# Tests unitaires - Utilisation de PHP Unit

## Installation

1. Copier les 2 fichiers **phpunit.phar** et **phpunit.cmd** fournis dans le répertoire PHP associé à Laragon ( C:\laragon\bin\php\php-7.1.14-Win32-VC14-x64 )
2. Ajouter le répertoire C:\laragon\bin\php\php-7.1.14-Win32-VC14-x64 au PATH de Windows (si ça n’est pas déjà fait)
3. Vérification :
   1. créez le répertoire projet du jour dans c:\Laragon\www , par exemple 20181119
   2. Exécutez la commande phpunit --version

Vous devez obtenir :



## Application sur un mini exemple

Récupérez le programme 005\_ArrayTest.php (issu du manuel PHPUnit).

Copiez le dans le répertoire du jour

Exécutez la commande suivante, en ligne de commande :

phpunit 005\_ArrayTest.php

* 1. Quel est le résultat ?
  2. Comment procède PHPUnit pour les tests unitaires ?

## Premier test unitaire

1. Ecriture d’une méthode de calcul

Tout d’abord, créez une classe PHP dans un fichier 010\_Calcul.php.

Dans ce fichier, créez une classe Calcul.

Dans cette classe, créez une méthode nommée carre, qui doit être une méthode de classe.

Cette méthode retourne simplement le carré du nombre passé en paramètre.

1. Test unitaire de la méthode de calcul
   1. Créez un programme php 010\_TestCarre.php
   2. Inspirez vous du programme fourni 005\_ArrayTest pour toute l’enveloppe (use, class, function) en changeant le nom de la classe et de la fonction
   3. Cherchez le nom d’une méthode PHPUnit permettant de tester l’égalité entre 2 valeurs : <https://phpunit.readthedocs.io/fr/latest/assertions.html>
   4. Testez alors si le carré du nombre 4 est égal à 16
   5. Enregistrez et tester le programme

Astuce : pour la partie B.d, cherchez comment appelez une méthode de classe directement (sans passer donc par la création et l’instanciation d’un objet : c’est ce qu’on fait dans Laravel quand on déclare une route).

1. Appel du test unitaire

Appelez le test unitaire ainsi crée.

Que donne le résultat ?

* Modifiez votre comparaison pour qu’elle produise un résultat faux et tester.

## Etude de cas : fin WebCaisse

Le module de gestion des ventes de la *WebCaisse* gère le programme de fidélisation pour chaque point de vente :

* Pour le programme de type 1 (tampon), le module de gestion des ventes permet de compter le nombre de visites d’un consommateur. À chaque achat, le consommateur gagne un tampon. Il peut bénéficier d’une réduction à partir d'un nombre de tampons paramétré dans *WebCaisse* pour le point de vente.

* Pour le programme de type 2 (montant achat), le module de gestion des ventes totalise le total dépensé par chaque consommateur. Lorsque le montant total des achats atteint un seuil paramétré pour le point de vente dans *WebCaisse*, le consommateur peut bénéficier d’une réduction.
* Pour le programme de type 3 (points), le module de gestion des ventes attribue un nombre de points fidélité lors de chaque achat du consommateur en fonction de la tranche d’achat :
  + 10 points (montant d'achat compris entre 100 € et 200 € inclus) ;
  + 20 points (montant d'achat supérieur à 200 € jusqu’à 500 €) ;
  + 50 points (montant d'achat supérieur à 500 €).

Lorsque le nombre de points atteint un seuil paramétré pour le point de vente dans *WebCaisse*, le consommateur peut bénéficier d’une réduction.

La réduction associée au programme de fidélisation est également paramétrée dans *WebCaisse* pour chaque point de vente.

La méthode *addFidelite()* de la classe ConsoFidele, fournie dans le dossier documentaire, est appelée à l’issue de chaque achat d’un consommateur inscrit au programme de fidélisation. Elle permet de mettre à jour l'attribut *pointsFidelite* de la classe ConsoFidele en fonction du programme de fidélisation choisi pour le point de vente.

La classe de test TestPointsFidelite**,** fournie dans le dossier documentaire, a été créée afin de valider la méthode *addFidelite.*

Certains tests proposés dans cette classe ne sont pas terminés. Le test *testInitConso()* ne traite pas le cas de la valeur initiale des points de fidélité, qui devrait être fixée à zéro lors de la création d'un consommateur fidèle.

**Question 4.1**

🡪 Compléter la méthode *testInitConso* du programme de test 020\_TestsUnitairesFidelite.php permettant de combler ce manque.

Votre chef de projet vous demande de compléter le test qui traite le cas du programme de fidélisation par points (type 3).

**Question 4.2**

🡪 Compléter la méthode *testAddMontant*du programme de test 020\_TestsUnitairesFidelite.phppermettant de valider les points de fidélité obtenus dans le cas d'un de programme de fidélisation par points.

## Pokemons: test unitaire – approche naive

Nous allons commencer avec une approche « naïve » du test d’API. Nous avons développé une classe **PokemonApi** qui récupère toutes les caractéristiques et statistiques d’un Pokemon donné.

Nous souhaitons développer des tests unitaires pour nous assurer qu’à chaque fois que nous faisons *évoluer* notre application, que nous lui ajoutons des fonctionnalités, il n’y a pas de *régression*, que ce qui fonctionnait fonctionne toujours.

1. 1er test : nombre de capacités

Analysez les programmes « 035\_Pokemon.php » et « 035\_TestPokemon.php ».

* Expliquez ce qu’ils font.

1. Ajout d’une méthode de comptage de statistiques sur pokemon

* Ecrivez une méthode permettant de compter le ***nombre de statistiques*** dans la réponse JSON (entrée stats) de l’api

1. Ajout d’une méthode test sur les statistiques de pokemon

* Ecrivez un test unitaire permettant de tester que le nombre de statistiques renvoyé par la réponse JSON est égal à 6.

## Pokemons: test unitaire – approche moderne et complète

Le problème avec notre approche naïve est que nous ne maîtrisons pas l’API Pokemon. D’ailleurs elle change souvent d’adresse, est parfois cassée.

Nous savons juste que la structure de l’API n’évolue pas, ou alors, quand elle évolue, elle assure la compatibilité.

Nous voulons cependant développer notre application autour de Pokemon et pouvoir la tester, dans tous ses aspects.

Comment faire alors que nous ne maîtrisons pas la disponibilité de l’API sous-jacente ?

La réponse : **utiliser des doubles**.

1. Modification du programme

La nécessité de pouvoir tester ses programmes dans la durée oblige un refactoring conséquent de notre classe d’origine.

* Comparez les programmes ***035\_Pokemon.php*** et ***040\_PokemonWrapper.php*** et listez les différences.

1. Utilisation de doubles

Afin de pouvoir s’affranchir de la disponibilité de l’API Pokemon, nous allons utiliser ce qu’on appelle dans le monde des tests des ***doubles***.

* En vous inspirant du test réalisé, ajoutez un nouveau test pour s’assurer de la bonne implémentation de la méthode de calcul du nombre de statistiques.