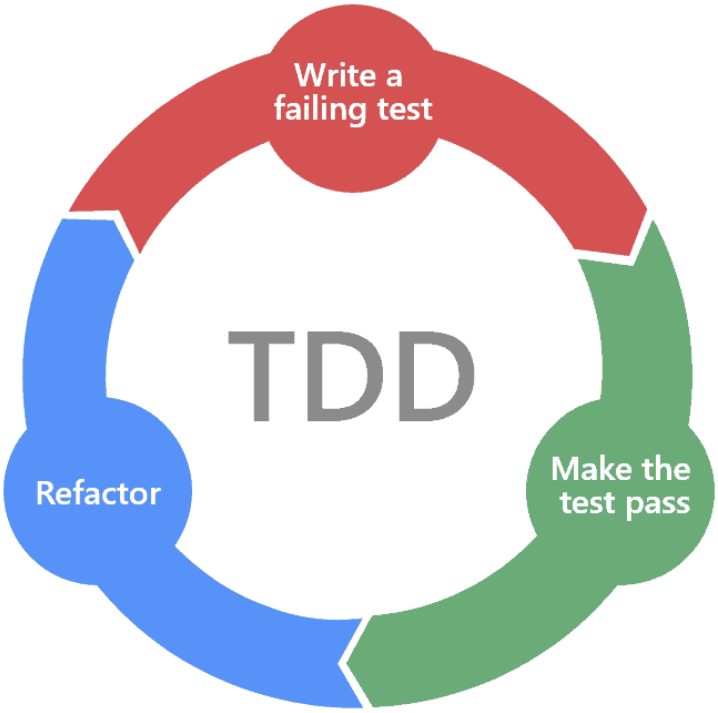
TDD¶

Test Driven Developement en Python¶



El desarrollo guiado por pruebas de software, o Test-driven development (TDD) es una práctica de ingeniería de software que involucra otras dos prácticas: Escribir las pruebas primero (Test First Development) y Refactorización (Refactoring). Para escribir las pruebas generalmente se utilizan las pruebas unitarias (unit test en inglés). En primer lugar, se escribe una prueba y se verifica que la nueva prueba falla. A continuación, se implementa el código que hace que la prueba pase satisfactoriamente y seguidamente se refactoriza el código escrito. El propósito del desarrollo guiado por pruebas es lograr un código limpio que funcione. La idea es que los requisitos sean traducidos a pruebas, de este modo, cuando las pruebas pasen se garantizará que el software cumple con los requisitos que se han establecido.

¿Como podemos implementarlo en Python?¶

Si bien la biblioteca estándar de Python viene con un marco de prueba unitario llamado unittest, Pytest es el marco de prueba de referencia para probar el código Python.

Pytest hace que sea fácil escribir, organizar y ejecutar pruebas. En comparación con unittest, de la biblioteca estándar de Python, Pytest:

1. Requiere menos código repetitivo para que sus conjuntos de pruebas sean más legibles.
2. Soportes de la llanura assertdeclaración, que es mucho más fácil de leer y más fácil de recordar en comparación con los assertSomethingmétodos - como assertEquals, assertTruey assertContains- en unittest.
3. Se actualiza con más frecuencia ya que no forma parte de la biblioteca estándar de Python.

4. Simplifica la configuración y el desmontaje del estado de prueba con su sistema de fijación.
5. Utiliza un enfoque funcional.

Además, con Pytest, puede tener un estilo coherente en todos sus proyectos de Python. Digamos que tiene dos aplicaciones web: una construida con Django y la otra construida con Flask. Sin Pytest, lo más probable es que aproveche el marco de prueba de Django junto con una extensión de Flask como Flask-Testing. Por lo tanto, sus conjuntos de pruebas tendrían diferentes estilos. Con Pytest, por otro lado, ambos conjuntos de pruebas tendrían un estilo de código coherente, lo que facilitaría el salto de uno a otro.

Pytest también tiene un gran ecosistema de complementos mantenido por la comunidad.

Algunos ejemplos:

- Pytest-django : proporciona un conjunto de herramientas creadas específicamente para probar aplicaciones de Django
- Pytest-xdist : se usa para ejecutar pruebas en paralelo
- Pytest-cov : agrega soporte de cobertura de código
- Pytest-instafail : muestra fallas y errores inmediatamente en lugar de esperar hasta el final de una ejecución

Pytest¶

Lo primero que tenemos que hacer si queremos utilizar Pytest para testear nuestro código es instalarlo para ello abriremos una nueva terminal e introduciremos el siguiente comando. Tal y como se puede ver estamos haciendo uso de pip el gestor de paquetes de Python que previamente habíamos instalado.

```
pip install pytest
```

Una vez hecho esto podemos comprobar la versión que tenemos instalada escribiendo el siguiente comando:

```
pytest --version
```

Y obtendríamos un resultado como este:

```
This is pytest version 4.6.9, imported from /usr/lib/python2.7/dist-packages/pytest.pyc
```

Una vez lo hemos instalado y hemos comprobado la versión que tenemos llega la hora de empezar a realizar nuestros primeros test.

A la hora de escribir las pruebas es necesario que tanto los ficheros donde las vamos a escribir como las mismas funciones de prueba dentro del fichero comiencen con el prefijo test_ ya que si no cuando llamemos a pytest pasandole la ruta del directorio este no las encontrará.

Un ejemplo de llamada a Pytest sería algo asi:

pytest /home/marcos/Desktop/Curso

Sintaxis¶

El funcionamiento de pytest es similar al de todas las librerías de testing. Esto lo podemos observar tanto en su sintaxis como en la forma de implementar los test. En este caso vamos ha hacer la comparativa entre gtest (el cual muchos de vosotros conocereis de la carrera) y pytest.

Python¶

```
assert Val1 == Val2   #¿Val1 igual Val2?

assert Val1 != Val2   #¿Val1 diferente Val2?

assert Val1 <  Val2   #¿Val1 menor que Val2?

assert Val1 <= Val2   #¿Val1 menor o igual que Val2?

assert Val1 >  Val2   #¿Val1 mayor que Val2?

assert Val1 >= Val2   #¿Val1 mayor o igual que Val2?

assert Val1 == TRUE   #¿Val1 igual TRUE?

assert Val1 == FALSE  #¿Val1 igual FALSE?
```