PHP

Table des matières

Auteur : M. Mithridate / G. Rodrigues
Relu, validé & visé par :
☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création : 25/08/2023
Date révision :

PHP

PHP

PHP

Emplacement table des matières suite.

PHP

1 Compétences pédagogiques :

Être capable de comprendre le fonctionnement des variables

Être capable de manipuler les opérateurs

Être capable d'utiliser les instructions conditionnelles

Être capable de manipuler un tableau

Être capable de comprendre les boucles

Être capable de créer et d'utiliser des fonctions

Être capable de comprendre le fonctionnement et l'intérêt de la programmation orienté objet

Être capable de créer et utiliser les classes

Être capable de créer et utiliser des objets

Être capable de comprendre les notions d'héritage

Être capable de comprendre les notions de polymorphisme

Être capable de créer des pages Web Dynamique

Être capable de mettre en place un système d'API

Être capable de connecter une application serveur à une base de données côté Back-end

Être capable de gérer des requêtes HTTP d'interaction côté Back-end

PHP

2 PHP c'est quoi ?

PHP (**PHP Hypertext Preprocessor**) est un langage de script libre conçu pour le développement d'applications web dynamiques.

Il s'intègre facilement dans du contenu html.

PHP est multiplateforme (Windows, Linux, Mac Os...).

Pour fonctionner PHP a besoin d'être installé sur un serveur web (Apache, NGINX, IIS sont les plus connus).

PHP est un langage qui s'exécute côté serveur et permet la génération de page web dynamique.

L'interpréteur PHP va alors générer une page web html.

De par sa polyvalence, PHP est également un langage de programmation à part entière qui peut être utilisé indépendamment d'un serveur web par exemple pour faire du scripting ou des applications graphiques (par exemple, via la librairie PHP-GTK : http://gtk.php.net/)

https://www.php.net/manual/fr/intro-whatcando.php

3 Spécificités du langage PHP

PHP est un langage « interprété »

On peut distinguer 2 grands types de langages de programmation : les langages interprétés et les langages compilés.

- **Langages interprétés** :
  - L'interpréteur lit le code et exécute les commandes ligne par ligne.
  - Le principal avantage de ces langages est leur côté multiplateforme : dès lors qu'un interpréteur existe sur la plateforme, un même code pourra y être exécuté.
  - Toutefois, cela a un coût en matière de ressources, puisqu'à chaque exécution du programme, l'interpréteur devra traduire les commandes avant qu'elles puissent être exécutées par la machine.
  - Exemples : PHP, Java, Python, JavaScript
- **Langages compilés** :
  - Les instructions sont compilées et traduites en langage machine avant de pouvoir exécuter le programme.
  - Le programme devra donc être compilé sur chaque machine sur lequel on souhaite l'exécuter.
  - Le principal avantage est l'optimisation des ressources nécessaires pour exécuter le programme, que ça soit au niveau de la mémoire nécessaire que de la vitesse d'exécution.
  - Exemples : C, C++, Pascal

Afin d'améliorer les performances des langages interprétés, la plupart de ces langages proposent en réalité une « pré-compilation ».

- **Java** utilise une **machine virtuelle**. Le compilateur Java va en réalité compiler le code en **byte code** et la machine virtuelle sera ensuite chargée d'interpréter ce byte code en langage machine.
- L'interpréteur **Python** propose (sans l'imposer) une option de précompilation des scripts en byte code.
- C'est un peu différent pour **PHP**. Le script est compilé à la volée lors de chaque appel serveur et est ensuite exécuté. Afin d'améliorer les performances et limiter les ressources, des **PHP cache code** ont été développés, permettant de mettre en cache sur le serveur le code compilé.

PHP

PHP est un langage « à typage faible » ou « non-typé »

Une autre grande différence des langages de programmation est le typage des variables, celui-ci pouvant être « fort » ou « faible ».

Un langage à typage fort vérifiera que le type de variable attendu est bien celui reçu, et renverra une erreur dans le cas contraire. De plus, les opérations sur des variables de types différents seront interdites.

A contrario, un langage faiblement typé sera beaucoup plus permissif, ce qui peut parfois entrainer des comportements surprenants.

- Exemples de langages à typage fort : C++, Java, Python
- Exemples de langages à typage faible : PHP, JavaScript, C

PHP

4 Evolution du PHP

PHP a été créé en 1994 par Rasmus Lerdorf pour la création de son site web personnel.

Son travail a été repris en 1997 par deux étudiants : Andi Gutmans et Zeev Suraski qui ont alors publié la version 3 de PHP.

Ils ont ensuite réécrit le moteur interne de PHP, aboutissant au moteur Zend Engine qui a servi de base à la version 4 de PHP. Cette version 4 est également celle qui a commencé à intégrer la programmation orientée objet au langage PHP.

La version 5 a été la version intégrant un modèle de programmation orientée objet complet.

A la suite de ça, de nombreuses évolutions et réécritures du moteur ont permis à PHP de gagner en performance et robustesse en s'inspirant des langages comme Java : support des tests unitaires, typage des méthodes et propriétés, standards de développement (PSR).

Voici une frise illustrant son évolution au fil des années : https://www.jetbrains.com/fr-fr/lp/php-25/

PHP

5 Environnement de travail :

Pour développer en PHP nous allons avoir besoin :

D'un serveur, WAMP (Windows) ou LAMP(Linux) suivant notre environnement de travail.

- Linux / Windows : Le système d'exploitation hébergeant le serveur
- Apache (serveur web pour héberger nos différents fichiers),
- MySQL (serveur de base de données, pour héberger nos bdd),
- PHP (interpréteur PHP),

Pour concevoir nos différents fichiers :

- Un éditeur de code (Visual studio code, PhpStorm, Notepad++, Bracket, Sublime Text, etc…),

Pour tester notre code :

- Un navigateur web pour afficher nos pages tester et contrôler le rendu. (Chrome, Mozilla Firefox, Edge, Safari etc…).

PHP

6 Syntaxe du langage :

Pour intégrer du code PHP nous écrivons nos scripts à l'intérieur de fichier avec l'extension **php**.

Emplacement des fichiers, Exemple :

Dans le dossier **C:\wamp64\www\exemple*index.php*** (exemple du chemin avec WAMP) du serveur apache (WAMP, XAMPP, Docker, LAMP, etc…)) nous allons créer un fichier **index.php**.

Par défaut, le serveur web renverra le contenu du fichier index.php (ou index.html) lorsque l'on visitera le site sans préciser de fichier en particulier.

Balises :

Lorsque PHP traite un fichier, il cherche les balises d'ouverture et de fermeture lui indiquant le code qu'il devra interpréter.

Nos scripts PHP devront être rédigés entre les balises :

- **< ?php** …code PHP… **?>** **:** ce sont les balises normales.
- **< ?** …code PHP… **?>** **:** ce sont les balises courtes. Toutefois ces dernières pouvant être désactivées, il est conseillé de n'utiliser que les balises normales.
- **< ?=** « du texte » **?>** : ces balises courtes permettent d'afficher du texte. Elles sont l'équivalent de
  < ?php echo « du texte » **?>**

PHP

Exemple :

La page sera accessible dans le navigateur web à l'adresse suivante :

localhost/exemple/index.php

NB : le fichier doit être exécuté et se trouver sur le serveur, si on ouvre simplement le fichier celui ne retournera rien.

*Depuis l'exemple précédent nous devons avoir le fichier à l'intérieur du répertoire WWW de Wamp ou HTDOCS de Xamp et créer un sous dossier (dans le dossier à la racine de **www** ou **htdocs** en fonction du logiciel) exemple, enfin créer un fichier index.php dans celui-ci. On saisira dans le navigateur web l'adresse suivante (url) **localhost/exemple/index.php**, pour exécuter le fichier. L'interpréteur PHP du serveur va alors lire le fichier **.php** et exécuter le code contenu dans celui-ci.*

Exemple de code d'une page :

```
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>ma première page php</title>
</head>
<body>
    <h1>mon premier programme</h1>
    <?php
       //le script php se trouvera entre ces balises
    ?>
</body>
</html>
```

PHP

Syntaxe générale :

- Les instructions sont séparées par des « ; ». Bien que ça ne soit pas obligatoire, il est conseillé d'effectuer un retour à la ligne après chaque instruction.

```php
<?php
    //script php;
?>
```

- Les blocs de code (classes, boucles, fonctions) sont délimités par des accolades : « { », « } ».

```php
<?php
  Class MaClasse
  {
        function maFonction()
        {
            // Code de la fonction
        }
  }
?>
```

- Indentation : Bien que non obligatoire, on indente nos blocs de code pour une meilleure lisibilité (4 espaces).

PHP

Commenter des lignes de codes :

```php
<?php
    //commentaire sur une ligne

    /*
    ------------------------------------
    Commentaire sur plusieurs lignes
    ------------------------------------
    */

    /**
    * ------------------------------------
    * PHPDoc
    * ------------------------------------
    */
?>
```

PHP

Création de notre premier programme en php :

Nous allons créer un programme qui va afficher dans le navigateur internet.

Hello World

-Créer une page index.php dans votre éditeur de code et déposer là à l'intérieur de votre dossier **www/cours** du serveur apache (ou **htdocs/cours** si vous utilisez **xampp**).

-A l'intérieur de la page saisir le code ci-dessous :

```
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Ma première page PHP</title>
</head>
<body>
    <h1>mon premier programme</h1>
    <?php
    //programme Hello Word
    //La commande echo permet d'afficher du contenu dans une page
html.
    echo "Hello World";
    ?>
</body>
</html>
```

Pourquoi Hello World ?:

https://deux.io/pourquoi-hello-world/

PHP

7 Les variables :

Une variable ça sert à quoi ?

Les variables permettent de stocker des valeurs (Saisies, Résultat d'un sous-programme)

Elles vont pouvoir contenir des valeurs de types différents (texte, numérique…)

Une variable est une sorte de boite étiquetée avec un contenu.

Pour avoir accès à son contenu nous utiliserons son étiquette (son nom).

Les types de variables :

Bien que le typage en PHP soit un typage faible et dynamique (c'est-à-dire que le type est défini lorsque l'on donne une valeur à la variable), une variable en PHP possède un type de donnée.

- Le type « chaine de caractères » ou String en anglais
- Le type « nombre entier » ou Integer en anglais,
- Le type « nombre décimal » ou Float en anglais,
- Le type « booléen » ou Boolean en anglais,
- Le type « tableau » ou Array en anglais,
- Le type « objet » ou Object en anglais,
- Le type « NULL » qui se dit également NULL en anglais.
- Le type « ressource » ou Resource en anglais faisant référence à une ressource externe (ex : stream FTP, PDF document, …)

PHP

Nommer une variable :

En PHP, les variables sont représentées par un signe « $ » suivi du nom de la variable.

De plus, le nom d'une variable en PHP doit répondre à certaines contraintes.

- Le nom d'une variable doit impérativement commencer par une lettre ou le symbole underscore « _ »
- Le nom d'une variable ne peut être composé que de lettres, chiffres et de symboles underscore.
- La casse des caractères est prise en compte. Ainsi, les variables $variable et $varIABLE sont différentes.
- Par convention, une variable commencera par une lettre en minuscule, et sera écrite en camelCase. De plus, on essayera toujours de donner un nom compréhensible à la variable pour faciliter la compréhension du code.

Déclaration d'une variable :

En PHP une variable s'écrira comme ci-dessous :

```
$nomVariable = valeur;
```

Le symbole dollars **$** désignera une variable au moment de sa création et quand on l'utilisera.

Le type de la variable est défini lorsque l'on lui attribue une valeur.

