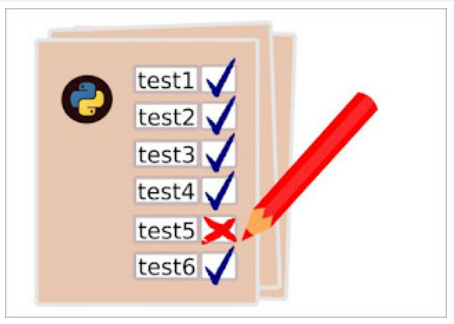
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

sábado, 23 de marzo de 2019

Probando el código con doctest



El módulo **doctest** forma parte de la librería estándar de Python y se utiliza para probar el código aprovechando su propia documentación.

Este módulo localiza información en el código (con la apariencia de una sesión interactiva de Python) situada entre los propios textos de la documentación o **docstrings**, que contienen tanto las pruebas a realizar como los resultados que se esperan obtener. Y cuando estas pruebas se ejecuten se generará un informe con los éxitos y errores que se produzcan.

El módulo **doctest** se puede utilizar para comprobar que la documentación está debidamente actualizada, pudiéndose verificar después de cada cambio del código que las pruebas funcionan tal como recoge la documentación; para realizar pruebas que detecten errores, carencias o incongruencias; y para realizar los tutoriales de un paquete con ejemplos que ilustren su funcionamiento a los usuarios.

Pruebas insertadas en el código fuente

Sirva el siguiente ejemplo para mostrar la facilidad de uso de **doctest**. En el ejemplo hay una función llamada **impar()** que en su documentación incluye cuatro pruebas precedidas por los caracteres "**>>>**" (los mismos que se utilizan en una sesión interactiva Python para indicar el lugar donde se introducen los comandos), incluyendo debajo de cada prueba el resultado esperado en cada una de ellas.

fmatemat1.py:

```
def impar(n):
    '''
    La función impar(n) devuelve:
    -  True: si número es impar
    - False: si número no es impar
    >>> impar(0)
    False
    >>> impar(1)
    True
    >>> impar(2)
    False
    >>> impar(3)
    True
    '''
    if n%2 != 0:
        return True
    else:
        return False

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

A continuación, ejecutar el código con la opción **-v** (verbose) para mostrar un informe con el resultado de las pruebas:

```
$ python3 fmatemat1.py -v
```

También, se obtendría el mismo informe omitiendo la clausula **if** y las líneas siguientes del código, ejecutando con:

```
$ python3 -m doctest fmatemat1.py -v
```

En cualquiera de los casos en la salida se obtiene un informe con el resultado de cada prueba, que en esta ocasión termina con el mensaje: *"4 passed and 0 failed. Test passed."*:

```
Trying:
    impar(0)
Expecting:
    False
ok
Trying:
    impar(1)
Expecting:
    True
ok
Trying:
    impar(2)
Expecting:
    False
ok
Trying:
    impar(3)
Expecting:
    True
ok
1 items had no tests:
    fmatemat
1 items passed all tests:
   4 tests in fmatemat.impar
4 tests in 2 items.
4 passed and 0 failed.
Test passed.
```

Incluir la opción **-v** es equivalente a añadir el parámetro **verbose=True** a la función **testmod()**.

Otra posibilidad de ejecución consiste en omitir la opción **-v** para mostrar sólo información relacionada con las pruebas que fallen.

Pruebas independientes del código fuente

Si el archivo .py es un módulo con muchas funciones y/o clases que se prevee incluya abundantes pruebas, otra alternativa que dejará además el código despejado consiste en extraer toda la documentación y las pruebas (o al menos las pruebas) a un fichero de texto.

En ese caso tendríamos dos archivos como en el siguiente ejemplo: uno con el código fuente como (fmatemat2.py) que utiliza la función **testfile()** para referenciar al archivo de texto y el propio archivo de texto (fmatemat2.txt) con la documentación y las pruebas.

fmatemat2.py:

```
def impar(n):
    if n%2 != 0:
        return True
    else:
        return False

if __name__ == '__main__':
    import doctest
    doctest.testfile('fmatemat2.txt')
```

fmatemat2.txt:

```
Módulo fmatemat2
================

Para utilizar la función 'impar'
```

simultáneamente varias operaciones en el mismo espacio de proceso se...

Archivo

marzo 2019 (1)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

```
---------------------------------

La función impar(n) devuelve:

    - True si número es impar o
    - False si no lo es

Primero importamos la función:

    >>> from fmatemat2 import impar

Ejemplos de uso:

    >>> impar(0)
    False
    >>> impar(1)
    True
    >>> impar(2)
    False
    >>> impar(3)
    True
```

En este caso se contabiliza una prueba más por la línea en la que se importa la función **impar**. Para obtener un informe con el resultado de las pruebas ejecutar:

