PHP

Table des matières

Auteur : Mathieu MITHRIDATE
Relu, validé & visé par :
☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création : 08 / 12 / 2022
Date révision : xx / xx / 20xx

PHP

Auteur : Mathieu MITHRIDATE
Relu, validé & visé par :
☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création : 08 / 12 / 2022
Date révision : xx / xx / 20xx

PHP

Auteur : Mathieu MITHRIDATE
Relu, validé & visé par :
☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création : 08 / 12 / 2022
Date révision : xx / xx / 20xx

PHP

PHP

Emplacement table des matières suite.

PHP

1 Compétences pédagogiques :

Etre capable de comprendre le fonctionnement des variables

Etre capable de manipuler les opérateurs

Etre capable d'utiliser les instructions conditionnelles

Etre capable de manipuler un tableau

Etre capable de comprendre les boucles

Etre capable de créer et d'utiliser des fonctions

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

Etre capable de créer et utiliser les classes

Etre capable de créer et utiliser des objets

Etre capable de comprendre les notions d'héritage

Etre capable de comprendre les notions de polymorphisme

Être capable de créer des pages Web Dynamique

Etre capable de mettre en place un système d'API

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

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

Auteur : Mathieu MITHRIDATE

Relu, validé & visé par :
- Jérôme CHRETIENNE
- Sophie POULAKOS
- Mathieu PARIS

Date création : 08 / 12 / 2022

Date révision : xx / xx / 20xx

PHP

2 PHP c'est quoi ?

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

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** pour les plus connus.

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

L'interpréteur PHP va alors générer une page web **HTML**.

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

PHP

3 Environnement de travail :

Pour développer en **PHP** nous allons avoir besoin :

D'un **serveur**, **wamp**, **xampp** (*Windows*) ou **Lamp** (*Linux*) suivant notre environnement de travail.

-**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, Notepad++, PHPStorm, Sublime Text etc…*),

Pour tester notre code :

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

PHP

4 Syntaxe du langage :

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

4.1 Emplacement des fichiers, Exemple :

Dans le dossier **C:\wamp64\www\exemple*index.php*** (exemple du chemin avec wamp) du serveur apache (wamp, Xamp, Laragon, Lamp etc…)) nous allons créer un fichier **index.php**.

Nos scripts php devront être rédigés entre les balises :

```
<?php
?>
```

4.2 Exemple :

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

localhost/exemple/index.php

NB : le fichier doit être exécuter 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.*

PHP

4.3 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>
```

Chaque ligne de nos scripts devra se terminer par un **;**

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

4.4 Commenter des lignes de codes :

```
<?php
      //commentaire sur une ligne
      /*
      ------------------------------------
      Commentaire sur plusieurs lignes
      ------------------------------------
      */
?>
```

PHP

4.5 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

5 Les variables :

5.1 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).

5.2 Les types de variables :

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.

5.3 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.

Exemple d'utilisation d'une variable :

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

5.4 Exemple :

`$varInt` = 0 pour un entier (**int**),

`$varNom` = « nom » pour une chaine de caractères (**String**).

5.5 Afficher le contenu d'une variable :

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

```
<?php
	//initialisation d'une variable
	$nbr =2 ;
	//la fonction php echo permet d'afficher le contenu de la variable
nbr
	echo $nbr ;
?>
```

5.6 Afficher le type d'une variable :

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

```
<?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);
?>
```

5.7 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**).

6 Les opérateurs :

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

On utilise les opérateurs mathématiques suivant :

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

6.1 Exercices Opérateurs:

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, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC 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

7 Concaténation :

En php nous pouvons concaténer des valeurs entres elles. C'est à dire ajouter des chaines de caractères, des nombres, valeur de variable au sein d'une même suite de caractères.

7.1 Exemple :

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

```
<?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
	$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
	echo "ma chaine de caractères contient 32 caractères";
?>
```

Concaténer des variables dans des chaines de caractères :

```
<?php
	$concat1 = "ma chaine $var";//version avec encadrement "" ;
	$concat2 = 'ma chaine '.$var.''; //version avec encadrement '' ;
	$concat3 = 'ma chaine {$var}'; //version avec les Templates String
'' ;
?>
```

7.2 Exercices :

Exercice 1 :

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

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

Exercice 2 :

PHP

-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 les mots suivants :***l'adrar***,

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

-Cela doit donner :

```
<p>bonjour l'Adrar</p>
```

8 Les Fonctions :

Les fonctions permettent de rationaliser du code qui va être exécuté plusieurs fois, plutôt que réécrire de nombreuses fois les mêmes lignes nous allons créer une fonction. La fonction va exécuter le code quelle contient (instructions entre les accolades). Pour utiliser la fonction nous devrons l'appeler par son nom.

8.1 Création d'une fonction :

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

```
<?php
	function nom_de_la_fonction()
	{
		echo "Ma fonction"; //affiche Ma fonction dans la page HTML
	}
?>
```

8.2 Appel d'une fonction :

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

8.3 Exemple :

```
<?php
	function ma_fonction() //création de la fonction
{
	echo "Ma fonction"; //affiche Ma fonction dans la page HTML
```

```
}
ma_fonction();//appel de la fonction
?>
```

8.4 Création d'une fonction avec des paramètres :

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

8.5 Exemple :