Exemple d'utilisation d'une variable :

```php
<?php
    $variable = 10;
    $total = $variable + 10; //total vaut 20 (10 de variable + 10 en numérique).
?>
```

PHP

Exemple :

```php
<?php
//Définition d'une variable de type int ayant pour valeur 0
$varInt = 0;

// Définition une variable de type float (nombre à virgule flottante)
$varFloat = 3.4;

// Définition d'une variable de type bool (booléen)
$varBool1 = true;
$varBool2 = false;

/*
Définition d'une variable de type string
On peut utiliser le symbole simple quote ('') ou double quote ("") pour
définir une chaine de caractères
*/
$varString1 = 'Une chaine de caractères';
$varString2 = "Une autre chaine de caractères";

// Définition d'une variable de type array
$varArray1 = array();
$varArray2 = [];
?>
```

Afficher le contenu d'une variable :

Pour afficher le **contenu** d'une variable nous utiliseront le code ci-dessous :

```php
<?php
//initialisation d'une variable
$nbr =2 ;

//la fonction php echo permet d'afficher le contenu de la variable nbr
echo $nbr ;
?>
```

PHP

Afficher le type d'une variable :

Pour afficher le **type** d'une variable nous utiliserons la fonction « gettype($maVariable) :

```php
<?php
//initialisation d'une variable
$nbr =2;

//affichage dans la page web avec la fonction echo
echo $nbr ;

//utilisation de la fonction gettype pour afficher le type de la variable
echo gettype($nbr); //Ceci affichera "integer"

// Debugger une variable.
var_dump($nbr); // Affichera "int(2)"
?>
```

Exercice variables :

Exercice 1 :

-Créer une variable de type int avec pour valeur 5,

-Afficher le contenu de la variable (utilisation de la fonction php **echo**),

-Afficher son type (utilisation de la fonction php **gettype**),

-Créer une variable de type String avec pour valeur votre prénom,

-Afficher le contenu de la variable (utilisation de la fonction php **echo**),

-Créer une variable de type booléen avec pour valeur false,

-Afficher son type (utilisation de la fonction php **gettype**).

PHP

8 Les constantes :

Une constante est un identifiant permettant de stocker une donnée simple.

Une fois définie, cette donnée ne peut plus être modifiée.

Par convention, le nom d'une constante est toujours écrit en majuscules.

Il existe deux façons de déclarer une constante :

```php
<? php
define('MA_CONSTANTE_1', 'Hello');
const MA_CONSTANTE_2 = 'World!';

echo MA_CONSTANTE_1 . ' ' . MA_CONSTANTE_2;
//affichera "Hello World!"
?>
```

9 Les opérateurs :

Les opérateurs arithmétiques :

Pour effectuer des opérations mathématiques sur des types numériques (int, long, float, etc…)

On utilise les opérateurs mathématiques suivants :

Addition :

$a + $b

Soustraction :

$a - $b

Multiplication :

$a * $b

Division :

$a / $b

Modulo :

$a % $b (reste de la division de **$a** divisé par **$b**)

Exponentielle :

$a ** $b (Résultat de l'élévation de **$a** à la puissance **$b**)

PHP

Exercices Opérateurs arithmétiques :

Exercice 1 :

-Créer 2 variables $a et $b qui ont pour valeur 12 et 10,

-Stocker le résultat de l'addition de $a et $b dans une variable $total,

-Afficher le résultat (utilisez la fonction **echo**)

Exercice 2 :

-Créer 3 variables $a, $b et $c qui ont pour valeur $a =5, $b =3 et $c = $a+$b,

-Afficher la valeur de chaque variable (utilisez la fonction **echo**).,

-passer la valeur de $a à 2,

-Afficher la valeur de $a,

-passer la valeur de $c à $b - $a,

-Afficher la valeur de chaque variable (utilisez la fonction **echo**).

Exercice 3 :

-Créer 2 variables $a et $b qui ont pour valeur 15 et 23,

-Afficher la valeur de chaque variable (utilisez la fonction **echo**).,

-Intervertissez les valeurs de $a et $b,

-Afficher la valeur de $a et $b (utilisez la fonction **echo**).

Exercice 4 :

-Ecrire un programme qui prend le prix HT d'un article (de type float), le nombre d'articles (de type integer) et le taux de TVA (de type float), et qui fournit le prix total TTC (de type float) correspondant.

-Afficher le prix HT, le nbr d'articles et le taux de TVA (utilisez la fonction **echo**),

-Afficher le résultat (utilisez la fonction **echo**).

PHP

Les opérateurs d'incrémentation :

Pré-incrémentation :

++$a *(incrémente $a de 1 ($a = $a + 1) puis retourne la valeur de $a)*

Post-incrémentation :

$a++ *(retourne la valeur de $a puis incrémente de 1 ($a = $a+1))*

Pré-décrémentation :

--$a *(décrémente $a de 1 ($a = $a – 1) puis retourne la valeur de $a)*

Post-décrémentation :

$a-- *(retourne la valeur de $a puis décrémente de 1 ($a = $a – 1))*

PHP

Priorité des opérateurs :

La priorité des opérateurs spécifie l'ordre dans lequel les valeurs doivent être analysées. Par exemple, dans l'expression 1 + 5 * 3, le résultat est 16 et non 18, car la multiplication ("*") a une priorité supérieure par rapport à l'addition ("+"). Des parenthèses peuvent être utilisées pour forcer la priorité, si nécessaire. Par exemple : (1 + 5) * 3 donnera 18.

Lorsque les opérateurs ont une précédence équivalente, c'est leur associativité qui détermine s'ils sont évalués de droite à gauche ou inversement. Voyez les exemples ci-après.

Le tableau qui suit liste les opérateurs par ordre de précédence, avec la précédence la plus élevée en haut. Les opérateurs sur la même ligne ont une précédence équivalente (donc l'associativité décide de l'ordre de leur évaluation).

Associativité	Opérateurs	Information additionnelle
non-associative	clone new	clone et new
gauche	[	array()
droite	++ -- ~ (int) (float) (string) (array) (object) (bool) @	types et incrément/décrément
non-associatif	instanceof	types
droite	!	logique
gauche	* / %	arithmétique
gauche	+ - .	arithmétique et chaîne de caractères
gauche	<< >>	bitwise
non-associatif	< <= > >= <>	comparaison
non-associatif	== != === !==	comparaison
gauche	&	bitwise et références
gauche	^	bitwise
gauche	|	bitwise
gauche	&&	logique
gauche	||	logique
gauche	? :	ternaire
droite	= += -= *= /= .= %= &= |= ^= <<= >>= =>	affectation
gauche	and	logique
gauche	xor	logique
gauche	or	logique
gauche	,	plusieurs utilisations

PHP

10 Concaténation :

En PHP nous pouvons concaténer des valeurs entres elles. C'est à dire ajouter des chaines de caractères, des nombres, valeurs de variables au sein d'une même chaine de caractères.

Exemple :

Ecrire le nom d'une variable dans une page web :

```php
<?php
$nom = 'test';
//on va utiliser le symbole \devant le nom de la variable, ce caractère permet //d'annuler l'interprétation du caractère qui va suivre, dans ce cas il va afficher le nom de la variable et non son contenu.
echo 'affichage de la variable s'appelant \$test' ;
?>
```

Ecrire la valeur d'une variable dans une page web :

```php
<?php
$nom = 'test';
echo "affichage du contenu de la variable  \$nom : $nom ";
?>
```

Concaténer des chiffres, des chaines de caractères et les afficher dans une page web :

```php
<?php
$str = 'une chaine de caractères';
$strLength = strlen($str);
echo "<br>La chaine de caractères '$str' contient $strLength caractères.";
?>
```

PHP

Concaténer des variables dans des chaines de caractères :

```php
<?php
$var = 'du texte';

//version avec encadrement ""
$concat1 = "ma chaine $var";

//version avec encadrement ''
$concat2 = 'ma chaine ' . $var;
?>
```

Exercices :

Exercice 1 :

-Créer une variable $a qui a pour valeur ***« bonjour »,***

-Afficher le **nom de la variable et sa valeur**.

Exercice 2 :

-Créer 1 variable $a qui a pour valeur **« bon »**,

-Créer 1 variable $b qui a pour valeur **« jour »**,

-Créer 1 variable $c qui a pour valeur **10**,

-Concaténer **$a, $b et $c + 1**,

-Afficher le **résultat** de la concaténation.

Exercice 3 :

-Créer une variable $a qui a pour valeur **bonjour**,

-Afficher un paragraphe (**balise html**) et à l'intérieur la phrase suivante :

l'adrar

-Ajouter la variable $a avant la phase dans le paragraphe,

-Cela doit donner :

*** bonjour l'adrar ***

Les opérateurs d'affectation :

$a = 5 *(affecte la valeur 5 à la variable $a)*

$a += 3 *(affecte la valeur $a+3 à la variable $a : équivalent à $a = $a + 3)*

$b = 'Bonjour '; (affecte la valeur 'Bonjour ' à la variable $b)

$b .= 'le monde.'; (Concatène 'le monde.' Et l'affecte à la variable $b : équivalent à $b = $b . 'le monde.')

11 Les Fonctions :

Les fonctions permettent de générer du code qui va être exécuté chaque fois que l'on va appeler la fonction par son nom. Les fonctions sont créées pour exécuter une tâche précise et permettre de mutualiser le code afin d'améliorer la qualité et maintenabilité du code.

Nous avons déjà utilisé la fonction gettype() qui est une fonction interne à PHP.

Nous pouvons trouver la liste des fonctions internes sur la documentation officielle de PHP (https://www.php.net/manual/fr/funcref.php) ou sur le site w3school (https://www.w3schools.com/php/default.asp).

Une fonction peut prendre des paramètres en entrée et retourner un résultat.

Création d'une fonction :

Pour créer une fonction en php nous allons utiliser la syntaxe suivante :

```php
<?php
function nomDeLaFonction()
{
}
?>
```

Le nom d'une fonction devra répondre au mêmes critères de nommage que les variables (commencer par une lettre ou le symbole underscore et ne comporter que lettres, chiffres ou symboles underscore. Par convention le nom d'une fonction sera écrit en camelCase.

PHP

Appel d'une fonction :

Pour appeler une fonction on va saisir le nom de la fonction suivi de ***()***

Exemple :

```php
<?php
function maFonction()
{
}
maFonction();
?>
```

Création d'une fonction avec des paramètres :

Une fonction avec des paramètres va nous permettre d'exécuter le code de celle-ci et d'adapter son traitement en fonction de ce que l'on va passer en paramètre. Le mot clé **return** permet de retourner (int, string, boolean etc..).

Exemple :

```php
<? php
maFonction(10,5);

function maFonction($a,$b)
{
    $result= $a+$b ;

    return $result ;
}
?>
```

PHP

Exercices :

Exercice 1 :

-Créer une fonction qui soustrait à **$a** la variable **$b** (*2 paramètres en entrée*),

-la fonction doit retourner le résultat (**return**).

Exercice 2 :

-Créer une fonction qui prend en entrée un nombre à virgule (**float**),

- la fonction doit retourner l'arrondi (**return**) du nombre en entrée (utiliser une fonction interne au langage).

Exercice 3 :

-Créer une fonction qui prend en entrée **3 valeurs** et retourne **somme** des 3 valeurs.

Exercice 4 :

-Créer une fonction qui prend en entrée **3 valeurs** et retourne la **valeur moyenne** des 3 valeurs (saisies en paramètre).

Typage des paramètres :

Le version 7 de PHP introduit le typage des paramètres d'entrée et de sortie des fonctions.

Cela permet un meilleur contrôle des entrées, ce qui est impératif pour du code de qualité et devra donc être utilisé autant que possible. En effet, si le paramètre reçu par la fonction n'est pas du bon type, le serveur lancera une erreur.

Le type de sortie d'une fonction en retournant aucune valeur est « void ».

Si un paramètre peut être de plusieurs types, on peut utiliser le symbole pipe « | » pour indiquer plusieurs paramètres (exemple : int|float).

Dans l'exemple ci-dessous, on a défini que les paramètres $a et $b étaient de type « float », et que le paramètre de sortie était également de type « float ».

```php
<? php
maFonction(10,5);

function maFonction(float $a,float $b):float
{
    return $a+$b ;
}
?>
```

Exercices :

Exercice :

Modifier les fonctions précédemment écrites en typant les paramètres d'entrée et de sortie.

12 Les structures conditionnelles :

Les structures conditionnelles, ou conditions vont nous permettre de conditionner l'exécution de certains blocs de codes si certaines conditions sont remplies (ou si elles ne le sont pas.

Pour cela nous allons utiliser les instructions ***if*** (***si***), ***elseif*** (**sinon si**) ***else*** (***sinon***).

