Documentation sur les Tests Unitaires

1. Introduction

Les tests unitaires sont des tests automatisés qui vérifient le bon fonctionnement d'une **petite unité de code** (comme une fonction ou une méthode) de manière isolée. Ils permettent d'identifier rapidement les erreurs et de garantir que chaque composant se comporte comme prévu.

---

2. Pourquoi Utiliser des Tests Unitaires ?

- **Détection Précoce des Erreurs** : En testant chaque unité individuellement, tu identifies rapidement les bugs.
- **Facilite la Maintenance** : Lors de modifications ou de refactorisations, les tests t'assurent que les fonctionnalités existantes ne sont pas cassées.
- **Documentation Vivante** : Les tests servent aussi d'exemples sur la manière d'utiliser le code.
- **Fiabilité et Qualité** : Ils améliorent la qualité globale du code et renforcent la confiance lors des déploiements.

---

3. Comment Rédiger un Test Unitaire ?

1. **Identifier l'Unité à Tester** : Choisis une fonction, une méthode ou un composant spécifique.
2. **Définir le Comportement Attendu** : Détermine quelles sont les entrées et sorties prévues (cas normal, cas limite, cas d'erreur).

3. **Écrire le Test :** Utilise un framework de test pour créer un script qui exécute l'unité avec des entrées précises et compare le résultat obtenu avec le résultat attendu.
4. **Exécuter le Test :** Lance le test pour vérifier s'il passe.
5. **Corriger et Réexécuter :** En cas d'échec, corrige le code et réexécute le test pour confirmer la résolution du problème.

---

4. Exemples Pratiques

Exemple en JavaScript avec Jest

```
js

Copier

// Fonction à tester

function addition(a, b) {

  return a + b;

}


// Test unitaire avec Jest

test('addition de 2 + 3 doit retourner 5', () => {

  expect(addition(2, 3)).toBe(5);

});
```

Exemple en PHP avec PHPUnit

```
php
```

Copier

```
// Fonction à tester
function addition($a, $b) {
  return $a + $b;
}

use PHPUnit\Framework\TestCase;

class AdditionTest extends TestCase {
  public function testAddition() {
    $this->assertEquals(5, addition(2, 3));
  }
}
```

---

5. Outils Courants pour les Tests Unitaires

- **JavaScript** : Jest, Mocha, Jasmine
- **PHP** : PHPUnit
- **Python** : unittest, pytest
- **Java** : JUnit

---

6. Bonnes Pratiques

- **Isolation** : Chaque test doit vérifier une seule fonctionnalité pour éviter les interférences.
- **Indépendance** : Les tests doivent pouvoir être exécutés dans n'importe quel ordre, sans dépendre les uns des autres.
- **Lisibilité** : Le code des tests doit être clair et bien documenté, afin de faciliter leur compréhension et leur maintenance.
- **Intégration Continue** : Automatise l'exécution des tests avec des outils CI/CD pour détecter rapidement les régressions.