ACTIVIDAD 18. Pruebas

Programar pruebas permite controlar la calidad del programa, saber si una modificación ha roto otra parte del código y en cierta forma documentan el código. Ello no garantiza que el programa esté exento de fallos. Básicamente en cada prueba la aplicación debe funcionar cuando las entradas y salidas son correctas y lanzar una excepción cuando los valores son incorrectos.

El módulo ***doctest*** busca partes de la documentación que parezcan sesiones interactivas y las ejecuta y verifica que funcionan correctamente. Veamos el siguiente ejemplo:

```
#Ejemplo de la función factorial().  For example,
import doctest
import math


def factorial(n):
    """Resuelve el factorial de n >= 0.

    Si el resultado es pequeño devolverá un int sino un long

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> [factorial(long(n)) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> factorial(30)
    265252859812191058636308480000000L
    >>> factorial(30L)
    265252859812191058636308480000000L
    >>> factorial(-1)
    Traceback (most recent call last):
        ...
    ValueError: n must be >= 0

    Factorials de números decimales no son aceptados:
    >>> factorial(30.1)
    Traceback (most recent call last):
        ...
    ValueError: n must be exact integer
    >>> factorial(30.0)
    265252859812191058636308480000000L

    No pueden ser numeros demasiado grandes:
    >>> factorial(1e100)
    Traceback (most recent call last):
        ...
    OverflowError: n too large
    """

    if not n >= 0:
        raise ValueError("n debe ser >= 0")
    if math.floor(n) != n:
        raise ValueError("n must be exact integer")
    if n+1 == n:  # catch a value like 1e300
        raise OverflowError("n too large")
    result = 1
    factor = 2
```

```
    while factor <= n:
        result *= factor
        factor += 1
    return result

if __name__ == "__main__":
    doctest.testmod()
```

Para ver el resultado.

```
$python example.py -v
```

Usar este tipo de pruebas ensucia mucho el código. Otra forma sería sacarlo en un fichero aparte (el texto en amarillo) y poner algo así:

```
import doctest

def factorial(n):
    return 1 if n < 1 else n * factorial(n-1)

if __name__ == "__main__":
    doctest.testfile('example2.txt')
```

Hay muchas ocasiones en que las pruebas realizadas con doctest se nos quedarán cortas. El módulo ***Unittest*** nos ofrece probar nuestros programas de una manera más simple. Un ejemplo sencillo:

```
import unittest

def factorial(n):
    return 1 if n < 1 else n * factorial(n-1)

class tester (unittest.TestCase):
    def test_1(self):
        self.assertEqual(1, factorial(1))

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

Otros métodos:

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

Actividades

En el programa que estamos realizando podemos incluir algún módulo de prueba. Por ejemplo, al hacer los cálculos de la factura (con unos datos previos) comprobar que los realiza correctamente.