Instruction if :

Instruction fondamentale dans un programme, elle permet l'exécution d'un bloc de code si une condition est remplie.

```php
<?php
    if ($heure > 6 && $heure < 21) {
        echo 'Il fait jour';
    }
?>
```

Instruction else :

Permet d'exécuter un bloc de code dans le cas où la condition du « if » ne soit pas remplie.

```php
<?php
    if ($heure > 6 && $heure < 21) {
        echo 'Il fait jour.';
    } else {
        echo 'Il fait nuit.';
    }
?>
```

PHP

Instruction elseif :

elseif est une combinaison de if et else. Si la condition précédente n'était pas remplie, l'interpréteur va vérifier si celle-ci l'est, et exécuter le bloc de code correspondant si c'est le cas.

```php
<?php
    //$i est un nombre
    if ($i < 0) {
        echo '$i est inférieur à 0';
    } elseif ($i > 50) {
        echo '$i est supérieur à 50';
    } else {
        echo '$i est compris entre 0 et 50 inclus';
    }
?>
```

Opérateur ternaire :

Un operateur permet de réaliser certaines conditions if…else de manière abrégée.

Par exemple, voici deux façons de vérifier si une personne est majeure, l'une avec une structure if…else classique, l'autre avec un opérateur ternaire.

```php
<?php
function isAdult(int $age): bool
{
    if ($age >= 18) {
        return true;
    } else {
        return false;
    }
}

function isAdultV2(int $age) : bool
{
    return ($age >= 18) ? true : false;
}

function isAdult200IQ (int $age): bool
{
    return $age >= 18;
}
?>
```

PHP

Instruction switch :

L'instruction switch équivaut une succession d'instructions if. Elle permet de comparer une variable à des valeurs, et exécuter un (ou plusieurs) bloc(s) de code en conséquence.

Attention, le switch effectue une comparaison large (il ne compare pas le type de donnée. Equivalent à ==).

```php
<?php
    //$i est un nombre
    switch($i) {
        case 1:
            echo '$i vaut 1';
            break;   //l'instruction break permet de sortir de la structure de contrôle et donc de ne pas effectuer les instructions suivantes.
        case 2:
            echo '$i vaut 2';
            break;
        case 3:
            echo '$i vaut 3';
            break;
        default:     //Bloc exécuté dans tous les cas, sauf si on a eu un break avant
            echo '$i ne vaut ni 1, ni 2, ni 3';
    }
?>
```

PHP

Opérateurs de comparaison :

Nous allons utiliser dans nos conditions les opérateurs de comparaisons.

Exemple	Nom	Résultat
$a == $b	Egal	true si $a est égal à $b après le transtypage (comparaison large).
$a === $b	Identique	true si $a est égal à $b et qu'ils sont de même type.
$a != $b	Différent	true si $a est différent de $b après le transtypage (comparaison large).
$a <> $b	Différent	true si $a est différent de $b après le transtypage (comparaison large).
$a !== $b	Différent	true si $a est différent de $b ou bien s'ils ne sont pas du même type.
$a < $b	Strictement inférieur	true si $a est strictement inférieur à $b.
$a > $b	Strictement supérieur	true si $a est strictement supérieur à $b.
$a <= $b	Inférieur ou égal	true si $a est inférieur ou égal à $b.
$a >= $b	Supérieur ou égal	true si $a est supérieur ou égal à $b.
$a <=> $b	Combiné / Spaceship operator	Retourne -1 si $a est < à $b, 1 si $a est > à $b et 0 si $a == $b, après transtypage (comparaison large).

Opérateurs logiques :

Voici les différents opérateurs logiques qui peuvent être utilisés :

Exemple	Nom	Résultat
$a && $b	And (Et)	true si $a ET $b sont true.
$a and $b	And (Et)	true si $a ET $b valent true.
$a || $b	Or (Ou)	true si $a OU $b est true.
$a or $b	Or (Ou)	true si $a OU $b valent true.
$a xor $b	XOR	true si $a OU $b est true, mais pas les deux en même temps.
!$a	Not (Non)	true si $a n'est pas true.

Exemple :

```php
<?php
    $a = 6;

    if($a <= 3 && $a > 0) {
        //teste si $a est plus petit que 3 et est supérieur à 0
        echo "la valeur de la variable \$a est plus petite que 3";
    } elseif($a >= 3 && $a < 5) {
        //teste si $a est plus grand ou égal et 3 et inférieur à 5
        echo "la valeur de la variable \$a est comprise entre 3 et
5";
    } else {  //sinon
        echo "la valeur de la variable \$a est supérieur à 5";
    }
?>
```

Exercices :

Exercice 1 :

- Créer une fonction qui teste si un nombre est **positif** ou **négatif**

Exercice 2 :

- Créer une fonction qui prend en entrée **3 valeurs** et retourne le nombre le plus **grand**

Exercice 3 :

- Créer une fonction qui prend en entrée **3 valeurs** et retourne le nombre le plus **petit**

Exercice 4 :

- Créer une fonction calculePrixFinal qui prend en entrée un paramètre $prix de type float et retournera le prix final.

- Si le prix est > à 2000€, la ristourne sera de 20%

- Si le prix est > à 1000€, la ristourne sera de 10%

- Sinon, la ristourne sera de 0

PHP

Exercice 5 :

- Créer une fonction qui prend en entrée **1 année (entier)** et qui affiche « l'année x est une année bissextile » si l'année est bissextile ou « l'année x n'est pas une année bissextile » si ce n'est pas une année bissextile

- Pour rappel une année bissextile est définie de la façon suivante (https://fr.wikipedia.org/wiki/Ann%C3%A9e_bissextile) :

    - Les années sont en général bissextiles si elles sont multiples de quatre

    - elles ne sont pas bissextiles si elles sont multiples de cent à l'exception des années multiples de quatre cents qui le sont.

13 Les boucles :

Comme dans tous les langages de programmation, PHP gère les structures de boucle.

La boucle est un élément de base d'un langage de programmation.

Les boucles permettent de répéter plusieurs fois une ou plusieurs instructions tant qu'une condition est vérifiée ou bien jusqu'à ce qu'elle soit vérifiée.

En PHP, il existe 4 types de boucles différentes :

- La boucle « for »
- La boucle « while »
- La boucle « do…while »
- La boucle « foreach »

Pour écrire une boucle (**for**):

Boucle for :

La boucle « for » prend 3 paramètres en entrée : un paramètre initial, une condition d'arrêt, et une opération à effectuer en fin d'itération.

Exemple de boucle for :

Tant que $i est inférieur à 10 on répète l'opération :

```php
<?php
// for (valeur initiale ; condition ; opération)
for ($i=0; $i < 10; $i++){
    echo 'Ceci est une boucle for en PHP';
    echo '<br>';
    echo "\$i = $i";
    echo '<br>';
}
?>
```

PHP

Boucle while :

L'instruction « while » est traduite en « tant que ». Tant que la condition est remplie, la boucle continue de s'exécuter.

Exemple de boucle while :

Tant que $i est inférieur à 10 on répète l'opération :

```php
<?php
$i = 0; //variable compteur
while ($i < 10) //boucle while
{
    echo 'Ceci est une boucle while en PHP';
    echo '<br>';
    echo "\$i = $i";
    echo '<br>';
    //à chaque tour j'incrémente $i (+1)
    $i++;
}
?>
```

Boucle do…while :

L'instruction « do…while » pourrait être traduite par « faire…tant que ». Elle est dérivée de la boucle « while ». La différence est que la boucle while vérifie la condition avant que l'on rentre dans la boucle, alors que dans le cas de la boucle do…while, on entrera toujours dans le bloc « do » de la boucle, et on en sortira si la condition n'est plus remplie.

Le choix d'utiliser « while » plutôt que « do…while » dépend de la situation.

Exemple de boucle do…while :

Tant que $i est inférieur à 10 on répète l'opération :

```php
<?php
$i = 0; //variable compteur
do {
    echo 'Ceci est une boucle do...while en PHP';
    echo '<br>';
    echo "\$i = $i";
    echo '<br>';
} while (++$i < 10);
?>
```

PHP

Boucle foreach :

A la différence des autres boucles, ce type de boucle n'est pas présent dans la majorité des langages de programmation, et est pourtant d'une efficacité redoutable.

Cette boucle permet de parcourir tous les éléments d'un objet « iterable » (tableau, chaine de caractères, collection d'objets.

Exemple de boucle foreach :

```php
<?php
foreach ($tableau as $valeur)
{
    echo $valeur.'<br />';
}
?>
```

Exercices sur les boucles :

Exercice 1 :

- Choisir un nombre compris entre 0 et 999

- A l'aide d'une boucle while, effectuez des tirages aléatoires (utilisation de la fonction PHP « rand() » jusqu'à trouver le bon nombre.

- Affichez le nombre d'itérations nécessaires pour trouver le nombre

Exercice 2 :

- Choisir un nombre de lignes

- Choisir un nombre de colonnes

- A l'aide de boucles « for », obtenez le résultat suivant :

0000000000
1111111111
2222222222
3333333333
4444444444
5555555555
6666666666

Exercice 3 :

- Ecrivez des boucles qui affichent ce qui est demandé

- Le nombre de colonne à afficher dépend du n° de ligne, à la ligne i, il faut afficher i colonnes.

- Le résultat attendu est celui-ci :

```
1
22
333
4444
55555
666666
7777777
```

14 les tableaux

Les tableaux, aussi appelés **arrays** en anglais, sont des structures de données essentielles et massivement utilisées en développement PHP.

Un tableau permet de stocker plusieurs données dans une même variable. Les données stockées peuvent être de types différents.

Un tableau est une structure de type « carte ordonnée ». Ce type de structure associe une valeur à une clé et peut représenter par exemple des tableaux, des listes, des collections ou des dictionnaires.

Une clé dans un tableau est forcément unique.

Un tableau peut prendre en valeur un autre tableau, ce qui permet de créer des structures multidimensionnelles.

Par défaut, la clé est une valeur numérique directement définie par l'interpréteur PHP. On parle dans ce cas de **tableau indexé**. La structure est comparable à une liste.

Nous avons aussi la possibilité de définir nous même une clé de type chaine de caractères afin de créer un **tableau associatif**, structure qui se rapproche d'un dictionnaire.

Déclaration et instanciation d'un tableau :

Il existe différentes façons de déclarer et instancier un tableau.

Dans le cas d'un tableau indexé, la **première valeur** aura toujours pour **index 0**.

PHP

Exemple déclaration de tableaux indexé numériquement et associatif :

```php
<?php
// Déclaration d'un tableau vide
$tableau = [];
// OU
$tableau = array();

// Déclaration d'un tableau indexé (liste de valeurs)
// Ce tableau a 4 données de types différents
$tableauIndexe = ['Denis',42,199.3, array('Marc')];

// Déclaration d'un tableau associatif
$villes = [
    'Toulouse' => 31,
    'Auch' => 32,
    'Albi' => 81,
    'Montauban' => 82,
];
?>
```

Exemple ajouter une valeur à un tableau :

```php
<?php
$legumes = [];
// Ajout d'un élément a un tableau indexé numériquement il sera
ajouté à la dernière position.
$legumes[] = 'salade';

// Ajout d'un élément a un tableau indexé numériquement à une
position (2° position).
$legumes[1] = 'carotte';

$legumes[2] = 'cerise';

// Suppression d'une donnée d'un tableau (fonction unset)
unset($legumes[2]);

// Ajout de la taille de la personne dans le tableau associatif
$identite = [];
$identite['taille'] = 180;
$identite['prenom'] = 'Guillaume';
?>
```

PHP

Exemple parcourir un tableau :

```php
<?php
//création d'un tableau $prenoms
$prenoms[0] = 'Mathieu';
$prenoms[1] = 'Sophie';
$prenoms[2] = 'Florence';
//ou
$prenoms = ['Mathieu', 'Sophie', 'Florence'];

//parcours de tout le tableau
foreach ($prenoms as $key => $value) {
    echo '<br>';
    //Affiche le contenu du couple clé => valeur à chaque tour.
    echo "$key => $value";
}

//parcours de tout le tableau avec une boucle for
for ($i=0; $i < count($prenoms); $i++) {
    echo '<br>';
    echo "$i => " . $prenoms[$i];
}

?>
```

Tableaux imbriqués :

Il est une pratique courante d'imbriquer des tableaux dans d'autres tableaux. Voici un exemple de ce cas d'utilisation pour la gestion d'un annuaire.

```php
<?php
$annuaire = [
    [
        'nom' => 'Roger Rabbit',
        'tel' => '0504030201',
        'email' => 'roger@rabbit.com'
    ],
    [
        'nom' => 'Donald Duck',
        'tel' => '0102030405',
        'email' => 'donald@duck.com'
    ],
];

// Accéder au numéro de téléphone de Donald
$telDonald = $annuaire[1]['tel'];
?>
```

PHP

Gestion des paramètres d'une application :

L'utilisation du tableau en mode « dictionnaire » rend son usage très populaire pour la gestion des paramètres clé-valeur d'une application PHP.

