6_Ecrire un test avec Unittest

Unittest est le framework de test fourni par défaut avec Python. Vous avez pu voir dans la précédente comment mettre en place des tests unitaires à l'aide du framework Pytest. Nous allons maintenant voir comment faire la même chose avec Unittest. Il n'y a presque pas de différence.

Comme pour Pytest, il faudra aussi définir une fonction qui contiendra le **scénario** et les **assertions** qui permettent de valider le test. De plus, l'organisation des fichiers de tests dans le projet restera identique.

Contrairement à Pytest, les tests doivent être codés dans une classe héritant de la classe `TestCase` du module Unittest. Les scénarios seront implémentés comme des **méthodes de cette classe**.

De plus, pour **exécuter** l'ensemble des tests du module, nous devons appeler la fonction `main` du module `unittest`. Ainsi, pour **lancer** l'ensemble des tests, nous devons ajouter à la fin du fichier un `main` avec l'appel de la fonction `unittest.main()`

```
if __name__ == "__main__":
    unittest.main()
```

D'autre part, la classe `TestCase` fournit un ensemble de **méthodes** permettant de réaliser nos assertions. Voici les plus communes :

Méthode	Vérifie que
assertEqual(a, b)	a == b
assertNotEqual(a, b)	a != b
assertTrue(a)	bool(a) is True
assertFalse(a)	bool(a) is False
assertIs(a, b)	a is b
assertIsNot(a, b)	a is not b
assertIsNone(a)	a is None
assertIsNotNone(a)	a is not None
assertIn(a, b)	a in b
assertNotIn(a, b)	a not in b
assertIsInstance(a, b)	isinstance(a, b)
assertNotIsIntance(a, b)	not isinstance(a, b)

Pour lancer le test :

```
python -m unittest nom_du_fichier
```

Résumé des différentes étapes pour implémenter un test :

- Importez le module `unittest` en haut du fichier de test.
- Créez la classe de test, la classe fille de `unittest.TestCase` .
- Implémentez votre scénario en créant une méthode commençant par “test”.

- Ajoutez le `main` à la fin du fichier.
- Lancez la commande adaptée à votre situation sur le terminal.

Exemple :

Mettons le code suivant dans un fichier test_string.py

```
import unittest

class TestString(unittest.TestCase):
    def test_should_capitalize_string(self):
        string = "hello"
        expected_value = "Hello"
        self.assertEqual(string.capitalize(), expected_value)


    def test_should_upper_string(self):
        string = "hello"
        expected_value = "HELLO"
        self.assertEqual(string.upper(), expected_value)


    def test_should_trim_string(self):
        string = "  hello "
        expected_value = "hello"
        self.assertEqual(string.strip(), expected_value)


if __name__ == "__main__":
    unittest.main()
```

Pour tester :

```
python -m unittest test_string
```

```
aluboya@PCPW11-PAR-001 MINGW64 ~/Documents/MES COURS/M/Tests Unitaires/TP/1/CM
$ python -m unittest test_string
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Exercices

Reprenons le projet de la calculatrice et le même plan de test, pour ajouter l'ensemble des tests unitaires avec le framework de test **Unittest**.

Votre mission :

- Ajoutez un package de tests qui contiendra l'**arborescence de test**.
- Créez la suite de tests concernant le module **view** avec Unittest.
- Créez la suite de tests concernant le module **operators** avec Unittest.

En résumé

- Avec le framework de test Unittest, les scénarios seront implémentés dans une **classe** héritant de `TestCase` et comme des **méthodes de cette classe**.
- Unittest fournit un ensemble de **méthodes** permettant de réaliser nos **assertions**.
- Il faut ajouter le `main` de Unittest à la fin du fichier.
- Lancez les tests d'un module de test à l'aide de la commande `python -m unittest <nom du module de test>`. Il existe d'autres commandes pour affiner le lancement des tests.