```
<? php
    ma_fonction(10,5);
    function ma_fonction($a,$b){
    $result= $a+$b;
    return $result;
    }
?>
```

8.6 Exercices :

Exercice 1 :

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

-la fonction doit renvoyer le résultat de la soustraction $a-$b (**return**).

Exercice 2 :

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

- la fonction doit renvoyer l'arrondi (**return**) du nombre en entrée.

Exercice 3 :

-Créer une fonction qui prend en entrée **3 valeurs** et renvoie la **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).

9 Typage des fonctions :

Le langage PHP depuis la version **7.4** à intégrer le **typage des fonctions**. Cela consiste à déterminer le type de donnée des différents paramètres de nos fonctions ainsi que leur valeur de retour (return).

PHP

9.1 Typage des paramètres :

Nous allons pouvoir préfixer nos paramètres avec le type de données que l'on attend. Cela implique que le ou les paramètres attendus doivent respecter le typage déclaré.

Exemple de syntaxe :

```
< ?php
	function maFonction(int $nbr){
		echo 'le nombre '.$nbr.' est bien de
type'.gettype($nbr).'<br>';
	}
?>
```

Si nous donnons une valeur qui ne correspond pas en entrée de la méthode, PHP va nous retourner une **erreur fatale** (**Attention Le script va s'arrêter ici !**).

```
< ?php
	echo '<pre>';
	var_dump(maFonction('toto'));
	echo '</pre>';
?>
```

Ce message nous indique que **l'argument attendu** (**$nbr**) doit être de type **integer** et non de type **string**.

```
Fatal error:  Uncaught TypeError: maFonction(): Argument #1 ($nbr) must be of type int, string given
```

PHP

Dans le cas où nous passons en paramètre non pas un **string** mais un nombre **float** ou un **boolean**. La méthode nous renverra la **conversion** et le typage deviendra **integer**.

```
< ?php
	var_dump(maFonction(14.6));
	echo '</pre>';
	echo '<pre>';
	var_dump(maFonction(true));
	echo '</pre>';
?>
```

```
Le nombre 14 est bien de typeinteger
Le nombre 1 est bien de typeinteger
```

NB :

Il faut donc faire très attention quand on **type** des **paramètres**, car la **vérification** et la **conversion** peuvent changer le comportement de notre méthode et dans le cas **integer** ou on va passer un **string** à la place cela lève un fatale erreur.

9.2 typage null :

De la même manière que nous avons vu le typage (format de données) PHP nous permet de donner un type null à nos paramètres, retour de fonction. Nous utiliserons le ? pour spécifier ce typage.

Exemple :

```
< ?php
	function exemple ( ?string $variable) :?string{
	return strtoupper($variable) ;
	}
?>
```

NB : Veuillez noter que à partir de **PHP 9** nous ne pourrons plus utiliser le type **null** avec les **chaines de caractères**. Que ce soit avec les fonctions **natives** de PHP ou nos **propres fonctions**. PHP lèvera une **erreur fatale**.

9.3 Typage itération (iterable) :

PHP nous permet d'utiliser un type d'itération, pour passer un ensemble de paramètres et les stocker dans un tableau. Nous n'avons pas besoin donc de les lister. Imaginons que nous avons une fonction ou l'on souhaite ajouter la valeur de tous les paramètres (faire une addition) et que nous ne savons pas à l'avance le nombre de valeurs à ajouter. Le type iterable va nous permettre de résoudre le problème.

```
< ?php
	function argumentVariables2(...$tab){
```

```
        $sum = 0;
        $cpt = 0;
        foreach ($tab as $value) {
              $sum += $value;
              $cpt++;
        }
        return 'Il y a :'.$cpt.' arguments, la somme vaut :'.$sum;
    }
    //la méthode fonctionnera peu importe le nombre de valeur que nous
    lui assignerons à son exécution.
    echo argumentVariables2(12,22,33,55,46) ;
    echo argumentVariables2(5,2,3) ;
?>
```

NB : Attention le type **iterable** ne peut être utilisé qu'en **dernier paramètre** d'une fonction.

10 Les conditions :

Les conditions vont nous permettre de tester, vérifier des valeurs et exécuter dans le cas où la condition se trouve vérifiée le code correspondant. Pour cela nous allons rédiger la syntaxe suivante :

if (***si vérifié***),

else if (**sinon si vérifié,** *nous pouvons en utiliser plusieurs*),

else (***sinon tous les autres cas***).

10.1 Opérateurs de comparaison :

Exemple	Nom	Résultat
$a == $b	Egal	true si $a est égal à $b après le transtypage.
$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.
$a <> $b	Différent	true si $a est différent de $b après le transtypage.
$a !== $b	Différent	true si $a est différent de $b ou bien s'ils ne sont pas du même type.
$a < $b	Plus petit que	true si $a est strictement plus petit que $b.
$a > $b	Plus grand que	true si $a est strictement plus grand que $b.
$a <= $b	Inférieur ou égal	true si $a est plus petit ou égal à $b.
$a >= $b	Supérieur ou égal	true si $a est plus grand ou égal à $b.

$a <=> $b $a est	Combiné	Un entier inférieur, égal ou supérieur à zéro lorsque Inférieur, égal, ou supérieur à $b respectivement.

10.2 Opérateurs logiques :

Nous allons également avoir besoin des opérateurs logiques :

Exemple	Nom	Résultat
$a and $b	And (Et)	true si $a ET $b valent 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.
$a && $b	And (Et)	true si $a ET $b sont true.
$a || $b	Or (Ou)	true si $a OU $b est true.

10.3 Exemples :

```php
<?php
        $a = 6;
        if($a<=3 and $a >0)
        {   //test si $a est plus petit que 3 et est supérieur à 0
            echo "la valeur de la variable \$a est plus petite que 3";
        }
        else if($a>=3 && $a <5)
        {   //test 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
        {   //test autre cas
            echo "la valeur de la variable \$a est supérieur à 5";
        }
    ?>
```

10.4 Test Switch case :

Le switch case va nous permettre d'exécuter du code en fonction de la valeur d'une variable. Nous allons gérer des **cas**. Le switch case permet de vérifier différents **cas** (valeurs), le code associé sera alors exécuté si le **cas** est **vérifié** (l*a valeur correspond*).

10.5 Exemple Switch Case :