Ce fichier contenant des données sensibles, il ne devrait jamais être stocké tel quel sur quelque repository Git que ce soit. Il faudra donc l'ajouter au fichier « .gitignore » de l'application. Nous versionnerons dans ce cas uniquement un fichier template (souvent suffixé par « -dist ») et l'utilisateur aura à sa charge de créer une copie de ce fichier pour y ajouter les données réelles.

Voici à quoi pourrait ressembler un fichier de config d'une application :

```php
<?php
$CONFIG = array(
    'dbtype' => 'mysql',
    'dbhost' => 'localhost',
    'dbname' => 'nom_de_la_db',
    'dbuser' => 'root',
    'dbpass' => 'password',
    'dataroot' => '/var/www/html',
);
?>
```

Exercices :

Exercice 1 :

- Générez un tableau de longueur 50 en injectant des valeurs aléatoires comprises entre -100 et 100

- Une fois les données injectées, affichez la taille du tableau

Exercice 2 :

- Créer une fonction qui affiche la valeur la plus grande du tableau (from scratch puis en utilisant une fonction interne à PHP).

Exercice 3 :

- Créer une fonction qui affiche la moyenne du tableau.

Exercice 4 :

- Créer une fonction qui affiche la valeur la plus petite du tableau (from scratch et en utilisant une fonction interne à PHP).

Exercices bonus :

https://gist.github.com/tomsihap/0ce95ee46a6b57d55144a67d68baed35

PHP

15 Fonctions internes à PHP :

Nous avons commencé à voir quelques fonctions internes à PHP. Toutefois, nous allons les détailler un peu plus.

Vous trouverez toutes les fonctions et des exemples d'utilisation dans la doc officielle de PHP : https://www.php.net/manual/fr/funcref.php

Fonctions relatives aux chaines de caractères :

empty()	Renvoie true si une chaine est vide, nulle, non définie, ou vaut 0, false, '0'
strlen()	Retourne la longueur d'une chaine de caractères
strrev()	Fonction « reverse » d'une chaine de caractères
str_repeat()	Répète une chaine de caract_res
substr()	Retourne une partie de la chaine de caractères
strcmp()	Compare 2 chaines de caractères
str_replace()	Remplace des caractères ou portions de la chaine
trim()	Supprime les espaces au début et à la fin de la chaine
strtolower()	Convertit la chaine en minuscules
strtoupper()	Convertit la chaine en majuscules
ucfirst()	Met en majuscule le 1er caractère d'une chaine
addslashes()	Echappe les caractères spéciaux d'une chaine avec des antislash
stripslashes()	Supprime les antislash d'une chaine
htmlentities()	Convertit tous les caractères éligibles en HTML
htmlspecialchars()	Convertit les caractères spéciaux en HTML
html_entity_decode()	Décode tous les caractères encodés en HTML
htmlspecialchars_decode()	Décode les caractères spéciaux encodés en HTML
strip_tags()	Supprime les balises HTML et PHP d'une chaine

PHP

Exemples :

```php
<?php
    // Longueur d'une chaine
$str = 'Bonjour le monde';
echo strlen($str);

// inversion d'une chaine
echo strrev($str);

// Utilisation des substrings
echo substr($str, 3, 4); // Retourne les 4 caractères à partir de
l'index 3 de la chaine $str -> jour

?>
```

Fonctions numériques :

ceil()	Arrondit un nombre à l'entier supérieur
floor()	Arrondit un nombre à l'entier inférieur
abs()	Retourne la valeur absolue d'un nombre
pow()	Retourne un nombre élevé à la puissance d'un autre nombre (équivalent $nb1**$nb2)
log()	Retourne le logarithme d'un nombre
rand()	Génère un nombre aléatoire
bindec()	Convertit un nombre de sa valeur binaire à décimale
decbin()	Convertit un nombre de sa valeur décimale à binaire
dechex()	Convertit un nombre de sa valeur décimale à hexadécimale
hexdec()	Convertit un nombre de sa valeur hexadécimale à décimale

PHP

Fonctions relatives aux tableaux :

explode()	Scinde une chaine de caractères dans un tableau
implode()	Joint les éléments d'un tableau dans une chaine de caractères
range()	Crée un tableau contenant un intervalle d'éléments
min()	Retourne la valeur la plus petite d'un tableau
max()	Retourne la valeur la plus grande d'un tableau
shuffle()	Mélange un tableau de façon aléatoire
array_slice()	Extrait une portion d'un tableau
array_shift()	« Dépile » le 1er élément d'un tableau (le retourne et le supprime du tableau)
array_pop()	« Dépile » le dernier élément d'un tableau (le retourne et le supprime du tableau)
array_unique()	Supprime les doublons d'un tableau
array_merge()	Combine 2 tableaux ou plus
in_array()	Vérifie si une valeur est présente dans un tableau
array_key_exists	Vérifie si une clé est présente dans un tableau
sort()	Trie un tableau
asort()	Trie un tableau associatif par valeur
ksort()	Trie un tableau associatif par clé

Exercice :

Exercice 1 :

- A l'aide de la fonction « range » créez un tableau contenant tous les nombres de 0 à 1000.

- Parcourez le tableau et extrayez tous les nombres premiers dans un autre tableau (un nombre premier n'est divisible que par 1 et par lui-même).

- Affichez les nombres premiers ainsi obtenus dans une liste HTML (
).

PHP

Exercice 2 :

- Reprenez l'exercice précédent

- Au lieu de copier les nombres premiers dans un autre tableau il faudra les supprimer du tableau initial

- A l'aide d'une fonction PHP, comparez ce tableau avec le tableau obtenu à la fin de l'exercice 1

PHP

15 Les variables superglobales :

Le transfert de données entre des pages web est géré en PHP par le biais de variables spéciales qui s'appellent superglobales.

Les variables superglobales sont des variables qui sont directement accessibles à n'importe quel endroit de l'application, quel que soit le contexte.

Voici la liste des variables superglobales :

$GLOBALS	Référence toutes les variables accessibles dans un contexte global
$_SERVER	Tableau contenant des variables relatives au serveur (nom du serveur, adresse IP, …)
$_GET	Tableau associatif des paramètres envoyés par la méthode HTTP GET
$_POST	Tableau associatif des paramètres envoyés par la méthode HTTP POST
$_FILES	Tableau associatif des variables de téléchargement de fichiers via http
$_COOKIE	Tableau associatif des variables passées au script par les cookies HTTP
$_SESSION	Tableau associatif des variables stockées dans la session HTTP
$_REQUEST	Tableau associatif contenant les variables $_GET, $_POST et $_COOKIE
$_ENV	Tableau associatif des variables d'environnement

Dans cette partie nous allons voir les Superglobales suivantes : $_GET et $_POST.

Chacune de ces variables contient un tableau avec le contenu des différents champs html d'un formulaire.

Les formulaires html possèdent 2 méthodes d'envoi possibles GET et POST.

La méthode GET fait passer les paramètres de formulaire dans l'url de la page.

Cette méthode peut être dangereuse car elle affiche directement les paramètres de la requête HTTP dans l'URL.

Ainsi, cette méthode devra uniquement être utilisée pour faire des requêtes destinées à l'affichage de contenu.

En revanche, elle ne devra jamais être utilisée pour effectuer des requêtes visant à enregistrer des données, ou pour des requêtes faisant transiter des données sensibles (identifiants de connexion).

PHP

La méthode POST fait passer les paramètres par le body de la requête HTTP.

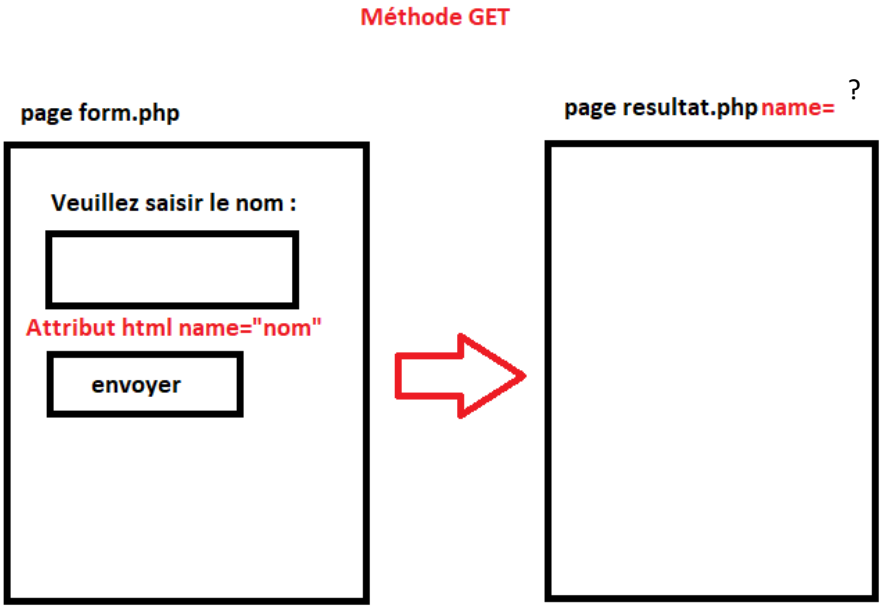
Cette méthode est à privilégier lorsque l'on fait transiter des données sensibles ou que l'on effectue des requêtes dans le but de modifier des données.

De plus elle permet de transférer des informations de taille plus importante.

Fonctionnement GET:

Le contenu des champs de formulaire va transiter dans l'url à la condition de nommer ces champs avec l'attribut html **name**.

Schéma transfert d'informations GET :



PHP

Exemple transfert de données en get :

Page form.php

```
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>formulaire</title>
</head>
<body>
    <form action="resultat.php" method="GET">
        <p>veuillez saisir votre nom :</p>
        <input type="text" name="nom">
        <br>
        <input type="submit" value="Envoyer">
    </form>
</body>
</html>
```

Cette page va envoyer à la page « resultat.php » le contenu du champ « nom » dans **l'url** sous la forme suivante :

http://resultat.php?nom=**valeur**.

Si l'on avait plusieurs champs dans le formulaire avec l'attribut name, ils seraient séparés du caractère **&** :

http://resultat.php?nom=valeur1&prenom=valeur2

PHP

Page resultat.php

```php
<?php
    //test de l'existence de la super globale $_GET
    if(isset($_GET['nom'])){
        $nom = $_GET['nom'];
        echo "Mon nom est : '.$nom.'";
    }
?>
```

Dans cette page nous allons afficher le contenu de la super globale **$_GET['nom']** avec la fonction **echo**.

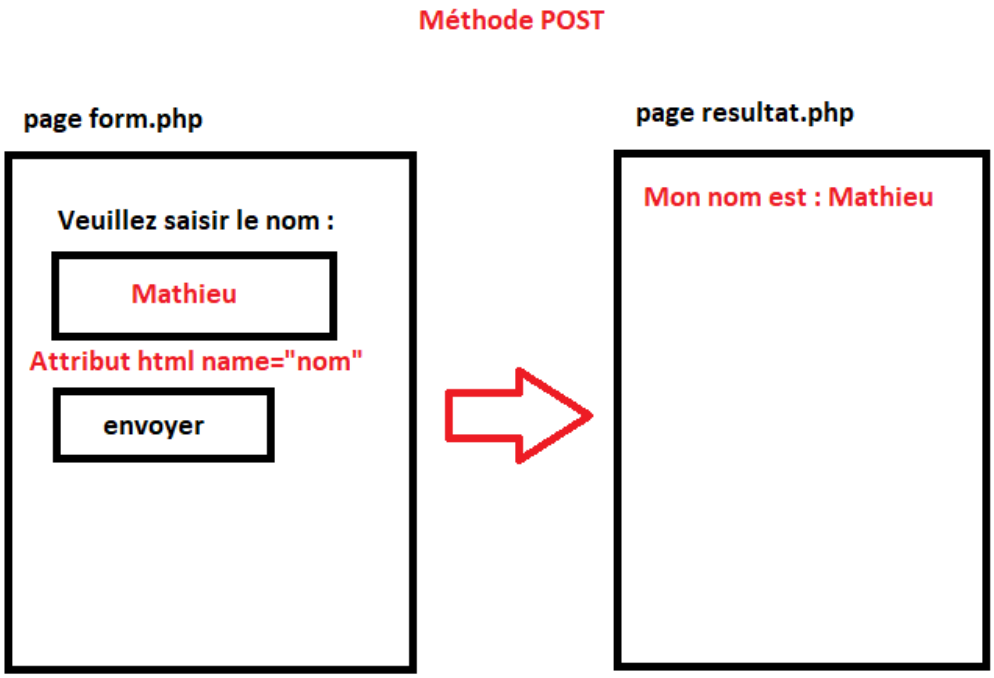
1. On vérifie l'existence de la super globale ***$_GET['nom']*** avec la fonction PHP ***isset()*** qui teste si la variable ***existe*** et si sa **valeur** n'est pas égale à **null**.
2. Ensuite on va afficher le contenu avec la méthode **echo** que l'on à vue précédemment et on concatène le résultat avec la chaine ***mon nom est :*** .

