el comando **`python nombre_del_archivo.py`**.

Para que este método funcione, se necesita invocar el método principal del **`unittest`** en el archivo de prueba.

Y por último, utilizando el **`discover`**. Se pueden autoejecutar las pruebas utilizando el comando **`python -m unittest discover`** sin mencionar el nombre de archivo de la prueba. Se encuentran las pruebas utilizando la convención de nomenclatura que se ha seguido. Por lo tanto, se debe nombrar a los archivos de prueba con la palabra clave **`test`** en el inicio.

Generalmente, en las pruebas, se compara la salida del código con la salida esperada. Así que, para comparar las salidas **`unittest`** proporciona diferentes métodos. La lista de métodos en **`unittest`** es:

Method	Checks that
`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`
`assertIsNot(a, b)`	`a is not b`
`assertIsNone(x)`	`x is None`
`assertIsNotNone(x)`	`x is not None`
`assertIn(a, b)`	`a in b`
`assertNotIn(a, b)`	`a not in b`
`assertIsInstance(a, b)`	`isinstance(a, b)`
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`

Para obtener mayor información sobre el módulo **`unittest`**, puede revisar la documentación oficial y aclarar sus dudas.

(*https://docs.python.org/es/3.9/library/unittest.html*).

Pruebas unitarias en Python utilizando unittest

Primero se escriben las funciones que se desean probar y a continuación se escribin las pruebas. En primer lugar, abra una carpeta en su editor de código favorito. Y cree un archivo llamado **`utils.py`**.

```
import math

def es_primo(n):
    if n < 0:
        return 'Los números negativos no están permitidos'

    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 cubico(a):
    return a * a * a


def decir_hola(nombre):
    return "Hola, " nombre
```

Se tienen tres funciones diferentes en el archivo **utils.py.** Ahora, se debe probar cada función con diferentes casos de prueba. Se escriben las pruebas para la primera función llamada **es_primo()**.

1. Cree un archivo llamado **test_utils.py** en la carpeta de ejemplo como **utils.py**.
2. Importe el módulo **utils** y **unittest**.
3. Cree una clase con el nombre **TestUtils** que extienda la clase **unittest.TestCase**. El nombre de la clase puede ser cualquiera. Trate de dar a la clase un nombre significativo.
4. Dentro de la clase, escriba un método llamado **test_es_primo** que acepte **self** como argumento.

5. Escriba diferentes casos de prueba con argumentos para **es_primo** y compare la salida con la salida esperada.
6. Ejemplo de caso de prueba **self.assertFalse(utils.es_primo(1)).**
7. Esperamos que la salida de **es_primo(1)** sea falsa en el caso anterior.
8. Similar al caso anterior, se hacen diferentes pruebas de casos basados en la función que se está probando.

Veamos las pruebas.

```
import unittest

import utils


class TestUtils(unittest.TestCase):
    def prueba_es_primo(self):
        self.assertFalse(utils.es_prime(4))
        self.assertTrue(utils.es_primo(2))
        self.assertTrue(utils.es_primo(3))
        self.assertFalse(utilidades.es_primo(8))
        auto.assertFalse(utilidades.es_primo(10))
        self.assertTrue(utilidades.es_primo(7))
        self.assertEqual(utils.es_primo(-3),
                         "Los números negativos no están
permitidos")


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

Se está invocando al método **main** de **unittest** el módulo para ejecutar las pruebas utilizando el comando **python filename.py** . Ahora, ejecute las pruebas.

La salida :

```
$ python prueba_utils.py
.
----------------------------------------------------------------
Ejecutó 1 prueba en 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.

```
clase TestUtils(unittest.TestCase):
    def prueba_es_primo(self):
        ...

    def prueba_cubica(self):
        self.assertEqual(utils.cubico(2), 8)
        self.assertEqual(utils.cubico(-2), -8)
        self.assertNotEqual(utils.cubico(2), 4)
        auto.assertNotEqual(utils.cubico(-3), 27)

    def prueba_saludar(self):
        self.assertEqual(utils.decir_hola("Hugo"), "Hola,
Hugo")
        self.assertEqual(utils.decir_hola("Programador"),
"Hola, Programador")
        self.assertNotEqual(utils.decir_hola("Pepe"), "Hola,
Pepe")
        self.assertNotEqual(utils.decir_hola("Maria"), "Hola,
Maria")
```

Se han utilizado sólo algunas de las funciones de comparación del módulo **`unittest`**.

Cómo ejecutar pruebas utilizando unittest

Otras dos formas de ejecutar las pruebas utilizando el módulo **`unittest`**.

1. Utilizando el nombre del archivo y el módulo **`unittest`**.

En este método, se utiliza el módulo **`unittest`** y el nombre del archivo para ejecutar las pruebas. El comando para ejecutar las pruebas es **`python -m unittest nombrearchivo .py`**.

En este caso, el comando para ejecutar las pruebas es **`python -m unittest test_utils.py`**.

2. Utilizar el método **`discover`**

Se utiliza el método **`discover`** del módulo **`unittest`** para autodetectar todos los archivos de prueba y ejecutarlos. Para autodetectar los archivos de prueba, se deben nombrar comenzando con la palabra clave **`test`**.

El comando para ejecutar las pruebas utilizando el método **`discover`** es **`python -m unittest discover`**. El comando detectará todos los archivos cuyos nombres empiecen por **`test`** y los ejecutará.

Las pruebas unitarias son pruebas básicas en el mundo de la programación, existen bibliotecas de terceros como pytest, Robot Framework, nose, nose2, slash, etc., que realizan pruebas similares.

Es su responsabilidad profundizar en la aplicación de pruebas unitarias, ya sea utilizando el módulo **`unittest`** o módulos de terceros.