```
<?php
    $value = 5; //variable value qui vaut 5
    switch($value){//vérification de la valeur contenue dans $value.
        case 1 : //cas si $value vaut 1
            echo '\$value est égale à 1'; //affiche $value est égale à 1
            break; //cette instruction arrête le code ici, arrêt on sort du
switch.
        case 2 : //cas si $value vaut 2 //affiche $value est égale à 2
            echo '\$value est égale à 2'; //affiche $value est égale à 2
            break; //cette instruction arrête le code ici, arrêt on sort du
switch.
        case 5 : //cas si $value vaut 5
            echo '\$value est égale à 5'; //affiche $value est égale à 5
            break; //cette instruction arrête le code ici, arrêt on du
switch.
    }
?>
```

10.6 Exercices :

Exercice 1 :

-Créer une fonction qui teste si un nombre est **positif** ou **négatif** (**echo** dans la page web).

Exercice 2 :

-Créer une fonction qui prend en entrée **3 valeurs** et retourne le nombre le plus **grand** (**echo** dans la page web).

Exercice 3 :

-Créer une fonction qui prend en entrée **3 valeurs** et retourne le nombre le plus **petit** (**echo** dans la page web).

PHP

Exercice 4:

-Créer une fonction qui prend en entrée **1 valeur** (l'âge d'un enfant). Ensuite, elle informe de sa **catégorie** (**echo** dans la page web) :

- "Poussin" de 6 à 7 ans
- "Pupille" de 8 à 9 ans
- "Minime" de 10 à 11 ans
- "Cadet" après 12 ans

Bonus : Refaire l'exercice en utilisant le **switch case**.

Auteur : Mathieu MITHRIDATE
Relu, validé & visé par :
☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création : 08 / 12 / 2022
Date révision : xx / xx / 20xx

PHP

11 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. Les boucles permettent également de parcourir des chaînes de caractères, tableaux et des objets.

Pour écrire une boucle (**for « pour »**), nous allons utiliser la syntaxe ci-dessous :

11.1 Exemple boucle for :

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

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

La boucle va afficher 10 fois *'Ceci est une boucle for en PHP'*.

Pour écrire une boucle (**while « tant que »**) :

11.2 Exemple boucle while :

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

```
<?php
	$i = 0;//variable compteur à l'extérieur de la boucle
	while ($i < 10) //boucle while tant que $i est plus petit que 10
	{
		//j'affiche la valeur de $i
		echo $i;
		//à chaque tour j'incrémente $i (+1)
		$i++;
		//je saute une ligne
		echo '<br>';
	}
?>
```

La boucle va afficher 0 1 2 3 4 5 6 7 8 9 (en sautant une ligne à chaque tour).

Pour écrire une boucle (**foreach « pour chaque »**) :

11.3 Exemple boucle foreach :

Ce type de boucle est en général utilisée pour parcourir un tableau, des chaines de caractères ou des objets.

Version tableau indexé (numéroté) :

```
<?php
	$tableau = array(' valeur1', ' valeur2', ' valeur3', ' valeur4');
	foreach($tableau as $valeur)
	{
	    echo "$valeur<br/> ";
	}
?>
```

Cette boucle va afficher le contenu de chaque colonne du tableau avec un saut à la ligne (valeur1, valeur2, valeur3, valeur4).

Version tableau associatif :

```
<?php
        $tableau = [' Nom'=>'Mithridate', ' Prénom'=>'Mathieu',  ' Age=> 42];
        foreach($tableau as $cle=> $valeur)
        {
            echo "$cle : $valeur<br/> ";
        }
?>
```

Cette boucle va afficher le nom de la colonne et la valeur associée pour chaque entrée du tableau avec un saut à la ligne (Nom : Mithridate, Prénom : Mathieu, Age : 42).

11.4 Exercices :

Exercice 1 :

Créer un script qui affiche les nombres de 1 -> 5 (*méthode echo*).

Exercice 2 :

Ecrire une fonction qui prend un nombre en paramètre (*variable **$nbr***), et qui ensuite affiche les dix nombres suivants. Par exemple, si la valeur de nbr équivaut à : 17, la fonction affichera les nombres de 18 à 27 (*méthode echo*).

PHP

12 les tableaux :

Un tableau PHP a pour fonction de stocker et manipuler des informations.

Les tableaux, aussi appelés **arrays** en anglais, sont des types de données structurés permettant de grouper des informations ensemble. Les tableaux peuvent stocker une ou plusieurs valeurs à la fois (de types différents).

Lors de la déclaration d'un tableau, il est inutile de préciser sa dimension et le type de données qu'il va contenir. PHP s'en charge tout seul. Chaque fois que l'on va ajouter une nouvelle entrée enregistrée dans le tableau, PHP agrandit sa taille de 1 élément.

Le langage PHP propose également deux types distincts de tableaux : les tableaux à **index numériques** et les tableaux **associatifs**.

12.1 Déclaration d'un tableau :

La déclaration d'un tableau vide se fait de la même manière qu'une variable, c'est à dire avec un signe dollars ($) et un nom.

Pour déclarer un nouveau tableau, il suffit d'utiliser la structure de langage **array()**. Cette fonction prend en paramètres facultatifs (séparés par une virgule), les valeurs que l'on souhaite insérer dans le tableau pour l'initialiser. Si rien n'est précisé en paramètre, le tableau créé sera vide. Un tableau **commence** toujours à **l'index 0**.

PHP

12.2 Exemple déclaration de tableaux indexé numériquement et associatif :

```php
<?php
	//déclaration d'un tableau vide (tab) :
	$tab = array() ;
	//ou version []
	$tab = [] ;
	//déclaration d'un tableau indexé numériquement :
	$tab1 = array(1,8,7,11) ;
	//ou version []
	$tab1 = [1,8,7,11) ;
	//déclaration d'un tableau associatif :
	$identite = array(
		'nom' => 'Mithridate',
		'prenom' => 'Mathieu',
		'age' => 41,
		'estFormateur' => true
	);
	//ou version []
	$identite = ['nom' => 'Mithridate', 'prenom' => 'Mathieu', 'age' =>
	41, 'estFormateur' => true] ;
?>
```

12.3 Exemple ajouter une valeur à un tableau :

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

12.4 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 => $value) {
		echo '<br>';
		//Affiche le contenu de la case à chaque tour.
		print_r($value);
	}
?>
```

12.5 Exercices :

Exercice 1 :

-Créer une fonction qui affiche la valeur la plus **grande** du tableau.

Exercice 2 :

-Créer une fonction qui affiche la **moyenne** du tableau.

Exercice 3 :

-Créer une fonction qui affiche la valeur la plus **petite** du tableau.

PHP

13 Les super globales :

Le transfert de données entre des pages web est géré en PHP par le biais de variables spéciales qui s'appellent super globale. Dans cette partie nous allons voir les Super Globales suivantes : **$_GET** et **$_POST**.

Chacune de ces variables va récupérer dans un tableau le contenu des différents champs html d'un formulaire.

Les formulaires html possèdent 2 méthodes d'envoi possibles **get** et **post**.

Get fait passer les informations dans l'url de la page, cette méthode est dangereuse car elle affiche dans l'url de la page le nom des variables et leur contenu.

Post fait passer les informations par le body de la page cette méthode est à privilégier car elle est plus sécurisée et surtout elle permet de transférer des informations de taille plus importante.

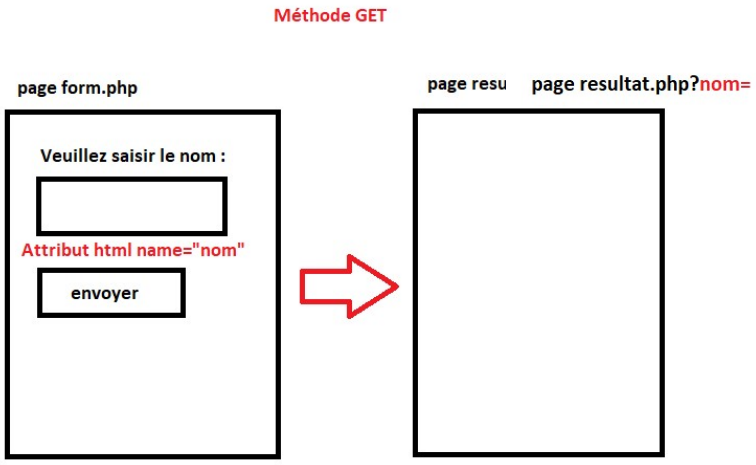
On devra utiliser l'attribut html (***name***) pour chaque élément du formulaire.

PHP

13.1 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 :



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**.

PHP

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

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

*Page **resultat.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**.

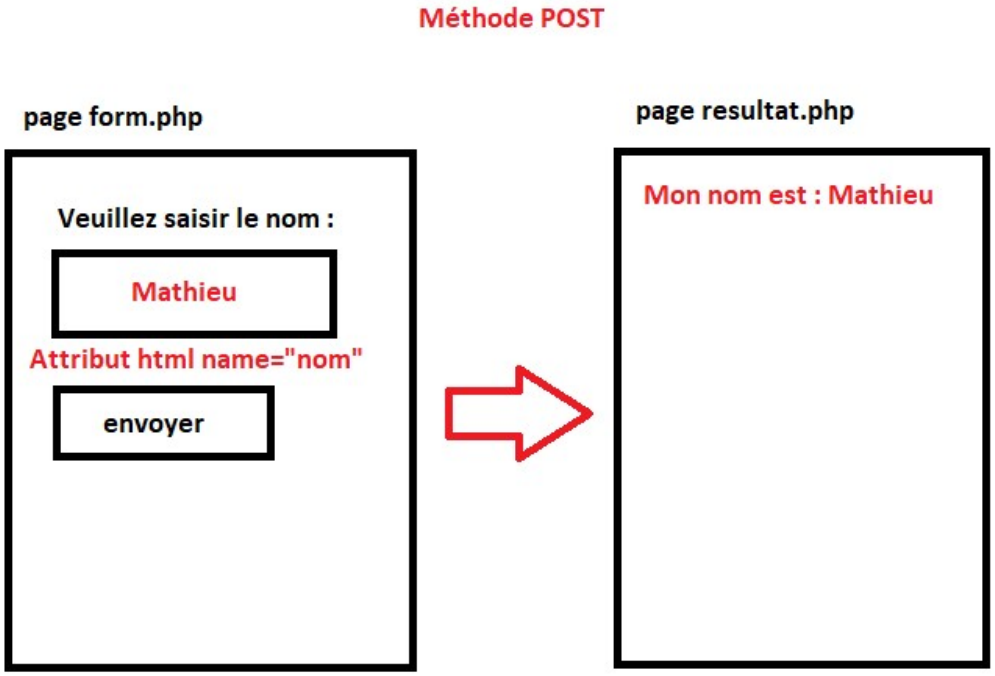
1 On vérifie l'existence de la super globale ***$_GET['nom']*** avec la fonction PHP ***isset()*** qui teste si la variable ***existe*** et si sa **valeur** n'est pas égal à **null**.

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 Mathieu***.

13.2 Fonctionnement POST :

Le contenu des champs de formulaire va transiter par le body de la page 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 : Mathieu***.

PHP

NB :

Si l'on souhaite traiter les données dans la page de formulaire, dans la partie action (html) on laisse soit le champ **vide** ou ont saisie **#**

13.3 Récupération d'inputs checkbox (HTML -> formulaire) mode POST :

Les cases à cocher (input->checkbox en HTML) se récupèrent de la façon suivante (si la case est cochée) en PHP :

1 Création d'un formulaire (méthode POST en HTML),

Dans le formulaire nous allons ajouter des inputs de type checkbox comme ci-dessous :

```
<p><input type="checkbox" name="box[]" value="1"/>1</p>
```

L'attribut **name** de chaque checkbox doit être le même, l'attribut **value** doit être unique. La super globale POST va stocker un tableau de toutes les attributs **value**.

Nous récupérerons le contenu de **value** avec une boucle **foreach** de **box[]** (car c'est un tableau, il s'appelle box pour l'exemple).

NB : Attention, les **checkbox** n'existent dans la super globale **POST** que si elles sont **cochées**.

PHP

Exemple transfert de données en post :

*Page **form.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>Récupération des checkBox cochées :</title>
</head>
<body>
    <h4>Cocher une ou plusieurs checkbox :</h4>
    <!--Formulaire HTML-->
    <form action="" method="post">
        <p><input type="checkbox" name="box[]" value="1"/>1</p>
        <p><input type="checkbox" name="box[]" value="2"/>2</p>
        <p><input type="checkbox" name="box[]" value="3"/>3</p>
        <p><input type="checkbox" name="box[]" value="4"/>4</p>
        <p><input type="checkbox" name="box[]" value="5"/>5</p>
        <p><input type="submit" value="Récupérer"></p>
    </form>
    <h4>Liste des checkbox cochées :</h4>
    <!--Code PHP-->
    <?php
        //vérification de la super globale $_POST['box']
        if(isset($_POST['box'])){
            //boucle pour parcourir chaque case cochés ($value équivaut à
value en HTML)
            foreach($_POST['box'] as $value){
                echo "<p>id de la box : $value</p>";
            }
        }
        else{
            echo "<p>Veuillez cocher une ou plusieurs checkbox</p>";
        }
    ?>
</body>
</html>
```

PHP

13.4 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

14 Interaction avec une base de données :

Le langage PHP permet d'interagir de façon simple et sécurisé (dans certains cas) avec des bases de données de type MYSQL (propriétaire ORACLE) ou MARIADB (équivalent open source).

Pour se faire nous devons respecter certaines étapes :

1 Se connecter à la base de données,

2 Exécuter la requête SQL,

3 Récupérer le résultat dans une variable (pour les requêtes de type select),

14.1 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 utiliserons la syntaxe suivante :

//connexion à la base de données

```
$bdd = new PDO('mysql:host=localhost;dbname=nom_de_la_bdd', 'root','',
[PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)];
```

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 l'url est localhost : identique au serveur apache) et son nom ***dbname*** = ***nom_de_la_bdd'***

-le paramètre suivant est le nom du compte dans l'exemple ci-dessus : ***'root',***

-le paramètre suivant est le mot de passe dans l'exemple ci-dessus il est vide : ***'',***

-le paramètre **array** ou **[]** (*tableau*) permet de spécifier le mode de l'objet **PDO** dans l'exemple ci-dessus il active le mode d'erreur avancé (*code d'erreur SQL*).

14.2 Exécution d'une requête SQL :

Pour interagir avec notre base de données et exécuter des requêtes SQL il existe plusieurs méthodes nous allons en voir 2 types :

-Les requêtes classiques qui ne sont pas sécurisées (*elles sont sensibles aux attaques par **injection SQL***),

-Les requêtes préparées qui elles sont plus sécurisées et bloquent l'**injection SQL**.

14.3 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) :

```
//Connexion à la base de données
$bdd = new PDO('mysql:host=localhost;dbname=nom_de_la_bdd', 'root','',
array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
//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 la variable $data, $req stocke la requête SQL.
	$req = $bdd->query('SELECT * FROM utilisateur');
	//boucle pour parcourir et afficher le contenu de chaque ligne de la
table
	while ($data = $req->fetch())
	{
		//affichage les informations d'une colonne de la bdd par son
		non d'attribut
		echo '<p>'.$data['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 **$data** le résultat de toute la requête **SQL** : ***select***, (*un tableau qui contient le résultat*).

La boucle **while** va nous permettre de parcourir le contenu de la variable **$data** et afficher pour chaque enregistrement de la base de données le contenu d'un **attribut** de la table sous la forme :

$data['nom_attribut'],

-L'afficher avec la méthode **echo** dans un paragraphe **html** (balise **p**).

Try catch :

Le paramètre ***PDO::ERRMODE_EXCEPTION*** dans le fichier de connexion active la gestion des exceptions.

Le **try catch** va nous permettre d'exécuter le code dans le **try**, s'il y a une erreur (*requête, connexion ou autre*) le message d'erreur sera redirigé dans le **catch** et s'affichera dans la page (*code erreur SQL*).

14.4 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.

```
//Connexion à la base de données
$bdd = new PDO('mysql:host=localhost;dbname=nom_de_la_bdd', 'root','',
array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
//Préparation de la requête SQL nous stockons dans une variable $req la
requête à exécuter
$req = $bdd->prepare('SELECT * FROM utilisateur WHERE nom_utilisateur
= :nom_utilisateur');
//Exécution de la requête SQL création à l'aide d'un tableau qui va
contenir le ou les paramètres à affecter à la requête SQL
$req->execute(array(
            'nom_utilisateur' => iconv("UTF-8", "ISO-8859-1//TRANSLIT",
$nom_utilisateur),
            ));
//boucle pour parcourir et afficher le contenu de chaque ligne de la table
while ($donnees = $req->fetch())
{
      //affichage des données d'une colonne du résultat de la requête par
      son non d'attribut (nom champ bdd)
      echo '<p>'.$donnees['nom_attribut'].'</p>';
{
```

Cette requête effectue le même traitement que la requête classique mais de façon **sécurisé**.

14.5 Méthode alternative (utilisation de return) :

Dans une requête (classique ou préparée) nous avons la possibilité de faire en sorte qu'elle retourne un tableau :
Associatif ou bien un tableau d'objet. Nous allons pour se faire utiliser la méthode de la classe PDO ***fetchAll***.

La méthode fetchAll prend en paramètre les options suivantes :

FETCH_ASSOC -> renvoie un tableau associatif ***fetchAll(PDO::FETCH_ASSOC),***

FETCH_OBJ -> renvoie un tableau associatif ***fetchAll(PDO::FETCH_OBJ).***

Le mot clé **return** va renvoyer en sortie l'option choisie (tableau associatif ou tableau d'objet).

Exemple de fonction qui retourne un tableau associatif :

```
function showAllArticle($bdd):array{
	try{
		$req = $bdd->prepare('SELECT * FROM article');
		$req->execute();
		$data = $req->fetchAll(PDO::FETCH_ASSOC);
		return $data;
	}
	catch(Exception $e)
	{
		//affichage d'une exception en cas d'erreur
		die('Erreur : '.$e->getMessage());
	}
}
```

PHP

14.6 Exercices :

Exercice 1 :

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 :

id_article (clé primaire),

nom_article de type varchar(50),

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 :

Un champ input avec comme attribut html **name = «nom_article »**,

Un champ input avec comme attribut html **name = «contenu_article »**,

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

Exercice 2 :

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

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

NB : On peut également utiliser **return** dans la fonction, et parcourir le résultat avec une boucle **foreach** :

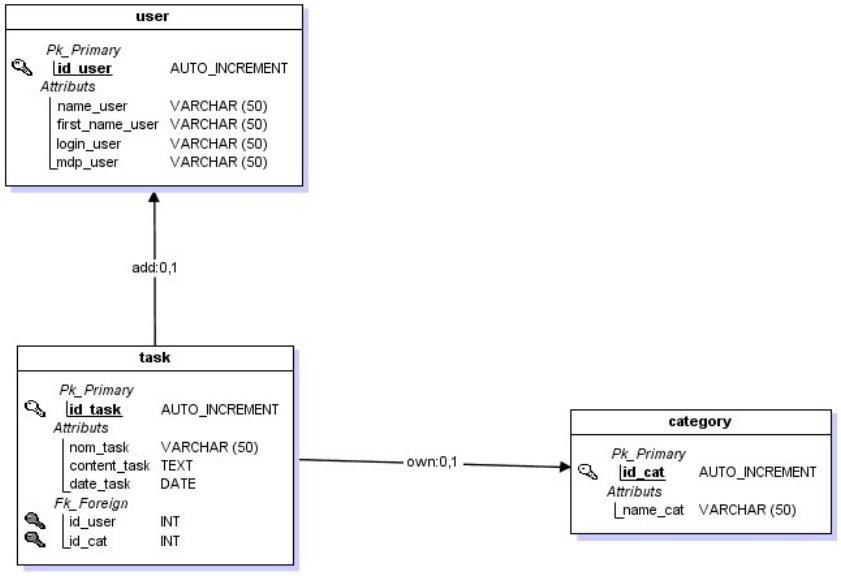
```
$list = getAllArticle($bdd);
//on stocke le résultat de la fonction dans une variable
foreach($list as $value)
{
        echo '<p>Numéro de l'article : '.$value['id_article'].'</p>' ;
}
/*on parcourt avec foreach le contenu du tableau et on echo le résultat.*/
```

PHP

Exercice 3 PROJET TASK Partie 1 :

a) Créer une base de données **MYSQL** depuis le **MLD** ci-dessous :

-Nom de la BDD : «**task**»,



b) Créer une page php qui va contenir un formulaire html avec comme méthode POST (balise **form**) cette page va nous permettre de créer nos comptes utilisateurs et les sauvegarder dans la base de données.

-A l'intérieur du formulaire ajouter les champs suivants :

Un champ input avec comme attribut html **name = «name_user»**,

Un champ input avec comme attribut html **name = «first_name_user»**,

Un champ input avec comme attribut html **name = «login_user»**,

Un champ input avec comme attribut html **name = «mdp_user»**,

Un champ input de type **submit** avec comme attribut html **value = « Ajouter »**

c) Ajouter le code php suivant :

-Créer 4 variables **$name_user**, **$first_name_user**, **$login_user**, **$mdp_user**,

PHP

-Importer le contenu des super globales **$_POST['name_user']**, **$_POST['first_name_user'], $_POST['login_user'], $_POST['mdp_user'],** et tester les avec la méthode ***isset() (dans la condition if)*** dans les variables créées précédemment (**$name_user, $first_name_user, $login_user, $mdp_user**),

-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 4 champs dans un nouvel enregistrement (requête **SQL insert**),

-Afficher après l'insertion en base de données les informations que vous avez saisies (nom, prenom, login, mot de passe).

d)Bonus :

-Afficher en bas de la page la liste des comptes utilisateurs créés avec une requête **SQL select**,

-Utiliser une requête **SQL préparée**.