PHP

Fonctionnement POST:

Le contenu des champs de formulaire va transiter par le body de la requête HTTP à la condition de nommer ces champs avec l'attribut html **name**.

Schéma transfert d'informations POST :



PHP

Exemple transfert de données en post :

Page form.php

```
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>formulaire</title>
</head>
<body>
    <form action="resultat.php" method="post">
        <p>veuillez saisir votre nom :</p>
        <input type="text" name="nom">
        <br>
        <input type="submit" value="Envoyer">
    </form>
</body>
</html>
```

Cette page va envoyer à la page resultat.php le contenu du champ nom dans le **body**.

Page resultat.php

```
<?php
    //test de l'existence de la super globale $_POST
    if(isset($_POST['nom'])){
        $nom = $_POST['nom'];
        echo "mon nom est : .'$nom.'";
    }
?>
```

Dans cette page nous allons afficher le contenu de la super globale **$_POST['nom']** avec la fonction **echo**.

1. On vérifie l'existence de la super globale ***$_POST['nom']*** avec la fonction PHP ***isset()*** qui teste si la variable ***existe*** et si sa **valeur** n'est pas égal à **null**.
2. Ensuite on va afficher le contenu avec la méthode **echo** que l'on à vue précédemment et on concatène le résultat avec la chaine ***mon nom est :*** .

NB :

Si l'on souhaite traiter les données dans le même script PHP que la page de formulaire, dans la partie action (html) on laisse soit le champ **vide,** ou on saisit **#**, ou on réécrit le nom du script php (form.php dans l'exemple ci-dessus).

PHP

Exercices :

Exercice 1 :

- Créer une page de formulaire dans laquelle on aura 2 champs de formulaire de type nombre.

- Afficher dans cette même page la somme des 2 champs avec un affichage du style :

La somme est égale à : valeur

Exercice 2 :

- Créer une page de formulaire dans laquelle on aura 3 champs de formulaire de type nombre :

- 1 champ de formulaire qui demande un prix HT d'un article,
- 1 champ de formulaire qui demande le nombre d'article,
- 1 champ de formulaire qui demande le taux de TVA,

- Afficher dans cette même page le prix TTC (prix HT*taux TVA*quantité) avec un affichage du style :

Le prix TTC est égal à : valeur €.

PHP

16 Importer des fichiers :

Nous avons la possibilité en PHP d'importer des fichiers à l'intérieur du répertoire du projet

Pour ce faire, il suffit d'utiliser un champ de formulaire de type « file ».

Les informations relatives au fichier uploadé seront ensuite récupérées dans la superglobale $_FILES.

Elle va s'utiliser comme les super globales précédentes $_GET et $_POST.

Quand on importe un fichier, celui-ci va se retrouver dans un dossier temporaire (à la racine du serveur apache, dans le dossier /tmp) à moins qu'un autre dossier soit fourni avec la directive upload_tmp_dir du php.ini.

Le serveur va lui donner un nom temporaire (ex : tmp_1569565322.jpg).

Pour importer un fichier nous allons créer un formulaire HTML comme ci-dessous (***index.php***):

```
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Importer un fichier </title>
</head>
<body>
    <form action="index.php" method="POST" enctype="multipart/form-
data">
        <h2>importer une image</h2>
        <input type="hidden" name="MAX_FILE_SIZE" value="30000" />
        <input type="file" name="file">
        <p><button type="submit">importer</button></p>
    </form>
</html>
```

Le formulaire sera soumis avec la méthode POST et le dossier sera importé dans le dossier /tmp à la racine du serveur.

Nous allons voir ci-dessous comment le traiter et le déplacer dans le bon dossier (par ex le dossier image à la racine de notre projet) :

- Créer un nouveau répertoire **import** à la racine du serveur web (*www/import ou htdocs\import*)
- Créer un fichier import.php et coller à l'intérieur le code html de la page précédente

PHP

Nous allons ajouter le code ci-dessous dans le fichier import.php pour récupérer le fichier et le déplacer dans le bon dossier (/image à la racine du projet import).

Code importation d'un fichier avec son nom.ext dans le dossier image à la racine du projet.

Pour ce faire nous allons :

- Vérifier si le fichier que l'on importe existe (utilisation de la super globale $_FILES)
- Créer différentes variables
- Déplacer le fichier dans le bon dossier avec la méthode (***move_uploaded_file***).

PHP

```php
<?php
    /*-------------------------------------------------------
                    Test (import du fichier) :
    -------------------------------------------------------*/
    //test si le fichier importé existe
    if(isset($_FILES['file'])){
        //stocke le chemin et le nom temporaire du fichier importé
(ex /tmp/125423.pdf)
        $tmpName = $_FILES['file']['tmp_name'];
        //stocke le nom du fichier (nom du fichier et son extension
importé ex : test.jpg)
        $name = $_FILES['file']['name'];
        //stocke la taille du fichier en octets
        $size = $_FILES['file']['size'];
        //stocke les erreurs (pb d'import, pb de droits etc...)
        $error = $_FILES['file']['error'];
        //déplacer le fichier importé dans le dossier image à la
racine du projet
        $fichier = move_uploaded_file($tmpName,
"../../imports/$name");
    }


/*-------------------------------------------------------
                    Formulaire HTML :
    -------------------------------------------------------*/
?>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
</head>
<body>
    <form action="" method="POST" enctype="multipart/form-data">
        <!-- MAX_FILE_SIZE doit précéder le champ input de type file
-->
        <input type="hidden" name="MAX_FILE_SIZE" value="30000" />
        <input type="file" name="file">
        <p><button type="submit">importer</button></p>
    </form>
</html>
```

PHP

Le champ caché MAX_FILE_SIZE (mesuré en octets) doit précéder le champ input de type file et sa valeur représente la taille maximale acceptée du fichier par PHP. Cet élément de formulaire doit toujours être utilisé, car il permet d'informer l'utilisateur que le transfert désiré est trop lourd avant d'atteindre la fin du téléchargement. Toutefois, il conviendra de le vérifier côté serveur, car toutes les informations côté client peuvent facilement être modifiées par l'utilisateur.

Envoi de fichiers et sécurité :

Les sites web aux fonctionnalités complexes proposent très souvent des fonctionnalités d'upload à leurs utilisateurs (photo de profil, …).

Toutefois, il conviendra d'être très vigilant car une fonctionnalité d'envoi de fichiers mal configurée peut constituer une vulnérabilité redoutable.

En effet, imaginons un attaquant qui utiliserais le formulaire d'envoi de fichier pour y déposer un fichier php. Si ce fichier était accepté et que l'attaquant réussissait à l'appeler par l'URL, le code présent dans ce fichier serait directement exécuté par le serveur web. Un attaquant mal intentionné exploiterait cette faille pour y déposer par exemple une backdoor.

C'est la **faille upload**.

Le moyen le plus simple de s'en protéger est de n'accepter qu'un nombre limité d'extensions de fichier, répondant parfaitement à notre usage, et de rejeter tout upload de fichier qui n'y répondrait pas.

Attention toutefois, car il existe, côté attaquant diverses techniques d'évasion (fichier avec plusieurs extensions, extensions exotiques, …).

Pour plus d'informations : https://book.hacktricks.xyz/pentesting-web/file-upload

PHP

17 Interaction avec une base de données :

Le langage PHP permet d'interagir de façon simple et sécurisé (dans certains cas) avec les systèmes de gestion de bases de données les plus courants (MySQL, MariaDB, OracleDB, postgreSQL,…) .

Il existe actuellement 2 principales façons « modernes » de se connecter à une base de données SQL et d'effectuer des requêtes SQL en PHP : l'utilisation de MySQLi (https://www.php.net/manual/fr/book.mysqli.php) ou l'utilisation de PDO (https://www.php.net/manual/fr/book.pdo.php).

Ces deux méthodes présentent chacune des avantages dans des cas d'utilisation spécifiques, toutefois nous étudieront exclusivement sur la méthode PDO qui est une méthode orientée objet.

Afin de pouvoir réaliser des requêtes SQL, nous devons respecter certaines étapes :

- Etablir ou récupérer une connexion à la base de données,
- Exécuter la requête SQL
- Récupérer le résultat dans une variable (pour les requêtes de type select) et le traiter

1 - Se connecter à la base de données :

La première des actions à effectuer pour interagir avec une base de données est de se connecter à celle-ci.

Pour se faire nous devrons instancier un objet PDO.

Nous utiliserons la syntaxe suivante :

//connexion à la base de données

```php
<?php
$db = new PDO('mysql:host=localhost;dbname=nom_de_la_bdd',
'root','', array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
$db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
?>
```

Cette ligne de code va stocker dans une variable $bdd un objet **PDO** (que vous verrons dans les chapitres prochains) qui va contenir les attributs suivants :

- mysql :host = ***localhost***; (base de données de type mysql dont le host (IP du serveur) est localhost : ici identique au serveur apache) et son nom dbname = ***nom_de_la_bdd'***
- le paramètre suivant est le nom de l'utilisateur qui se connectera à la DB dans le contexte de notre application, ici : ***'root'***
- le paramètre suivant est le mot de passe de l'utilisateur, dans l'exemple ci-dessus il est vide : ***'',***
- le paramètre array permet de spécifier certaines options. Ici, une option qui active le mode d'erreur avancé.

- L'appel de la méthode setAttribute ($db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC)) permet de définir que les requêtes SQL que nous exécuterons devront retourner un résultat sous la forme de tableau associatif, dont la clé sera le nom de la colonne récupérée.

2 - Exécution d'une requête SQL :

La façon la plus simple pour exécuter une requête avec PDO est d'utiliser la méthode « query » : par exemple :

```php
<?php
$rows = $db->query('SELECT * FROM book ORDER BY title');
foreach ($rows as $row) {
    var_dump($row);
};
?>
```

Toutefois, lorsque nous devons utiliser des paramètres de filtres (clause where notamment), nous pouvont procéder de 2 façons différentes :

-Les requêtes classiques qui ne permettront pas de se prémunir des injections SQL. Leur utilisation sera donc totalement prohibée lorsque nous n'aurons pas le contrôle total des paramètres (lorsqu'ils seront fournis par l'utilisateur). Il est donc préférable de ne jamais les utiliser pour éviter tout problème.

-Les requêtes préparées qui sont, elles sécurisées car PDO échappera directement les paramètres, limitant fortement les risques d'injections SQL.

PHP

Exemple de requête classique :

En premier lieu nous devons nous connecter à la base de données (en utilisant le code vu dans la partie 1 du chapitre 13) :

```php
<?php
//Connexion à la base de données
$db = new PDO('mysql:host=localhost;dbname=nom_de_la_bdd', 'root', '',
array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
$db->exec("set names utf8");
//Exécution de la requête SQL avec un try catch pour la gestion des
exceptions (messages d'erreurs)
try {
    //requête pour stocker le contenu de toute la table le contenu est
stocké dans le tableau $reponse
    $reponse = $db->query('SELECT * FROM utilisateur');
    //boucle pour parcourir et afficher le contenu de chaque ligne de la
table
    while ($donnees = $reponse->fetch()) {
        //affichage des données d'une colonne de la bdd par son non
d'attribut
        echo '<p>' . $donnees['nom_attribut'] . '</p>';
    }
} catch (Exception $e) {
    //affichage d'une exception en cas d'erreur
    die('Erreur : ' . $e->getMessage());
}

?>
```

Cette requête va stocker dans une variable **$reponse** la requête ***select***.

Dans la boucle **while** nous allons ensuite, par la méthode « fetch », récupérer chaque ligne de résultat de la requête et la stocker dans une variable **$donnees.**

Nous pouvons ensuite afficher pour chaque enregistrement le contenu d'un **attribut ($donnees['nom_attribut'])** et l'afficher avec la méthode **echo** dans un paragraphe **html** (balise **p**).

PHP

Exemple de requête préparée :

Notre requête préparée va exécuter une requête **SQL** de type **select** similaire à la requête classique ci-dessus mais dans laquelle nous allons lui passer un **paramètre** (**$nom_utilisateur**) qui contiendra un nom d'utilisateur.

```php
<?php
//Connexion à la base de données
$db = new PDO('mysql:host=localhost;dbname=nom_de_la_bdd', 'root',
'', array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
$db->exec("set names utf8");

//Préparation de la requête SQL nous stockons dans une variable $req
la requête à exécuter
$req = $db->prepare('SELECT * FROM utilisateur where nom_utilisateur
= :nom_utilisateur');
// On va "binder" le paramètre correspondant au "non_utilisateur" de
type String
$req->bindParam('nom_utilisateur', $nomUtilisateur, PDO::PARAM_STR);
//Une fois tous les paramètres bindés, on peut exécuter la requete
$req->execute();

//boucle pour parcourir et afficher le contenu de chaque ligne de la
table
while ($donnees = $reponse->fetch()) {
    //affichage des données d'une colonne du résultat de la requête
par son non d'attribut
    echo '<p>' . $donnees['nom_attribut'] . '</p>';
}
//fermeture de la connexion à la bdd
$req->closeCursor();
?>
```

Cette requête effectue le même traitement que la requête classique mais injecte un paramètre de filtre de façon sécurisé.

PHP

Exercices :

TP 1ère partie :

a) Créer une base de données **MYSQL** avec les informations suivantes :
- Nom de la bdd : « **articles** »,
- une table nommée **article** qui va posséder les champs suivants :
  i. **id_article** (clé primaire),
  ii. **nom_article** de type varchar(50),
  iii. **contenu_article** de type varchar (255),

b) Créer une page php qui va contenir un formulaire html avec comme méthode POST (balise **form**)
- A l'intérieur du formulaire rajouter les champs suivants :
  i. Un champ input avec comme attribut html **name = «nom_article »**
  ii. Un champ input avec comme attribut html **name = «contenu_article »**
  iii. Un champ input de type **submit** avec comme attribut html **value = «Ajouter»**

c) Ajouter le code php suivant:
- Créer 2 variables $name, $content
- -Importer le contenu des 2 super globales **$_POST['nom_article']**, **$_POST['contenu_article']** et tester les avec la méthode ***isset()*** dans les variables créés précédemment (**$name** et **$content**),
- Ajouter le code de **connexion** à la base de données en vous inspirant des exemples vus dans ce chapitre,
- Ajouter une **requête simple** qui va insérer le contenu des 2 champs dans un nouvel enregistrement (requête **SQL insert**),

d) Bonus :
- Utiliser une requête **SQL préparée** à la place de la requête **simple.**
- Afficher dans un paragraphe le nom et le contenu de l'article ajouté en bdd en dessous du formulaire.

PHP

TP 2ème partie :

a) Créer une page php,
b) Ajouter le script php permettant de se connecter à la base de données **articles**,
c) Ajouter le script php qui va effectuer une requête **SQL select** permettant de récupérer tous les articles,
d) Formater le résultat de la requête (dans le résultat de la boucle while) pour quelle l'affiche sous cette forme :

```
<p>numéro de  l'article : id de l'article n</p>

<br>

<p>nom de l'article : nom de l'article n</p>

<br>

<p>contenu de l'article : contenu de l'article n</p>

<br>
```

(La liste de tous les articles devra reprendre la mise en forme ci-dessus -> a l'intérieur de la boucle while).

18 PHP et JavaScript :

Interactions entre PHP et JavaScript

Comme vous avez pu le constater, le code PHP étant interprété côté serveur, il est « invisible » côté front (en utilisant l'inspecteur du navigateur par exemple).

C'est une chance car on ne souhaiterait pas que notre code serveur soit visible de tous, étant donné qu'il peut contenir des informations sensibles telles que nos informations de connexion à la base de données.

Toutefois, il est parfois nécessaire de récupérer des informations calculées côté serveur et de les traiter dans nos scripts JavaScript. Et bien, comme nous l'avons fait pour afficher des données calculées dans notre page HTML (en utilisant notamment la fonction « echo »), nous devrons ouvrir et refermer des balises php dans notre script JS lorsque nous voudrons utiliser une donnée du serveur.

Voici un exemple d'utilisation :

```
<!DOCTYPE html>
<html lang="en">
<body>
    <h1>Test d'intégration du PHP dans des scripts JavaScript

        <?php
        // Définition de variables et de calculs en php
        $a = 3;
        $b = -5;
        ?>

        <script>
            alert("Ca se passe dans la console bro");

            // On additionne, en php, les variables $a et $b
            let addition1 = <?php echo ($a + $b); ?>;

            // On récupère les variables $a et $b pour les mettres
dans les variables JS a et b
            // Puis on fait l'addition en JS cette fois
            let a = <?= $a ?>;
            let b = <?= $b ?>;
            let addition2 = a + b;

            console.log("Résultat de l'addition faite directement en
PHP : " + addition1);
            console.log("Résultat de l'addition faite en 2 temps,
après récupération des variables en JS puis calcul en JS : " +
addition2);
        </script>
</body>
</html>
```

PHP

AJAX : Asynchronous JavaScript And XML

Dans le web moderne, que ça soit pour des raisons ergonomiques ou pour optimiser les temps de chargement des pages web, nous ne voulons pas toujours recharger une page web au submit d'un formulaire.

Un exemple très courant est par exemple lorsque l'on est sur une page affichant une liste de résultats (par exemple une liste d'articles de blog, ou une liste de produits sur une boutique en ligne). Pour ce genre de pages, on a souvent un formulaire qui permet de filtrer ou trier les résultats, et on souhaite que le formulaire gérant ces filtres mette à jours uniquement le tableau ou la liste de résultats.

Pour ce faire, nous utilisons AJAX, qui permet de faire des requêtes au serveur et traiter la réponse de façon asynchrone (en arrière-plan). La réponse que l'on reçoit nous permettra de mettre à jour le DOM (Document Object Model) de la page web, en modifiant uniquement certaines parties de cette dernière.

Il est à noter que le XML, langage de balise autrefois utilisé comme format de réponse du serveur « favori » dans le cadre des requêtes Ajax est aujourd'hui souvent remplacé par le format JSON, format beaucoup moins verbeux.

Objet XMLHttpRequest

L'objet XMLHttpRequest est l'objet natif nous permettant de traiter les requêtes asynchrones en JS.

Cet objet est toujours largement utilisé mais progressivement délaissé au profit de l'API « Fetch » de plus haut-niveau et implémentant notamment les « promesses ».

Pour effectuer une requête Ajax, nous allons devoir suivre 4 étapes :

1. Créer un objet XMLHttpRequest
2. Initialiser et préparer la requête (URL, paramètres, méthode)
3. Envoyer la requête
4. Définir des gestionnaires d'événements nous permettant de traiter les réponses du serveur ou les cas d'erreur.

```
// Instanciation d'un nouvel objet XMLHttpRequest
let xhr = new XMLHttpRequest();

// Préparation de la requête (méthode, URL)
xhr.open("GET", "/url/de/mon/script.php");

// Définition du format de la réponse (ici JSON)
xhr.responseType = "json";

// Envoi de la requête
xhr.send();
```

Traiter la réponse renvoyée par le serveur

Pour traiter les réponses du serveur, nous allons utiliser le gestionnaire d'événements XMLHttpRequest :

- **Utilisation des gestionnaires d'événements de l'objet XMLHttpRequest**
  3 événements peuvent être déclenchés au cours du traitement de notre requête :
  - **L'événement « load »** qui se déclenche lorsque la requête a bien été effectuée et que la réponse a été reçue.
    Au sein du gestionnaire d'événement « load », nous pouvons tester la valeur du status code http de la réponse
    Voici quelques-uns des status codes les plus fréquents :
    - 100 Continue : Requête en cours
    - 200 OK : Requête correctement exécutée
    - 401 Unauthorized : Problème d'identification
    - 404 Not Found : l'url n'a pas été trouvée
    - 500 Internal Server Error : Erreur côté serveur.

    Pour s'assurer que tout s'est bien passé, on veut donc vérifier que le status code est bien 200.
    On peut donc ensuite traiter la réponse du serveur qui sera accessible par la propriété « response » de l'objet XMLHttpRequest
  - **L'événement « error »** qui se déclenche lorsque la requête n'a pas pu aboutir
  - **L'événement « progress »** qui se déclenche à intervalles réguliers et nous permet de savoir où en est l'exécution de la requête

Pour plus d'informations et des exemples d'utilisation :
https://developer.mozilla.org/fr/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest

PHP

Voici ce à quoi ça pourrait ressembler au niveau du code :

```
//On crée un objet XMLHttpRequest
let xhr = new XMLHttpRequest();

//On initialise notre requête avec open()
xhr.open("GET", "une/url");

//On veut une réponse au format JSON
xhr.responseType = "json";

//On envoie la requête
xhr.send();

//On créé une fonction anonyme qui sera exécutée au trigger de l'événement.
//En l'occurrence ici l'événement load
xhr.onload = function(){
    //Si le statut HTTP n'est pas 200, on a eu un probleme
    if (xhr.status != 200){
        //...On affiche le statut et le message correspondant
        alert("Erreur " + xhr.status + " : " + xhr.statusText);
    //Si le statut HTTP est 200, on affiche le nombre d'octets téléchargés
et la réponse
    }else{
        alert(xhr.response.length + " octets  téléchargés\n" +
JSON.stringify(xhr.response));
    }
};

//On créé une fonction anonyme qui sera exécutée au trigger de l'événement.
//En l'occurrence ici l'événement error
xhr.onerror = function(){
    alert("La requête a échoué");
};

//On créé une fonction anonyme qui sera exécutée au trigger de l'événement.
//En l'occurrence ici l'événement progress
//Pendant le téléchargement.
xhr.onprogress = function(event){
    //lengthComputable = booléen; true si la requête a une length
calculable
    if (event.lengthComputable){
        //loaded = contient le nombre d'octets téléchargés
        //total = contient le nombre total d'octets à télécharger
        alert(event.loaded + " octets reçus sur un total de " +
event.total);
    }
};
```

PHP

API Fetch

L'utilisation de l'API Fetch est une façon plus moderne de réaliser des requêtes AJAX.

L'API Fetch définit des objets **Request**, **Response** et **Header** (permettant de créer et manipuler des requêtes, réponses, et headers HTTP) et propose également un mixin « **Body** » permettant d'interagir avec le corps de la requête.

L'API Fetch va également mettre à notre disposition la méthode « fetch() » qui permettra d'exécuter les requêtes.

La méthode fetch() attend au minimum la ressource vers laquelle on souhaite effectuer la requête (souvent le script PHP, une image, une API externe, …) et peut prendre un objet JSON de paramètres.

La méthode fetch() renvoie une promesse (objet de type Promise) qui va se résoudre avec un objet Response. La promesse sera rompue si la requête HTTP n'a pas pu être effectuée, mais le plus souvent, y compris en cas d'erreur, la promesse sera remplie. Il suffira donc de connaitre le status de la réponse HTTP. On utilisera les attributs « ok » et « status » de l'objet Response.

Pour plus d'informations et des exemples d'utilisation :
https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

PHP

Voici un exemple d'utilisation :

```
//Création d'un objet Headers
var myHeaders = new Headers();

// On initialise l'objet JSON qui définira les propriétés de la
requête. Ici, on définit la méthode (GET), l'objet Headers, le mode
"cors" autorisant les requêtes cross-origin
var myInit = {
  method: "GET",
  headers: myHeaders,
  mode: "cors",
  cache: "default",
};

//flowers.jpg est la destination de notre requête, et myInit nos
options
fetch("flowers.jpg", myInit)
  .then(function (response) {      // Definition de notre 'promesse'
    if (response.ok) {      //If response.ok, on est sur un status
code HTTP compris entre 200 et 299
      response.blob().then(function (myBlob) {
        var objectURL = URL.createObjectURL(myBlob);
        myImage.src = objectURL;
      });
    } else {  //Sinon, un probleme est apparu. On peut
éventuellement vérifier le status code http (response.status) pour
savoir ce qu'il s'est passé
      console.log("Mauvaise réponse du réseau");
    }
  })
  .catch(function (error) {
    console.log(
      "Il y a eu un problème avec l'opération fetch : " +
error.message,
    );
  });
```

PHP

Balise HTML <template>

La balise <template> est une balise qui n'est pas affichée sur la page. Elle est utilisée pour gérer l'affichage des réponses de nos requêtes Ajax.