```
$ python3 fmatemat2.py -v
```

Escribiendo pruebas para doctest. Generalidades

Para poder escribir pruebas para **doctest** resulta necesario conocer el modo en que se analiza el texto que incluye dichas pruebas:

- En primer lugar indicar que se analizan todas las cadenas de documentación de las funciones, clases y métodos existentes del módulo actual y no son analizados los módulos importados. En ese caso lo que si es posible es declarar una variable **__test__** con un diccionario con el conjunto de pruebas. En el diccionario las claves son cadenas que se corresponden con nombres de funciones o métodos (del módulo importado) a los que se les asigna un **docstring** con una o más pruebas:

fmatemat3.py:

```
from fmatemat1 import impar

__test__ = {
'impar':"""
>>> impar(5)
True

>>> impar(6)
False
"""
}

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

- La columna o posición en una línea donde se inicie la escritura de una prueba no es significativa.
- Los resultados de una prueba (si existen) deben situarse a continuación de líneas que contienen las cadenas '**>>>**' o '**...**'.
- Los comentarios precedidos con el carácter **#** y las asignaciones no devuelven ningún resultado.
- El resultado de una prueba no debe contener una línea en blanco porque **doctest** lo interpretará como el final de la salida de una prueba. Para que sea interpretado como parte de ella es necesario incluir la etiqueta <BLANKLINE> en la salida esperada.
- Cada vez que **doctest** encuentra una cadena de documentación para pruebas utiliza una copia de las variables globales para que las pruebas no modifiquen el valor de las variables reales. Esto implica que se puedan utilizar con libertad nombres de variables definidas en el código que serán también independientes de otras cadenas de documentación, si las hubiere.

El siguiente ejemplo contiene una prueba un poco más sofisticada que incluye un comentario, una importación, asignaciones y la clausula **if**.

fmatemat4.py:

```
def impar(n):
    '''
    La función impar(n) devuelve:
    -  True: si número es impar
    - False: si número no es impar
    >>> # Obtiene número de día del año y verifica que sea impar
    >>> from datetime import datetime
    >>> numdia = datetime.now().timetuple().tm_yday
    >>> if numdia < 183:
    ...     semestre = 1
    ... else:
    ...     semestre = 2
    >>> impar(numdia)
    True
    >>> impar(9)
    True
    '''
    if n%2 != 0:
        return True
    else:
        return False

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Ejecutar las pruebas con:

```
$ python3 fmatemat4.py -v
```

```
Trying:
    from datetime import datetime
Expecting nothing
ok
Trying:
    numdia = datetime.now().timetuple().tm_yday
Expecting nothing
ok
Trying:
    if numdia < 183:
        semestre = 1
    else:
        semestre = 2
Expecting nothing
ok
Trying:
    impar(numdia)
Expecting:
    True
ok
Trying:
    impar(9)
Expecting:
    True
ok
1 items had no tests:
    __main__
1 items passed all tests:
   5 tests in __main__.impar
5 tests in 2 items.
5 passed and 0 failed.
Test passed.
```

- El resultado de una prueba puede ser el mensaje que devuelve una excepción si se expresa de manera literal, aunque **doctest** hará bien su trabajo cuando las salidas contengan detalles que cambian dependiendo de un valor como un número de línea, una ruta de un archivo, o bien, cuando se omitan estos detalles:

fmatemat5.py:

```
def impar(n):
    '''
    La función impar(n) devuelve:
```

```
    -  True: si número es impar
    - False: si número no es impar
    >>> impar(0)
    False
    >>> impar('uno')
    Traceback (most recent call last):
    TypeError: not all arguments converted during string formatting
    '''
    if n%2 != 0:
        return True
    else:
        return False

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Ejecutar las pruebas con:

```
$ python3 fmatemat5.py -v
```

```
Trying:
    impar(0)
Expecting:
    False
ok
Trying:
    impar('uno')
Expecting:
    Traceback (most recent call last):
    TypeError: not all arguments converted during string formatting
ok
1 items had no tests:
    __main__
1 items passed all tests:
   2 tests in __main__.impar
2 tests in 2 items.
```

- Si en una salida se esperan tabuladores es recomendable habilitar la directiva NORMALIZE_WHITESPACE para una correcta evaluación de las pruebas.
- Para poder utilizar barras invertidas '\' en salidas como parte de los mensajes sin producir efectos adversos (por ejemplo, produciendo un salto de línea '**\n**') deben incluirse en cadenas crudas (**raw**) o bien escribirse por duplicado '****'.

Opciones para modificar el comportamiento de doctest

Existen una serie de opciones o banderas que permiten controlar el comportamiento de **doctest**. Estas opciones están disponibles como constantes del módulo representadas como expresiones binarias, como directivas y desde la línea de comandos con la opción **-o** (desde Python versión 3.4).

Relacionado:

- Docstrings.
- Solucionando errores con el depurador.
- El depurador PuDB.

Ir al índice del tutorial de Python

Publicado por Pherkad en 4:50

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua