Pruebas con Pytest

Antonio Espín Herranz

Instalación

- **pip install pytest**
- Es posible que sea necesario abrir una consola como administrador.

- Prueba para saber si se ha instalado correctamente: python

  >>>import pytest

  – Si no da error es que está instalada.

Test

- Una vez que se ha instalado el módulo pytest disponemos de un **comando pytest** que lanzaremos desde la línea de comandos (consola MSDOS) .

- El comando localiza los ficheros que empiecen por **test_*.py** o que terminen por la palabra ***_test.py**. En directorios y subdirectorios.
  - **test_**funciones.py
  - **test_**mis_clases.py
  - Funciones**_test**.py

- Se puede lanzar sólo un fichero de test con el comando:
  - ***pytest –q test_ejemplo.py***

Test

- Cuando se lanza el comando, da un informe de los ficheros que han fallado, ejecuta todos los ***métodos de una clase o funciones que empiecen por test***.

```
test_class.py .F
test_sample.py F
test_sysexit.py .
```

- A parte informa de cada método o función que ha fallado.
  - **F** → Fallo, dentro del fichero correspondiente
  - **El punto** . Uno de los métodos ha ido bien.
  - Al final emite un resumen de todos los test.

Ejemplo

- **Con funciones**:

```
def func(x):
    return x + 1
```

```
    def test_answer():
>       assert func(3)==5
E       assert 4 == 5
E        +  where 4 = func(3)

test_sample.py:7: AssertionError
```

```
def test_answer():
    assert func(3)==5
```
→ **F**

 # Comprueba la salida de la función con el valor esperado si no, coincide emite el error.

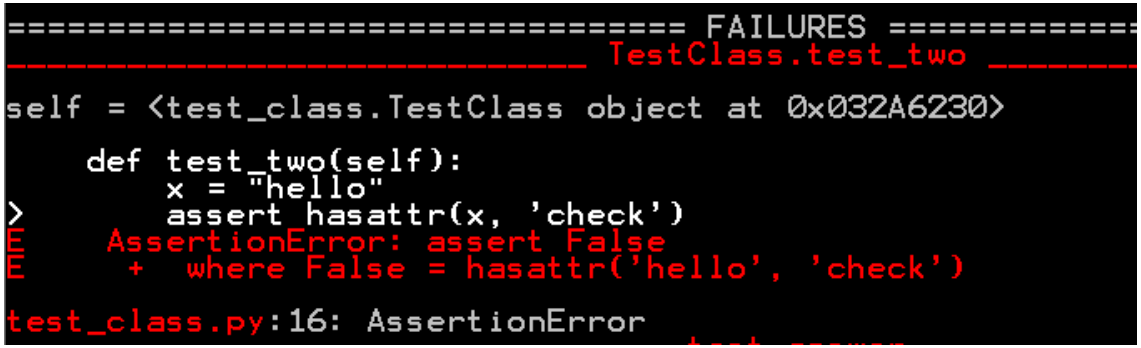
Ejemplo

- **Con clases:**

```
class TestClass:

    def test_one(self):
        x = "this"
        assert 'h' in x  → OK

    def test_two(self):
        x = "hello"
        assert hasattr(x, 'check') → F
```



- En la clase **NO** hay que **implementar** el método **__init__**

Assert

- El **assert** es una instrucción de Python que te permite definir condiciones que deben cumplirse siempre.
- Se utilizan para **comparar resultados** esperados con la ejecución de métodos, o comprobaciones que obtengamos un True o un False.
- Cuando las expresiones nos devuelven False se produce un **AssertionError** y test fallará.

Pruebas de duración de Tiempo

- Esto sirve para averiguar cuales son las pruebas más lentas.
- Cuando llamamos a pytest se le puede pasar el parámetro ***--durations=n*** y nos devuelve cuales son las n pruebas más lentas.
- Si queremos ver el tiempo de todas las pruebas, indicar en n el número de pruebas que tenemos.

Pruebas de duración de Tiempo

- Por ejemplo, en pruebas de cuanto tarda en calcular 1000 veces el factorial de 900. Y tenemos dos funciones: una iterativa y otra recursiva. Si queremos ver el tiempo de ambas para compararlo lanzar el comando:

  pytest -- durations=2 **pytest -vv**

```
collected 2 items

test_factorial.py ..                                                    [100%]

============================ slowest 2 test durations ============================
0.97s call     test_factorial.py::test_fact
0.63s call     test_factorial.py::test_fact2
============================ 2 passed in 1.62 seconds ============================
```

IMPLEMENTAR