ETS-UT3-3. Tests en Python con Doctest

comprobando nuestro código

Una vez que hemos generado los casos de prueba para nuestro código debemos aplicar los mismos usando alguna de las herramientas de test que existen para el lenguaje de programación con el que estemos trabajando

Existen para Python diferentes frameworks y herramientas que nos permiten automatizar los tests que aplicamos para validar y comprobar nuestro código.

Normalmente los tests a aplicar los generamos a partir de los casos de prueba que hemos obtenido a partir de las diferentes técnicas existentes. Un ejemplo es la del camino básico que vimos en los apuntes anteriores.

doctest

Una de los módulos para realizar tests para Python más sencillos de utilizar es **doctest**. Viene incluida en la librería estandar de Python

Para su utilización se incluyen fragmentos de código en los comentarios y luego el módulo `doctest` ejecuta dichos tests para confirmar que el código de nuestro programa se comporta de acuerdo a lo esperado.

Incorporando Doctest a una función

Partimos de la función que vimos en la actividad UT3-A2 a la que le pasamos tres parámetros y nos devuelve el mayor:

```
def bigger(a, b, c):
    if a > b and a > c:
        return a
    if c > b:
        return c
    else:
        return b
```

Empezamos añadiendo a la documentación a la función una descripción de lo que realiza:

```
def bigger(a, b, c):
    """
    given thre integers, return the bigger
    """
    if a > b and a > c:
        return a
    if c > b:
        return c
    else:
        return b
```

Hasta aquí nada nuevo.

Es una buena costumbre añadir también información del tipo de los parámetros de la función y del valor que devuelve. La forma de hacerlo es:

```
def bigger(a, b, c):
    """
    given thre integers, return the bigger

    :param a: int
    :param b: int
    :paran c: int
    :return int
    """
    if a > b and a > c:
        return a
    if c > b:
        return c
    else:
        return b
```

> **Nota**: Los dos pasos anteriores son opcionales y no son necesarios para realizar los tests. En cualquier caso es buena constumbre documentar las funciones incluidas en nuestros módulos.

La tabla con los casos de prueba que habíamos obtenido era:

Camino	Entrada	Prueba	Salida
`I-1-2-3-5-6-F`	`a>b=True a>c=False c>b=True`	`a=5 b=3 c=6`	**6**
`I-1-2-3-5-7-F`	`a>b=True a>c=False c>b=False`	`a=5 b=3 c=2`	**5**
`I-1-2-3-4-F`	`a>b=True a>c=True`	`a=7 b=5 c=4`	**7**
`I-1-2-5-6-F`	`a>b=False c>b=True`	`a=5 b=6 c=8`	**8**

Incorporamos los casos de prueba de la siguiente forma:

```
def bigger(a, b, c):
    """
    given thre integers, return the bigger

    :param a: int
    :param b: int
    :paran c: int
    :return int

    >>> bigger(5, 3, 6)
    6
    >>> bigger(5, 3, 2)
    5
    >>> bigger(7, 5, 4)
    7
    >>> bigger(5, 6, 8)
    8
    """
    if a > b and a > c:
```

```
        return a
    if c > b:
        return c
    else:
        return b
```

Vemos que:

- Las líneas en las que se especifica el código a ejecutar empiezan por `>>>` como en el intérprete de Python. Se podría usar también tres puntos seguidos `...`
- A continuación se incluye el resultado esperadode la ejecución del test

Ya solo queda ejecutar los tests. Importamos el módulo `doctest` y hacemos una llamada al tipo de test que queremos ejecutar. Para ello añadimos las siguientes líneas al principio y al final del mismo:

```
import doctest
...
doctest.testmod()
```

El código final sería:

```
import doctest

def bigger(a, b, c):
    """
    given thre integers, return the bigger

    :param a: int
    :param b: int
    :paran c: int
    :return int

    >>> bigger(5, 3, 6)
    6
    >>> bigger(5, 3, 2)
    5
    >>> bigger(7, 5, 4)
    7
    >>> bigger(5, 6, 8)
    8
    """
    if a > b and a > c:
        return a
    if c > b:
        return c
    else:
        return b

doctest.testmod()
```

Para aplicar los tests, si alamacenamos el módulo en un fichero de nombre `bigger.py` desde el terminal ejecutamos:

```
$ python3 bigger.py -v
```

```
Trying:
    bigger(5, 3, 6)
Expecting:
    6
ok
Trying:
    bigger(5, 3, 2)
Expecting:
    5
ok
Trying:
    bigger(7, 5, 4)
Expecting:
    7
ok
Trying:
    bigger(5, 6, 8)
Expecting:
    8
ok
1 items had no tests:
    __main__
1 items passed all tests:
   4 tests in __main__.bigger
4 tests in 2 items.
4 passed and 0 failed.
Test passed.
```

Vemos que la función pasa los tests.

Qué pasaría si hemos cometido un error en nuestra función. Supongamos que en la línea 23 ponemos al reves la comparación.