PHP

15 Importer des fichiers Super Globale $_FILES :

Nous avons la possibilité en PHP d'importer des fichiers à l'intérieur du répertoire de notre projet exemple : (www\nom_du_projet\)

Pour ce faire nous allons utiliser la super globale **$_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**) le serveur va lui donner un nom temporaire ex tmp_1569565322.jpg.

15.1 import d'une image Formulaire :

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="file" name="file">
        <p><button type="submit">importer</button></p>
    </form>
</html>
```

Le fichier va être importé en mode **POST** dans le dossier **/tmp** à la racine du serveur.

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

-Nous allons créer un nouveau répertoire **import** à la racine du serveur web (*www/import ou htdocs\import*),

-Créer un dossier image à la racine du projet import,

-Créer un fichier index.php et coller à l'intérieur le code html de la page précédente,

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

15.2 Script d'import PHP :

PHP

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
    /*-----------------------------------------------------
                    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, "./image/$name");
    }
```

PHP

```
/*-----------------------------------------------------
                    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="index.php" method="POST" enctype="multipart/form-data">
        <h2>importer une image</h2>
        <input type="file" name="file">
        <p><button type="submit">importer</button></p>
    </form>
</html>
```

Exercice TP 1 :

Réalisez le TP file import (importer des images->BDD).

PHP

16 Modèle MVC :

16.1 Théorie Modèle MCV :

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 :

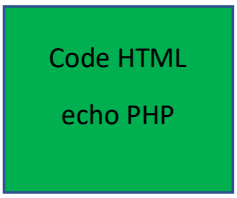
Modèle : dans cette partie nous allons déplacer toute les parties du code qui vont nous permettre l'accès aux données, afin de les préparer pour le contrôleur. C'est tout ce qui va concerner les requêtes **SQL**.

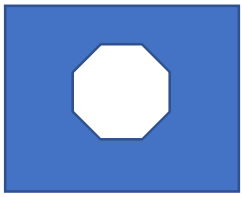
Vue : la vue se concentre sur toute la partie affichage, c'est les interfaces que l'utilisateur final va voir et avec lesquelles il va interagir dans son navigateur internet.

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

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

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

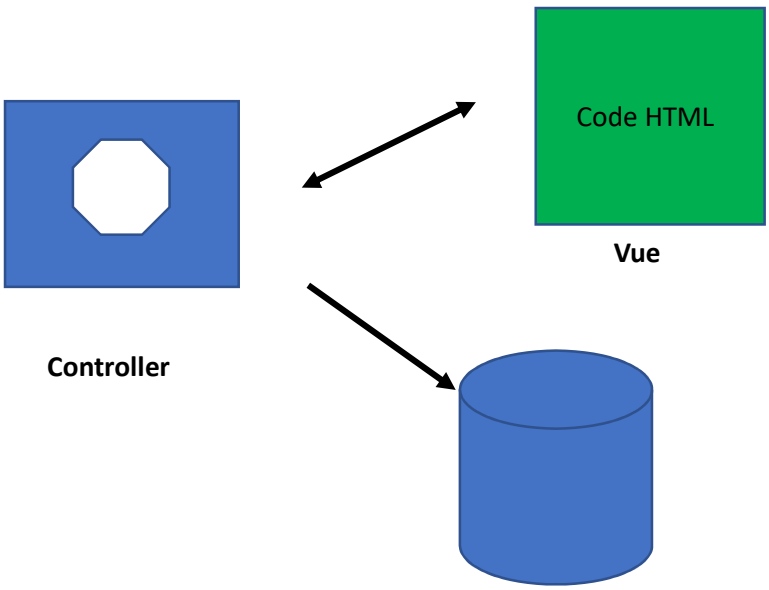




Modèle (*accès à la base de données*) Vue (*contiens le code HTML*) Controller (*logique et traitement*)

PHP

Echange entre les différentes couches :



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

16.2 Exemple :

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

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

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

```
<?php
    function addArticle($bdd, $name, $content){
        try
        {
            //Exécution de la requête SQL insert
            $req = $bdd->prepare('INSERT INTO article(nom_article,
contenu_article)
            VALUES(?,?)');
            $req->bindParam(1, $name, PDO::PARAM_STR);
            $req->bindParam(2, $content, PDO::PARAM_STR);
            return "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());
        }
    }
      addArticle($bdd, $name, $content) ;
```

PHP

```
?>
```

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','',
	[PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)];
?>
```

Enfin nous allons déplacer la logique (*conditions*) dans une nouvelle page (*qui sera notre **controller***) que nous allons nommer **controller_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>';
	}
	?>
```

16.3 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

17 Classe et objet :

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

17.1 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 (ou un plan de construction) qui va nous permettre de créer des **objets** :

Gardons 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**.

17.2 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
        class Vehicule{
        }
?>
```

17.3 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**.

17.4 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** à notre véhicule pour définir son nom, son nombre de roues sa vitesse.

17.5 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
<?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 (pour l'héritage entre classe)**.

17.6 Créer et appeler des méthodes :

Dans nos classes nous avons la possibilité de créer des **fonctions** (*méthodes*) qui serons utilisables par nos objets.

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

17.6.1 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 : "nom du véhicule » Vrooom !!!!"
```

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

```
<?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>";
	}
}
?>
```

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

17.6.2 Appel d'une méthode :

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

17.7 Constructeur

Pour pouvoir affecter des valeurs à un objet directement au moment de son instanciation nous devons ajouter une méthode à la classe. Celle-ci se nomme constructeur (c'est la méthode qui est appelée au moment du new Classe()).

Par défaut un constructeur vide est disponible dans la classe (même si la méthode n'est pas écrite dans la classe).

