# Tests unitaires

## Présentation

Les tests unitaires sont essentiels pour maintenir la qualité et la fiabilité du code.

Les tests unitaires sont conçus pour vérifier le bon fonctionnement des "unités" de code, qui peuvent être des méthodes, des classes, ou des fonctions, de manière **isolée**.

### Isolation

Chaque test unitaire se concentre sur une petite partie du code, sans dépendre de l'état global du système ou d'autres composants. Ceci permet de détecter précisément où les erreurs se produisent et de garantir que chaque unité fonctionne comme prévu.

### Automatisation

Les tests peuvent être automatisés, ce qui signifie qu'ils peuvent être exécutés à chaque modification du code sans intervention humaine. Cela facilite la détection rapide des régressions et des bugs introduits lors des changements.

## Bonnes pratiques

### Nommage Clair

Nommez vos tests de manière explicite pour indiquer quelle unité ils testent et quel résultat ils attendent.

### Indépendance

Chaque test doit être indépendant des autres, sans dépendances d'ordre d'exécution ou d'état partagé.

### Code de Test de Qualité

Le code de vos tests doit être aussi propre et maintenable que votre code de production.

## La règle des 3 'A'

La règle des trois "**A**", ou "**AAA**", est une structure couramment utilisée pour écrire des tests unitaires.

Le terme "**AAA**" est un acronyme pour **Arrange, Act, Assert**, et chaque "**A**" représente une section du test :

### Arrange (Organiser/Préparer):

- Objectif : Mettre en place les conditions nécessaires pour le test.
- Actions typiques :
  - ✓ Initialiser les objets.
  - ✓ Configurer les valeurs.
- Exemple : Si vous testez une méthode de calcul, vous pouvez initialiser les valeurs ou les objets nécessaires pour le test dans cette phase.

### Act (Agir/Exécuter):

- Objectif : Exécuter l'action ou la méthode que vous voulez tester.
- Actions typiques :
  - ✓ Appeler la méthode ou la fonction à tester avec les paramètres préparés.
- Exemple : Si vous testez une méthode de calcul, vous appelez cette méthode avec les valeurs que vous avez préparées.

**Assert** (Affirmer/Vérifier):

- Objectif : Vérifier que le comportement ou le résultat est conforme aux attentes.
- Actions typiques :
  - ✓ Utiliser des méthodes d'assertion pour valider les résultats.
- Exemple : Vérifier que la méthode de calcul renvoie le résultat attendu.

# Mise en œuvre en PHP

## Dossier spécifique aux tests

Créer un dossier nommée **tests** à la racine de votre projet.

## Fichier de test par classe

Pour chaque classe que vous souhaitez tester, créer un fichier de tests nommé **classe-test.php**.

Ex : pour souhaitez tester la classe **Adherent**, le fichier de test se nommera **adherent-test.php**.

## Structure d'un fichier de test

Exemple : **adherent-test.php**

La partie **ENTETE** du fichier de test

```php
<?php

require_once __DIR__ . '/../vendor/autoload.php';
require_once __DIR__ . "/utils/couleurs.php";

echo PHP_EOL;
echo GREEN_BACKGROUND.BLACK;
echo "Tests : classe Adherent";
echo RESET;
echo PHP_EOL;
```

La partie **TEST UNITAIRE** permettant de tester une méthode de la classe

```php
echo "Test : adhésion valide quand la date d'adhésion n'est pas dépassée (- 1 an) \n";
// Arrange
$adherent1 = new Adherent("Jean", "Dupond", "22/09/2023");
// Act
$resultat = $adherent1->isAdhesionValide();
// Assertion
if ($resultat) {
    echo GREEN."Test OK".RESET.PHP_EOL;
} else {
    echo "Test pas OK" . PHP_EOL;
}
```

→ **Initialiser les objets**

→ **Appeler la méthode à tester**

→ **Vérifier que le résultat est conforme aux attentes**

```php
echo "Test : création d'un adhérent avec date adhésion non précisée 'défaut date du jour') \n";
// Arrange
$adherent1 = new Adherent("Jean", "Dupond");
// Assertion
if ($adherent1->getDateAdhesion()->format('d/m/Y') === (new \DateTime())->format('d/m/Y')) {
    echo GREEN."Test OK".RESET.PHP_EOL;
} else {
    echo "Test pas OK" . PHP_EOL;
}
```

→ **Initialiser les objets**

→ **Vérifier que le résultat est conforme aux attentes**