Un usage classique sera de mettre le template d'une ligne d'un tableau de résultat à l'intérieur de cette balise, ou un champ de formulaire dont l'affichage est dynamique (exemple : ajouter plusieurs catégories d'articles dans le cas d'une relation many to many).

Exemple :

```
<table id="producttable">
  <thead>
    <tr>
      <td>UPC_Code</td>
      <td>Product_Name</td>
    </tr>
  </thead>
  <tbody>
    <!-- existing data could optionally be included here -->
  </tbody>
</table>

<template id="productrow">
  <tr>
    <td class="record"></td>
    <td></td>
  </tr>
</template>
```

Exercices

TP 3ème partie :

a) Ajouter un champ de recherche sur la page de résultats des articles (input de type text)
b) Faire en sorte que ce champ permette de filtrer les résultats de la requête pour que les articles soient retournés si le nom ou le contenu de l'article contient la valeur recherchée. La liste des articles devra être renvoyée au format JSON (**fonction PHP « json_encode() »**
c) A l'aide d'un listener sur le champ de recherche, faire en sorte qu'à chaque fois que la valeur de l'input change, une requête Ajax soit envoyée (**event « onkeyup »**).
d) A partir du JSON des articles et à l'aide d'une balise « template », mettre à jour le DOM à la réception de la réponse.

Correction

Fichier articles_ajax.php : le « contrôleur »

```php
<?php

$conn = null;
try {
    //Création de la connexion
    $conn = new PDO("mysql:host=localhost;dbname=articles", 'root', 'rootpwd');
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// Attribut permettant de lever une erreur en cas de problème à l'utilisation de l'objet PDO (clé : PDO::ATTR_ERRMODE, valeur :PDO::ERRMODE_EXCEPTION)
    $conn->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC); // Attribut permettant de retourner le résultat de la requête SQL sous forme de tableau associatif dont la clé est le nom de la colonne dans la BDD

    // Condition ternaire
    $searchValue = !empty($_GET['search']) ? '%'.$_GET['search'].'%' : '%%';

    $stt = $conn->prepare("SELECT * FROM article WHERE nom_article LIKE :searchvalue OR contenu_article LIKE :searchvalue2");
    $stt->bindValue('searchvalue', $searchValue);
    $stt->bindValue('searchvalue2', $searchValue);
    $stt->execute();
    $articles = $stt->fetchAll();

    $response = json_encode($articles);

    echo $response;
} catch (PDOException $e) {
    echo $e->getMessage();
}
```

PHP

Fichier list_articles_ajax.php : le code HTML / JavaScript

Code HTML

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>Tous les articles</title>
    <meta name="description" content="">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.c
ss" rel="stylesheet"
        integrity="sha384-
4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGCHeKRQN+PtmoHDEXuppvnDJzQIu9"
crossorigin="anonymous">
</head>

<body>
    <input type="text" name="search" id="search" onkeyup="updateQuery()">
    <div>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th scope="col">id</th>
                    <th scope="col">nom article</th>
                    <th scope="col">contenu article</th>
                </tr>
            </thead>
            <tbody id="table_tbody">

            </tbody>
        </table>
        <template id="row_table_template">
            <tr scope="row">
                <td></td>
                <td><a></a></td>
                <td></td>
            </tr>
        </template>
    </div>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.
min.js"
        integrity="sha384-
HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmnlMuBnhbgrkm"
        crossorigin="anonymous"></script>
</body>
</html>
```

PHP

Script JS version XMLHttpRequest (à ajouter dans list_articles_ajax.php)

```
<script>
        document.addEventListener("DOMContentLoaded", () => {
        updateQuery();
        });


    function updateQuery(){
        // Récupération de la value de l'élément input
        let searchValue = document.getElementById("search").value;
    //Création de l'objet XMLHttpRequest
    let xhr = new XMLHttpRequest();
    // Création de la requête de type GET
    xhr.open("GET", "articles_ajax.php?search="+searchValue);
    xhr.responseType = "json";
    //Envoi de la requête
    xhr.send();

    // L'événement onload est déclenché lorsque la requête s'est terminée
sans erreur
    xhr.onload = function() {
        document.getElementById("table_tbody").innerHTML = ""; // Purge de
la table actuelle
    const tbody = document.querySelector("tbody");  // Récupération de
l'élément du DOM tbdody
    const template = document.querySelector("#row_table_template"); //
Récupération du template de ligne

    // Boucle permettant d'itérer sur chacune des lignes retournées par le
script PHP (au format JSON)
    for(let i=0; i<xhr.response.length; i++) {
                    const clone = template.content.cloneNode(true);
    let td = clone.querySelectorAll("td");  // Sélection des éléments td
    let link = clone.querySelectorAll("a"); // Sélection des éléments a
    td[0].textContent = xhr.response[i]['id_article'];
    td[2].textContent = xhr.response[i]['contenu_article'];
    link[0].setAttribute('href', "article.php?id_article="+
xhr.response[i]['id_article']);
    link[0].textContent = xhr.response[i]['nom_article'];

    tbody.appendChild(clone);   // Ajout de notre ligne tr a la suite des
autres
                }
            }

    // L'événement onerror est déclenché lorsque la requête a levé une
erreur
    xhr.onerror = function() {
        console.log(xhr);
            }
        }
</script>
```

PHP

Script JS version API Fetch (à ajouter dans list_articles_ajax.php)

```
<script>
    document.addEventListener("DOMContentLoaded", () => {
        updateQuery();
    });

    function updateQuery() {
        // Récupération de la value de l'élément input
        let searchValue = document.getElementById("search").value;
    //Creation d'un objet Header
    let headers = new Headers();
    let requestParams = {
        method: "GET",
    headers: headers,
    mode: "cors",
    cache: "default",
            };
    //Utilisation d'un objet URLSearchParams pour une gestion plus propre
des parametres
    let searchParams = new URLSearchParams({
        'search': searchValue,
            });
    fetch("articles_ajax.php?" + searchParams, requestParams).
    // fetch("articles_ajax.php", requestParams).
    then(function (response) {
                    if (response.ok) {
                        return response.text();
                    }
                }).then(function (articles) {
        document.getElementById("table_tbody").innerHTML = ""; // Purge de
la table actuelle
    const tbody = document.querySelector("tbody");  // Récupération de
l'élément du DOM tbdody
    const template = document.querySelector("#row_table_template"); //
Récupération du template de ligne
    JSON.parse(articles).forEach(function (article) { // loop though the
response
        console.log(article);
    const clone = template.content.cloneNode(true);
    let td = clone.querySelectorAll("td");  // Sélection des éléments td
    let link = clone.querySelectorAll("a"); // Sélection des éléments a
    td[0].textContent = article['id_article'];
    td[2].textContent = article['contenu_article'];
    link[0].setAttribute('href', "article.php?id_article=" +
article['id_article']);
    link[0].textContent = article['nom_article'];

    tbody.appendChild(clone);   // Ajout de notre ligne tr a la suite des
autres
                    });
                });
        }
</script>
```

PHP

18 Modèle MVC :

Dans les chapitres précédents nous avons au sein d'une même page inclus la vue html ainsi que le code PHP.

Afin de mieux organiser notre code, pour permettre une plus grande facilité de mise à jour, nous allons lui appliquer le modèle MVC.

Dans ce modèle ou pattern chacun de nos fichiers aura un rôle bien défini :

Modèle : dans cette partie nous allons déplacer toute les parties du code qui vont nous permettre l'accès aux données, afin de les préparer pour le contrôleur. C'est tout ce qui va concerner les requêtes **SQL**.

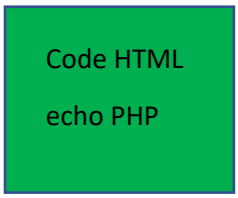
Vue : la vue se concentre sur toute la partie affichage, c'est les interfaces que l'utilisateur final va voir et avec lesquelles il va interagir dans son navigateur internet.

Dans cette partie on va retrouver toute la structure **HTML**.

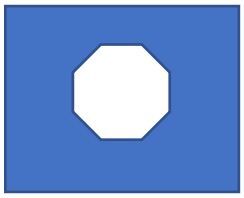
Controler : C'est le controler qui va gérer toute la logique de notre page, ainsi que les calculs et traitement des données. Le controler va demander les données au modèle et adapter la vue en fonction de celle-ci. Le controler va avoir un rôle d'aiguillage.

Dans cette partie on va retrouver exclusivement du code **PHP**.

Modèle (accès à la base de données)



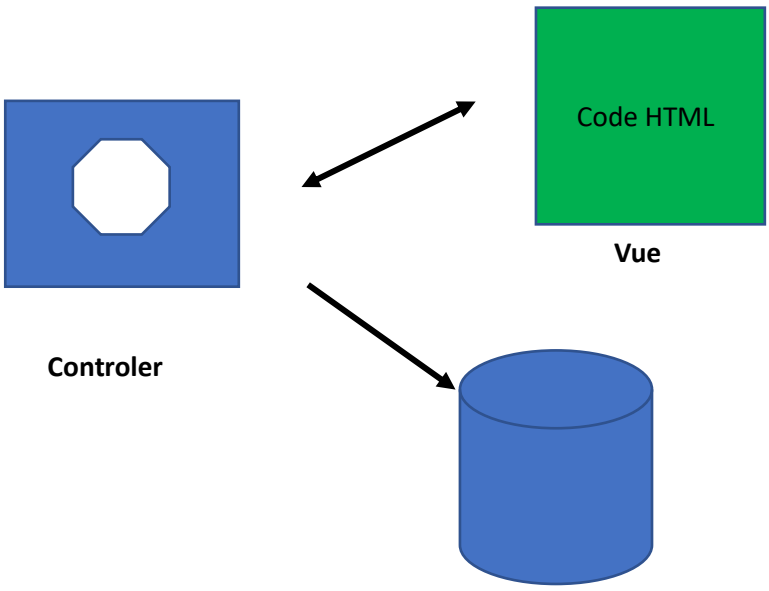
Vue (contiens le code HTML)



Controler (logique et traitement)

PHP

Echange entre les différentes couches :



Pour intégrer notre code nous allons avoir besoin d'utiliser une méthode PHP qui se nomme include().

Exemple :

Reprenons l'exercice 1 du chapitre précédent, nous allons restructurer et découper le code de cette façon :

Toute la partie html (notre formulaire) nous allons le déplacer dans un nouveau fichier que nous allons appeler **vue_article.php** comme ci-dessous :

```
<html lang="fr">
<head>
    <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ajouter un article</title>
</head>
<body>
    <form action="" method="post">
```

```
        <p>saisir le nom de l'article :</p>
        <input type="text" name="nom_article">
        <p>saisir le contenu de l'article :</p>
        <input type="text" name="contenu_article">
        <input type="submit" value="Ajouter">
    </form>
</body>
</html>
```

Toute la partie PHP (requête SQL) va être déplacer dans un nouveau fichier que nous allons nommer **model_article.php**. Comme ci-dessous :

```
try
{
                //Exécution de la requête SQL insert
        $reponse = $bdd->query('insert into article(nom_article,
    contenu_article) values("'.$name.'", "'.$content.'")');
    echo "ajout de l'article : $name qui a comme contenu : $content";
}
catch(Exception $e)
{
        //affichage d'une exception en cas d'erreur
        die('Erreur : '.$e->getMessage());
}
```

Afin de réutiliser la connexion à la base de données dans l'ensemble de notre code nous allons déplacer la connexion dans un nouveau fichier que nous allons nommer **connect.php** comme ci-dessous :

```
<?php
        //connexion à la bdd
        $bdd = new PDO('mysql:host=localhost;dbname=articles', 'root','',
        array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
?>
```

Enfin nous allons déplacer la logique (condition et test) dans une nouvelle page (qui sera notre controler) que nous allons nommer **controler_article.php**. Comme ci-dessous :

