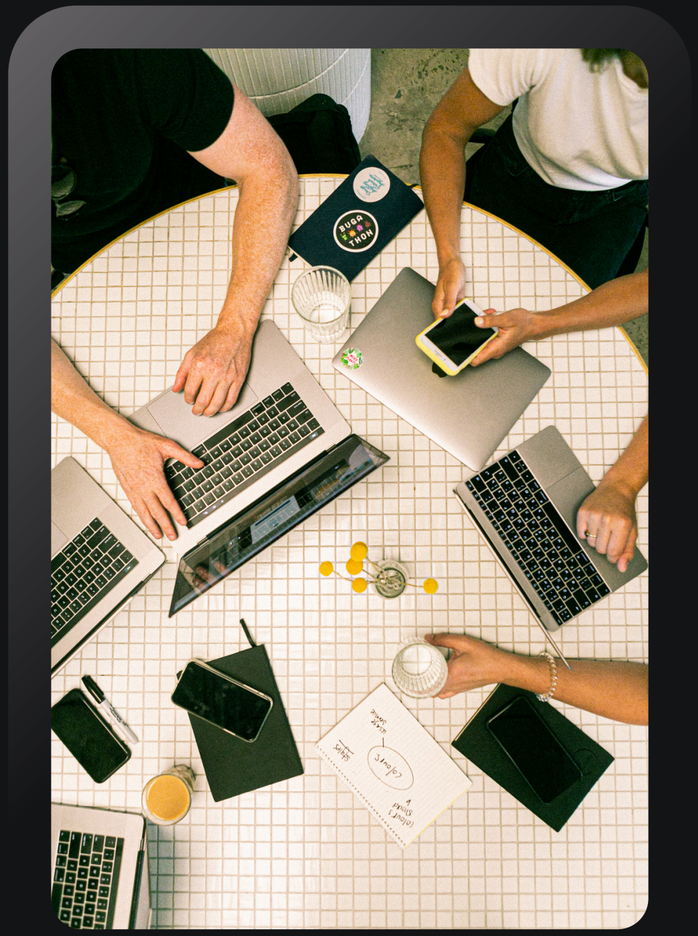
Les tests unitaires

Assurer un code de qualité avec tests automatisés

Au planning ...

- Définition
- Enjeux
- Les principes-clés
- Frameworks
- Exemple
- Automatisation



Définition

Les tests unitaires consistent à tester automatiquement les plus petites unités/composantes d’un code plus complet.

Typiquement, on teste les résultats retournés par des fonctions, des classes, et des méthodes de classes pour s’assurer que le résultat obtenu correspond bien à ce que l’on attend.

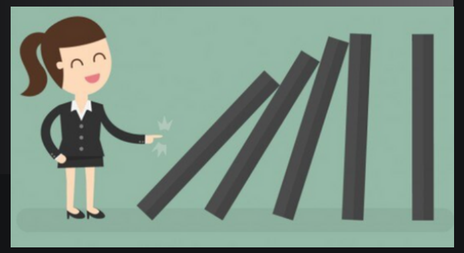
Par exemple, on pourrait tester une fonction qui effectue une somme en lui passant divers paramètres pour s’assurer que la réponse est toujours correcte. L’idée est de garantir que le programme ne va pas "casser" ou retourner un résultat erroné. Exemple : comment notre code se comporterait-il s’il devait faire une opération arithmétique sur des chaînes de caractères ?

Enjeux

Détection précoce des bugs

En éliminant les bugs dès le début, on évite de construire sur une base instable.

Ainsi, il sera moins nécessaire de revenir en arrière plus tard dans le projet.



Débogage simplifié

Si les composants du code sont correctement isolés, il est beaucoup plus facile d'identifier le bloc problématique et de le réparer sans affecter le reste du code.

Augmente la qualité du code

Lors de la mise en place de tests unitaires, on est "forcé" à mieux structurer son code et à créer des composants qui fonctionnent individuellement et sont donc plus facilement réutilisables.

Le code devient également beaucoup plus lisible, en plus d'être moins sujet aux bugs.