En lugar de:

```
    if c > b:
```

Ponemos:

```
    if b > c:
```

Ejecutamos los tests sin la opción `-v` para que la salida no salga con tanto detalle:

```
$ python3 bigger.py
**********************************************************************
File "bigger.py", line 10, in __main__.bigger
Failed example:
    bigger(5, 3, 6)
Expected:
    6
Got:
    3
**********************************************************************
File "bigger.py", line 16, in __main__.bigger
Failed example:
    bigger(5, 6, 8)
```

```
Expected:
    8
Got:
    6
**********************************************************************
1 items had failures:
   2 of   4 in __main__.bigger
***Test Failed*** 2 failures.
```

Vemos que nuestra función no ha pasado los tests.

En este caso nos tocaría **depurar** nuestra función para averiguar que errores hemos cometido y solucionarlos.

Aplicando Doctest al programa principal

Los tests que acabamos de generar se aplican dentro de la función si queremos aplicar tests usando la función lo podemos hacer poniendo en comentario **docstring**, de forma similar, fuera de la función

```
import doctest
"""
Test de ejemplo usando funcion en programa ppal
>>> bigger(5, 7, 0)
7
"""

def bigger(a, b, c):
    """
    given thre integers, return the bigger

    :param a: int
    :param b: int
    :paran c: int
    :return int

    >>> bigger(5, 3, 6)
    6
    >>> bigger(5, 3, 2)
    5
    >>> bigger(7, 5, 4)
    7
    >>> bigger(5, 6, 8)
    8
    """
    if a > b and a > c:
        return a
    if c > b:
        return c
    else:
        return b

doctest.testmod()
```

Al ejecutar tests:

```
python3 test-doctest.py -v
Trying:
    bigger(5, 7, 0)
Expecting:
    7
ok
Trying:
    bigger(5, 3, 6)
Expecting:
    6
ok
Trying:
    bigger(5, 3, 2)
Expecting:
    5
ok
Trying:
    bigger(7, 5, 4)
Expecting:
    7
ok
Trying:
    bigger(5, 6, 8)
Expecting:
    8
ok
2 items passed all tests:
   1 tests in __main__
   4 tests in __main__.bigger
5 tests in 2 items.
5 passed and 0 failed.
Test passed.
```

- En la **línea 3** vemos que se ejecuta el test que acabamos de añadir
- En el resumen que se muestra a partir de la línea 27 vemos que se han ejecutado 5 tests:
    - 1 en el programa principal
    - 4 dentro de la función

Invocando tests desde línea de comandos

Si no queremos insertar en nuestro módulo las líneas:

```
import doctest
...
doctest.testmod()
```

Para nuestro ejemplo el fichero `bigger.py` tendría el siguiente contenido:

```
"""
Test de ejemplo usando funcion en programa ppal
>>> bigger(5, 7, 0)
7
"""

def bigger(a, b, c):
```

```
    """
    given thre integers, return the bigger

    :param a: int
    :param b: int
    :paran c: int
    :return int

    >>> bigger(5, 3, 6)
    6
    >>> bigger(5, 3, 2)
    5
    >>> bigger(7, 5, 4)
    7
    >>> bigger(5, 6, 8)
    8
    """
    if a > b and a > c:
        return a
    if c > b:
        return c
    else:
        return b
```

Podemos invocar desde la línea de comandos la ejecución de los test indicando con la opción `-m` el módulo `doctest` al ejecutar nuestro programa:

```
$ python -m doctest -v bigger.py
```

Especificando tests en fichero externo con doctest

Si queremos que nuestros tests no estén incluidos en el cuerpo de la función sino en un fichero externo, **doctest** permite hacerlo. Veámoslo con un ejemplo:

No incluimos los tests en el fichero `bigger.py`

```
# Contenido del fichero bigger.py
def bigger(a, b, c):
    """
    given thre integers, return the bigger

    :param a: int
    :param b: int
    :paran c: int
    :return int
    """
    if a > b and a > c:
        return a
    if c > b:
        return c
    else:
        return b
```

Dentro de la misma carpeta creamos un ficheri de nombre `bigger_tests.txt` con el siguiente contenido:

```
Importamos el módulo

    >>> from bigger import bigger

Hacemos tests

    >>> bigger(5, 3, 6)
    6
    >>> bigger(5, 3, 2)
    5
    >>> bigger(7, 5, 4)
    7
    >>> bigger(5, 6, 8)
    8
```

Vemos que:

- Primero debemos importar el módulo en el que se encuentra la funcion o funciones a las que queremos hacer los tests
- A continuación ejecutamos los tests y mostramos los resultados esperados.
- Las líneas en las que se especifica lo que vamos a ejecutar y el resultado no es necesario que estén al principio de la línea, pueden estar en cualquier posición.

Para ejecutar los tests en la línea de comandos:

```
$ python3 -m doctest -v bigger_tests.txt
Trying:
    from bigger import bigger
Expecting nothing
ok
Trying:
    bigger(5, 3, 6)
Expecting:
    6
ok
Trying:
    bigger(5, 3, 2)
Expecting:
    5
ok
Trying:
    bigger(7, 5, 4)
Expecting:
    7
ok
Trying:
    bigger(5, 6, 8)
Expecting:
    8
ok
1 items passed all tests:
   5 tests in bigger_tests.txt
5 tests in 1 items.
5 passed and 0 failed.
Test passed.
```

> Fíjate que ejecutamos el fichero de texto y no el fichero que contiene la función.

Recursos

- Doctest - Documentación oficial de Python
- Doctest - Hector Docs