```
<?php
    //ajout de la vue
    include('vue_article.php');
    //connexion à la BDD
    include('connect.php');
    //test existence des champs nom_article et contenu article
    if(isset($_POST['nom_article']) and
isset($_POST['contenu_article']))
    {
        //création des 2 variables qui vont récupérer le contenu des super
globales POST
        $name = $_POST['nom_article'];
        $content = $_POST['contenu_article'];
        //ajout du model
        include('model_article.php.');
    }
    else{
    //affichage dans la page html de ce que l'on a enregistré en bdd
    echo '<p>veuillez remplir les champs de formulaire</p>';
    }
?>
```

Exercices :

Exercice 1 :

Reprendre l'exercice 1 de la partie précédente et l'adapter en MVC (se servir de l'exemple du cours) et remplacer la partie **model** par la requête **préparée**.

Intégrer la partie bonus (affichage de l'article ajouté dans un paragraphe).

Exercice 2 :

Reprendre l'exercice 2 de la partie précédente et l'adapter en MVC.

Exercice 3 :

Reprendre l'exercice 3 de la partie précédente et l'adapter en MVC.

PHP

19 Classe et objet :

Dans ce chapitre nous voir php sous un nouvel angle, en utilisant la programmation orientée objet.

Une classe des objets c'est quoi ?

La programmation orienté objet nous permet de modéliser dans notre code des éléments de la vie réelle :

Un véhicule, un animal, un bâtiment etc…

Une **classe** fonctionne comme une recette de cuisine qui va nous permettre de créer des **objets** :

Reprenons l'image d'une recette de cuisine c'est le plan (**classe**) qui va nous permettre de réaliser le plats, la recette a besoin d'ingrédients (que nous appellerons **attributs ou propriétés)**, d'étapes (que nous nommerons **méthodes** ou **fonctions**). A la fin de la recette nous allons obtenir un plat (que nous appellerons **objet**). Chaque fois que l'on réalisera la recette nous obtiendrons un nouveau plat qui sera unique et donc un **nouvel objet**.

En programmation orienté objet cela sera la même chose. Nous créerons une **classe** qui sera notre recette, elle contiendra des **attributs** (ou **propriétés**) que l'on peut voir comme nos ingrédients et nous aurons des **méthodes** ou **fonctions** pour effectuer les différentes étapes de réalisation du plat qui sera notre **objet**.

Créer une classe en PHP :

Pour créer une classe en PHP nous allons créer un nouveau fichier avec la syntaxe suivante :

Ex classe véhicule :

Cette classe va nous permettre de créer des véhicules, elle contiendra des attributs et des méthodes ou fonctions. Le nom d'une classe commence toujours par une majuscule :

```php
<?php
    class Vehicule{
    }
?>
```

PHP

Instancier un objet :

Pour créer un nouveau véhicule depuis la classe Vehicule nous utiliserons la syntaxe suivante :

```
< ?php
        //import du fichier class.php qui contient la classe Vehicule
        require './class.php'
        //création d'un nouveau véhicule depuis la classe Vehicule
        $voiture = new Vehicule();
?>
```

Ce code va nous permettre de créer un nouvel **objet** voiture depuis la **classe Vehicule**.

Ajouter des attributs :

Afin de personnaliser notre classe nous allons créer des attributs (variables), cela va nous permettre d'ajouter des propriétés dans nos objets.

```
< ?php
        class Vehicule
        {
        //Attributs :
        public $nomVehicule ;
        public $nbrRoue;
        public $vitesse ;
         }
?>
```

Dans l'exemple ci-dessus nous avons ajouté des **attributs** pour définir un nom à notre véhicule , nombre de roues et la vitesse de notre véhicule.

PHP

Affecter une valeur à un attribut d'un objet :

Pour affecter une valeur à un attribut d'un objet (*la valeur ne sera pas affectée à la classe mais à l'instance de notre objet*) on utilise la syntaxe suivante :

```
<?php
    //appel du fichier class.php qui contient la classe Vehicule
    //require est équivalent à include
    require './class.php';
    //création d'un nouveau véhicule depuis la classe Vehicule
    $voiture = new Vehicule();
    //ajout de valeur aux attributs de la classe Vehicule
    $voiture->nomVehicule = "Audi A3";
    $voiture->nbrRoue = 4;
    $voiture->vitesse = 250;
?>
```

Le code ci-dessus affecte des valeurs aux **attributs** de **l'objet** voiture (nomVehicule = Audi, nbrRoue = 4 et vitesse = 250)

NB : Cette syntaxe est valide uniquement quand les attributs sont en **public**. Nous Verrons plus tard qu'il est conseillé de passer les attributs en **private** ou **protected**.

PHP

Créer et appeler des méthodes :

Dans notre classe nous avons la possibilité de créer des fonctions qui serons utilisable par nos objets.

Nous allons créer dans notre classe Vehicule plusieurs méthodes que nos objets pourront utiliser.

Exemple création d'une méthode démarrer :

Dans le fichier **classe Vehicule** (vehicule.php) nous allons créer une fonction qui va démarrer le véhicule elle va afficher dans une page html un paragraphe avec comme contenu :

« Démarrage de la « nom du véhicule » Vrooom !!!!»

Pour ce faire nous allons utiliser la syntaxe ci-dessous :

```php
<?php
    class Vehicule{
        /*-------------------------------------------------------
                            Attributs :
        -------------------------------------------------------*/
        public $nomVehicule ;
        public $nbrRoue;
        public $vitesse ;
       /*-------------------------------------------------------
                            Fonctions :
        -------------------------------------------------------*/
        //fonction démarrer le véhicule
        public function demarrer(){
            echo "<p>Démarrage de la $this->nomVehicule Vrooom !!!!</p>";
        }
      }
?>
```

PHP

La variable **$this** correspond à **l'instance courante** de notre **objet**.

Pour utiliser cette méthode sur un objet nous utiliserons la syntaxe suivante :

```
<?php
    //appel du fichier class.php qui contient la classe Vehicule
    //require est équivalent à include
    require './class.php';
    //création d'un nouveau véhicule depuis la classe Vehicule
    $voiture = new Vehicule();
    //ajout de valeur aux attributs de la classe Vehicule
    $voiture->nomVehicule = "Audi A3";
    $voiture->nbrRoue = 4;
    //utilisation de la méthode démarrer
    $voiture->demarrer();
?>
```

Pour ce faire nous utilisons l'opérateur **->** puis le nom de la fonction suivi de parenthèses.

NB : Pour afficher le détail d'un objet nous utilisons la fonction php ***var_dump($objet)***.

PHP

Exercices :

Exercice 1 :

Créer un fichier **test_objet.php** qui va nous servir de fichier d'exécution,

Créer une nouvelle classe Maison **maison.php** qui va contenir les attributs suivants :

-nom, longueur, largeur.

Instancier une nouvelle maison dans le fichier **test_objet.php** avec les valeurs de votre choix (**nom**, **longueur** et **largeur**),

-Créer une méthode **surface** qui calcule et affiche la superficie de la maison (**longueur * largeur**) dans la **classe** Maison.

-Appeler la méthode **surface** et **afficher** sous la forme suivante le résultat :

*" la surface de **nomMaison** est égale à : **x** m2 ".*

Bonus

Ajouter un attribut **nbrEtage** à la classe Maison,

Modifier la méthode **surface** pour quelle prenne en compte le paramètre **nbrEtage**.

PHP

Exercice 2 :

-Créer un fichier **vehicule.php** qui va contenir la classe,

-Dans ce fichier recréer la classe Vehicule comme dans le cours (**attributs** et **méthodes**),

-Créer un fichier **test_objet.php** au même niveau que vehicule.php,

-Appeler avec **require()** ou **include()** le fichier de la classe **Vehicule**,

-Instancier 2 nouveaux véhicules dans le fichier **test_objet.php** avec les paramètres suivants :

-Objet **voiture** (nomVehicule = « Mercedes CLK », nbrRoue = 4, vitesse 250),

-Objet **moto** (nomVehicule = « Honda CBR », nbrRoue = 2, vitesse = 280),

-Créer une fonction **detect()** qui détecte si le véhicule est une moto ou une voiture (la méthode retourne une **string** ***moto*** ou ***voiture*** avec **return**) dans le fichier de classe **vehicule.php**,

-Exécuter la méthode **detect()** sur les 2 objets voiture et moto dans le fichier **test_objet.php**.

-Afficher le type de véhicule dans le fichier **test_objet.php**,

-Créer une méthode **boost** qui ajoute 50 à la vitesse d'un objet dans le fichier de classe **vehicule.php**,

-Appliquer la méthode **boost** a la voiture dans le fichier **test_objet.php**,

-Afficher la nouvelle vitesse de la voiture dans le fichier **test_objet.php**.

Bonus :

-Créer une méthode **plusRapide()** dans le fichier **vehicule.php** qui compare la vitesse des différents véhicules (*moto* et *voiture*) et retourne le véhicule le plus rapide des 2 avec un **return**.

-Exécuter la méthode **plusRapide()** dans le fichier **test_objet.php**.

-Afficher le véhicule le plus rapide dans le fichier **test_objet.php.**

Exercice 3 Projet Task :

Créer toutes les **classes** du projet task (**user, task, category**) et ajouter les dans le dossier **model** du projet.

Créer un fichier pour chacune de nos classes (**user.php**, **task.php**, **category.php**).

PHP

20 Portée des objets :

En programmation orienté objet chacun de nos attributs, méthode vont avoir une portée en fonction du paramètre devant celle-ci :

Public : l'attribut ou propriété, méthode sera accessible depuis n'importe où dans notre projet c'est la valeur par défaut. Ce paramètre sera utilisé au niveau des **méthodes** afin que celle-ci soit *accessible depuis n'importe quel endroit de notre projet*. Il est **fortement déconseillé** de l'utiliser pour les attributs de notre classe. Pour y accéder nous créerons des méthodes **getter** et **setter** pour lire et écrire les **valeurs** des **attributs**.

Private : l'attribut ou propriété, méthode sera accessible uniquement au sein de la classe c'est la valeur que nous allons attribuer par défaut à nos attributs.

Protected : l'attribut ou propriété, méthode sera accessible depuis n'importe où dans le même dossier de notre projet. Ce paramètre pourra être utilisé au niveau des **méthodes** et des **attributs** afin que ceux-ci soit accessible depuis n'importe quel fichier contenu dans le **même répertoire**. Ce paramètre assure une sécurité à l'extérieur de notre dossier, les **méthodes** et **attributs** seront inaccessibles à **l'extérieur du dossier**.

PHP

Getter et setter :

Afin de ***sécuriser*** les **attributs** de nos **classes** nous allons tous les passer en mode **private**.

Pour **lire** et écrire du contenu au sein d'un **attribut** (valeur) nous allons créer autant de **méthodes** que d'**attributs** (**propriété**).

La **méthode getter** va nous permettre d'accéder aux **valeurs** d'un **attribut** d'une **classe** en **private**. Cette méthode sera toujours en public.

La **méthode setter** va nous permettre de modifier les **valeurs** d'un **attribut** d'une **classe** en **private**. Cette méthode sera toujours en public.

1 Passer les attributs de la classe en private :

```php
<?php
	class Vehicule
	{
		//Attributs :
		private $nomVehicule ;
		private $nbrRoue;
		private $vitesse ;
	}
?>
```

2 Ajouter les méthodes Getter et Setter :

Pour se faire nous allons utiliser la syntaxe suivante :

```
< ?php
        //Getter nomVehicule récupère le nom du véhicule
        public function getNomVehicule()
        {
        return $this->nomVehicule;
        }
        //setter nomVehicule remplace le nom du véhicule
        public function setNomVehicule($new_nom_vehicule)
        {
                $this->nomVehicule = $new_nom_vehicule;
        }
?>
```

3 modifier les méthodes existantes :

La fonction demarrer le véhicule va s'écrire sous la forme suivante :

```
<?php
//fonction demarrer le véhicule
public function demarrer()
{
      $demarrage = '<p>Démarrage de la '.$this->getNomVehicule().'Vrooom !!!!</p>';
      return $demarrage;
}
?>
```

Exercices :

Exercice 1 :

Créer un nouveau fichier **vehicule_private.php,**

Repartir de la base de la classe **Vehicule** et passer les tous les **attributs** en **private**,

Ajouter les **getters** et **setter**,

Editer les **méthodes** afin qu'elles s'adaptent avec les nouveaux paramètres.

Exercice 2 Projet Task :

Modifier toutes les **classes** du projet task (**user, task, category**) et ajouter les dans le dossier **model** du projet. En incluant les **Getter** et les **Setter** pour chacun des attributs.

Créer toutes les méthodes ajout (requête **insert** en mode **préparé**) que vous allez nommer :

createUser, **createCategory**, **createTask**.