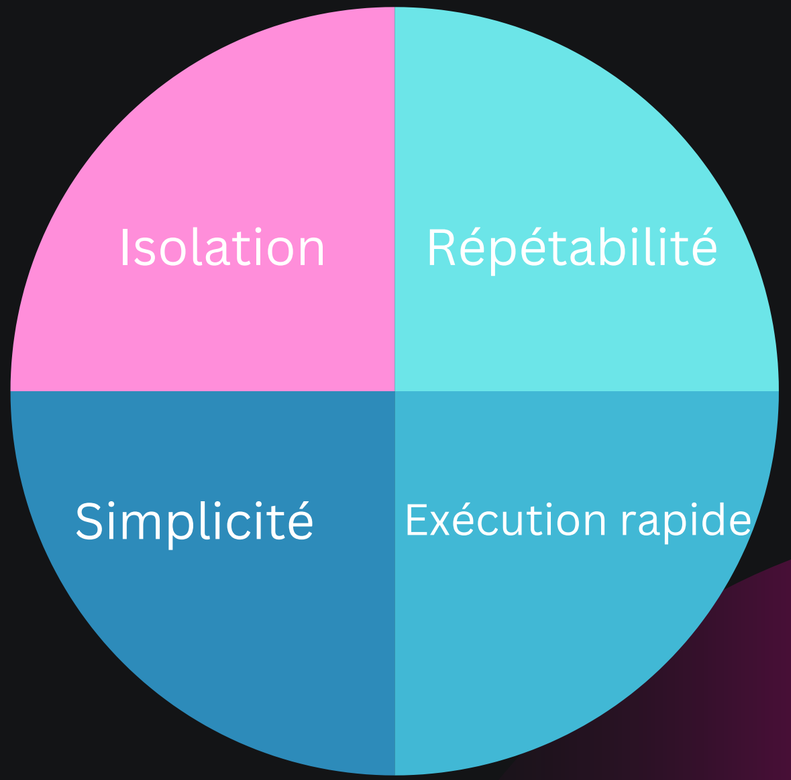
Une documentation efficace

Les tests unitaires peuvent servir à documenter le code. Chaque test décrit le fonctionnement d'une partie de code, et du résultat qui sera renvoyé.

Vu que les tests unitaires sont directement liés au code, ils sont mis à jour en même temps que notre code.

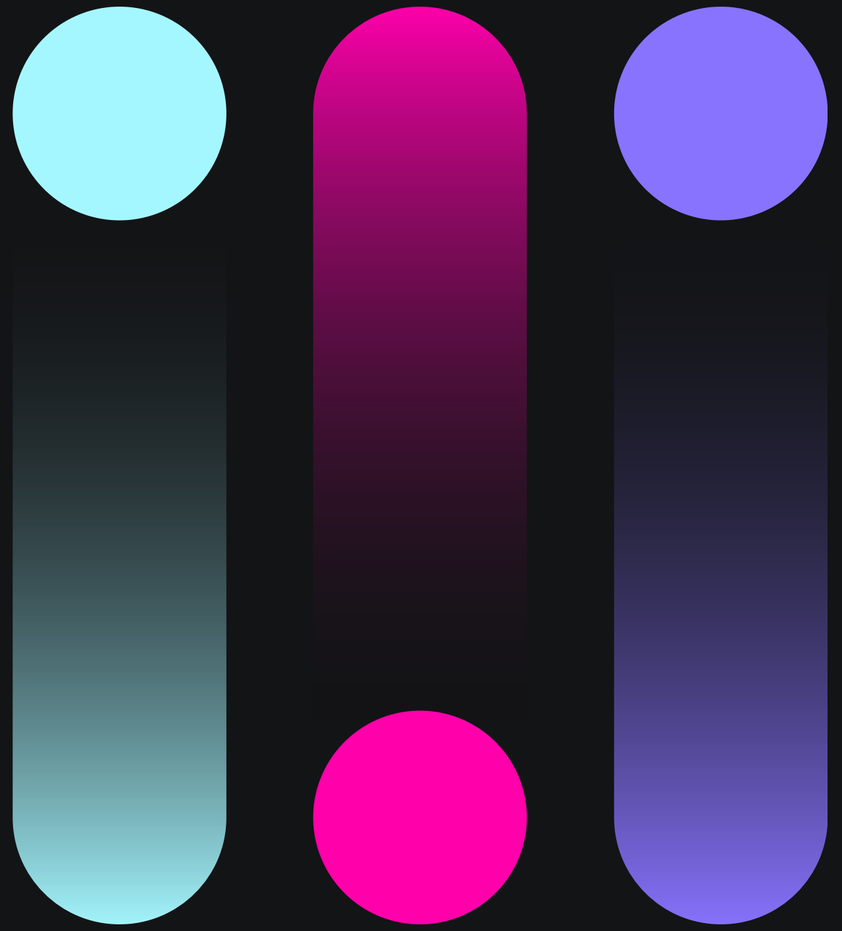
Les principes clés

- **Isolation:** Les tests doivent être exécutés indépendamment
- **Répétabilité:** Les tests doivent produire le même résultat à chaque fois
- **Simplicité:** Chaque test doit se concentrer sur une fonctionnalité unique
- **Exécution rapide:** Les tests doivent être rapides à exécuter



Quelques frameworks de tests unitaires

- Python :
  - unittest
  - pytest
- Java :
  - JUnit
  - TestNG
- JavaScript :
  - Mocha
  - Jasmine
- C# :
  - NUnit
  - MSTest
- PHP :
  - PHPUnit
  - Codeception



Un exemple

```python
import unittest
from math_functions import add

class TestMathFunctions(unittest.TestCase):

    def test_add(self):
        self.assertEqual(add(1, 2), 3)
        self.assertEqual(add(-1, 1), 0)
        self.assertEqual(add(0, 0), 0)
        self.assertEqual(add(-1, -1), -2)
        self.assertEqual(add(1.5, 2.5), 4.0)

    def test_add_with_strings(self):
        with self.assertRaises(TypeError):
            add('a', 'b')

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

→ Résultat :

```
...
----------------------------------------
Ran 2 tests in 0.001s

OK
```

Automatisation des tests

Le testing est d’autant plus utile s’il s’effectue automatiquement.
Par exemple, à chaque modification du code (à chaque commit, à chaque pull request)

Quelques outils pour mettre automatiser le testing :

- Jenkins : Serveur d'automatisation open-source, extensible via des plugins.
- Travis CI : Service CI intégré avec GitHub, facile à configurer via un fichier .travis.yml.
- GitHub Actions : Plateforme CI/CD intégrée à GitHub, offrant des workflows personnalisables.
- GitLab CI : Service CI/CD intégré à GitLab, défini via un fichier .gitlab-ci.yml.

Thank You