.UBA fiuba

FACULTAD DE INGENIERÍA

Test unitario (Unittest)

Algoritmos y Programación I

rev1.1 - Diego Serra

Test Unitario:

Una prueba unitaria es una pieza de código automatizada, construida para invocar a una unidad de trabajo pequeña de un sistema y realizar validaciones sobre los supuestos establecidos del comportamiento de dicha unidad de trabajo.

DocTest

Módulo doctest:

El módulo doctest busca pedazos de texto que lucen como sesiones interactivas de Python, y entonces ejecuta esas sesiones para verificar que funcionen exactamente como son mostradas.

Nos permite ejecutar pruebas unitarias.

Ejemplo:

```
def es_par(numero):
    """
    Función que indica si el número es par o no
    >>> es_par(2)
    True
    >>> es_par(3)
    False
    >>> es_par(0)
    True
    """
    return numero % 2 == 0

def mostrar_resultado(resultado):
  print('Es par' if resultado else 'No es par')

def main():
  resultado = es_par(9)
  mostrar_resultado(resultado)
```

Ejecutar:

Asumiendo que el código anterior se encuentra en un archivo llamado prueba.py , para ejecutar los test debemos invocar al modulo doctest desde el interprete para que analice nuestro archivo.

```
_>py -m doctest -v prueba.py
```

Resultado:

En el caso de éxito nos indicará que todos los test estuvieron ok:

3 passed and 0 failed.
Test passed.

En el caso de falla nos indicará la cantidad de tests que fallaron. además de cuales han fallado:

2 passed and 1 failed.
*****Test Failed*** 1 failures**

Ejemplo con invocación a main:

```
def es_par(numero):
    """
    Función que indica si el número es par o no
    >>> es_par(2)
    True
    >>> es_par(3)
    False
    """
    return numero % 2 == 0

def mostrar_resultado(resultado):
  print('Es par' if resultado else 'No es par')

def main():
  resultado = es_par(9)
  mostrar_resultado(resultado)

main()
```

Problema:

Ahora se presenta el problema que cuando ejecutamos las pruebas también se invoca a la función main, cosa que no deseamos al ejecutar las pruebas de la forma que se indicó.

Solución:

Para resolver esto haremos uso de:

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

Cuando el interprete ejecuta un archivo, previamente setea algunas variables del sistema, como por ejemplo la variable __name__

Solución:

La variable **__name__ tomará el valor '__main__'** cuando el intérprete ejecuta de forma directa el archivo.
Tomará otro nombre cuando no es ejecutado de forma directa y es importado como módulo.

Ejemplo con invocación a main:

```
def es_par(numero):
    """
    Función que indica si el número es par o no
    >>> es_par(2)
    True
    >>> es_par(3)
    False
    """
    return numero % 2 == 0


def mostrar_resultado(resultado):
  print('Es par' if resultado else 'No es par')


def main():
  resultado = es_par(9)
  mostrar_resultado(resultado)


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

Esto nos permite poder ejecutar el programa de forma normal, o bien ejecutar los test con doctest

Ejecutar:

Ahora estamos en condiciones de ejecutar el programa de forma directa o bien los tests de forma independiente.

Ejecutar Programa

```
_>py prueba.py
```

Ejecutar Tests

```
_>py -m doctest -v prueba.py
```

Links:

Documentación Doctest:
https://docs.python.org/es/3/library/doctest.html