entrega_6

July 23, 2018

1 Programación para la Bioinformática

1.1 Unidad 6: Testing y calidad del software

1.1.1 Ejercicios y preguntas teóricas

1.1.2 Pregunta téorica 1

Enumera y explica en qué consisten otras librerías Python para probar código (mínimo 3):
Mi respuesta

1.1.3 Pregunta téorica 2

Explica en tus propias palabras el concepto de **cobertura de código** (*test coverage* en Inglés):
Mi respuesta

1.1.4 Pregunta téorica 3

Aunque nos hemos centrado en los tests unitarios, hemos comentado que existen otros tipos de tests. Enumera **al menos dos otros tipos de tests** y explica en tus propias palabras qué uso tienen y en qué consisten.
Mi respuesta

1.1.5 Pregunta teórica 4

£Es posible hacer pruebas cuando utilizamos objetos *Exception* en nuestro código? Si es así, explica cómo y pon como mínimo 2 ejemplos que se puedan ejecutar:
Mi respuesta

1.1.6 Ejercicio 1

Hablamos de tests en verde cuando todos nuestros tests se ejecutan correctamente y dan el resultado esperado y tests en rojo en caso contrario.

En el siguiente ejercicio, escribe el código necesario que haga cumplir todos los tests, es decir, que los tests estén *en verde*.

```
In [1]: import unittest
        import sys
```

```python
class Fraccion(object):
    """Clase que representa una fracción matemática"""

    def __init__(self, numerador, denominador):
        """Inicializa el objeto fracción"""
        # Código a completar
        pass

    def get_numerador(self):
        """Retorna el numerador de la fracción"""
        return self.numerador

    def get_denominador(self):
        """Retorna el denominador de la fracción"""
        # Código a completar
        pass

    def multiplica(self, other):
        """Devuelve la multiplicación de fracciones"""
        return Fraccion(self.get_numerador() * other.get_numerador(), other.get_denomi


class TestFraccion(unittest.TestCase):

    def test_crear_fraccion(self):
        f = Fraccion(1, 2)
        self.assertIsNotNone(f)

    def test_numerador(self):
        f = Fraccion(1, 2)
        self.assertEqual(f.get_numerador(), 1)

    def test_denominador(self):
        f = Fraccion(2, 4)
        self.assertEqual(f.get_denominador(), 4)

    def test_multiplicacion_fracciones(self):
        f1 = Fraccion(1, 2)
        f2 = Fraccion(2, 5)

        f3 = f1.multiplica(f2)

        self.assertEqual(f3.get_numerador(), 2)
        self.assertEqual(f3.get_denominador(), 10)


if __name__ == '__main__':
```

```
            suite = unittest.TestLoader().loadTestsFromTestCase( TestFraccion )
            unittest.TextTestRunner(verbosity=1,stream=sys.stderr).run( suite )
```

```
.FEE
======================================================================
ERROR: test_multiplicacion_fracciones (__main__.TestFraccion)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-1-6b59d35a106f>", line 45, in test_multiplicacion_fracciones
    f3 = f1.multiplica(f2)
  File "<ipython-input-1-6b59d35a106f>", line 24, in multiplica
    return Fraccion(self.get_numerador() * other.get_numerador(), other.get_denominador())
  File "<ipython-input-1-6b59d35a106f>", line 15, in get_numerador
    return self.numerador
AttributeError: 'Fraccion' object has no attribute 'numerador'

======================================================================
ERROR: test_numerador (__main__.TestFraccion)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-1-6b59d35a106f>", line 35, in test_numerador
    self.assertEqual(f.get_numerador(), 1)
  File "<ipython-input-1-6b59d35a106f>", line 15, in get_numerador
    return self.numerador
AttributeError: 'Fraccion' object has no attribute 'numerador'

======================================================================
FAIL: test_denominador (__main__.TestFraccion)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-1-6b59d35a106f>", line 39, in test_denominador
    self.assertEqual(f.get_denominador(), 4)
AssertionError: None != 4

----------------------------------------------------------------------
Ran 4 tests in 0.009s

FAILED (failures=1, errors=2)
```

1.1.7 Ejercicio 2

Escribe una función cualquiera y escribe algunas pruebas de código utilizando ***doctest***:

```
In [2]: # Mi respuesta
```

1.1.8 Ejercicio 3

Los cuaterniones (en Inglés *quaternion*) son un tipo matemático que funcionan como extensión de los vectores en espacio 3D añadiendo una dimensión extra, muy utilizados en videojuegos para aplicar rotaciones sobre un conjunto de puntos y que tienen claras ventajas en comparación con las matrices de rotación. Podéis aprender más sobre ellos en la Wikipedia.

A continuación tenéis el código de una clase *Quaternion* que implementa algunas funciones sencillas (suma y resta). Escribid tantos tests como consideréis para conseguir una buena cobertura de código:

```
In [3]: import unittest
        import sys


        class Quaternion:

            def __init__(self, w=1., x=0., y=0., z=0.):
                """Crea un cuaternión. Devuelve por defecto el cuaternión unitario."""
                self.w = w
                self.x = x
                self.y = y
                self.z = z

            def __neg__(self):
                """Negación de un cuaternión.

                Se llama: q2 = -q1
                """
                return Quaternion(-self.w, -self.x, -self.y, -self.z)

            def __add__(self, other):
                """Implementa la suma de cuaterniones.

                Ejemplo:
                q1 = Quaternion()
                q2 = Quaternion()

                q3 = q1 + q2

                """
                return Quaternion(self.w+other.w, self.x+other.x, self.y+other.y, self.z+other

            def __sub__(self, other):
                """Implementa la resta de cuaterniones."""
                return Quaternion(self.w-other.w, self.x-other.x, self.y-other.y, self.z-other


        class TestQuaternion(unittest.TestCase):
```

```
            # Código a completar
            pass


        if __name__ == '__main__':
            suite = unittest.TestLoader().loadTestsFromTestCase( TestQuaternion )
            unittest.TextTestRunner(verbosity=1,stream=sys.stderr).run( suite )
```

```
----------------------------------------------------------------------
Ran 0 tests in 0.000s

OK
```