Pour créer un constructeur personnalisé (qui va imposer de donner des valeurs à différents attributs) nous utiliserons la syntaxe suivante :

```php
<?php
    class Classe{
        //attributs
        public $nom;
        public $taille;
        //constructeur
        public function __construct(string $nom, ?int $taille){
            $this->nom = $nom;
            $this->taille = $taille;
        }
    }
?>
```

Quand nous instancierons des objets depuis cette classe nous devrons **obligatoirement** renseigner les valeurs définies en paramètre :

```php
//la classe oblige à renseigner une valeur pour l'attribut nom et taille
$classe = new Classe('nom', 20);
```

17.8 Méthode toString :

Si l'on ***echo*** un ***objet*** on se retrouve avec une erreur de type :
`Object of class` ***NomClasse could not be converted to string***. Pour corriger cette erreur nous allons devoir recréer la ***magic method*** **__toString** avec la syntaxe suivante :

```php
public function __toString(){
      return $this->attribut;
      //ici on remplace la valeur par l'attribut de la classe que l'on
      souhaite afficher (par ex le //nom)
}
```

17.9 Exercices :

Exercice 1 :

PHP

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 :

```
"<p>la surface de nomMaison est égale à : x m2</p>".
```

Bonus

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

Modifier la méthode **surface** pour qu'elle prenne en compte le paramètre **nbrEtage**.

Auteur : Mathieu MITHRIDATE
Relu, validé & visé par :
☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création : 08 / 12 / 2022
Date révision : xx / xx / 20xx

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 **Vehicules** 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 Vehicule** 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 Vehicule le plus rapide des 2 avec un **return**.

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

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

Auteur : Mathieu MITHRIDATE
Relu, validé & visé par :
☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création : 08 / 12 / 2022
Date révision : xx / xx / 20xx

PHP

18 Portée des objets (encapsulation) :

En programmation orienté objet chacun de nos attributs, méthode vont avoir une portée d'utilisation en fonction du paramètre devant celle-ci on parle d'encapsulation :

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

18.1 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.

18.1.1 Passer les attributs de la classe en private :

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

18.1.2 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;
	}
?>
```

18.2 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;
        }
?>
```

18.3 Exercices :

Exercice 1 :

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

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

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

Editer les **méthodes** afin qu'elles s'adaptent avec les nouveaux paramètres (utilisation des **getters** et des **setters** plutôt que les **attributs**).

PHP

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 ajouts (requête **insert** en mode **préparé**) que vous allez nommer :

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

PHP

19 Super Globale SESSION et connexion :

19.1 Super Globale SESSION :

La super globale **SESSION** est une super globale particulière, elle va nous permettre de faire transiter des données au travers d'un site web. L'autre particularité de la super globale **SESSION** est que à la différence des supers globales **GET**, **POST**, **FILES** (qui sont générées par un formulaire) nous allons pouvoir les **créer** et y **associer** les valeurs de notre **choix**.

La super globale SESSION est stockée coté **serveur**.

Pour utiliser les supers globale **SESSION**, nous appellerons la fonction native de PHP :

```
session_start();
```

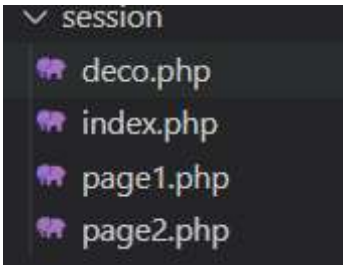
Cette fonction va nous permettre de **créer**, **utiliser**, **modifier** les supers globales **SESSION** dans **chaque page** ou la **méthode** est **appelée**.

Pour détruire les super globales SESSION et la session affectée nous utiliserons la fonction native de PHP :

```
session_destroy();
```

19.2 Exemple :

Nous allons créer un projet PHP avec la structure suivante :



PHP

Création d'une page **index.php** qui va contenir le code suivant :

```
<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>Connexion</title>
</head>
<body>
	<p><a href="page1.php">Page1</a></p>
	<p><a href="page2.php">Page2</a></p>
	<p><a href="deco.php">Déconnexion</a></p>
</body>
</html>
<?php
	//démarrage de la session (à utiliser sur toutes les pages)
	session_start();
	$_SESSION['name'] = "Mathieu";
	//affichage du contenu de la super globale $_SESSION['name']
	echo ''.$_SESSION['name'].' est connecté';
	//test si on à parcouru la page page1.php
	if(isset($_SESSION['page1'])){
		echo '<p>Nous avons visité la page1.php</p>';
	}
	//test si on à parcouru la page page1.php
	if(isset($_SESSION['page2'])){
		echo '<p>Nous avons visité la page2.php</p>';
	}
	//test déconnecté
	if(isset($_GET['deco'])){
		echo '<p>Déconnecté</p>';
	}
?>
```

PHP

Création d'une page **page1.php** qui va contenir le code suivant :

```
<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>page1</title>
</head>
<body>
    <p><a href="index.php">Home</a></p>
    <p><a href="page2.php">Page2</a></p>
    <p><a href="deco.php">Déconnexion</a></p>
</body>
</html>
<?php
    //démarrage de la session (à utiliser sur toutes les pages)
    session_start();
    //création super globale $_SESSION['page1']
    $_SESSION['page1'];
    //affichage du contenu de la super globale $_SESSION['name']
    echo ''.$_SESSION['name'].' est connecté';
    //test si on à parcouru la page page1.php
    if(isset($_SESSION['page1'])){
        echo '<p>Nous avons visité la page1.php</p>';
    }
    if(isset($_SESSION['page2'])){
        echo '<p>Nous avons visité la page2.php</p>';
    }
?>
```

PHP

Création d'une page **page2.php** qui va contenir le code suivant :

```
<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>page2</title>
</head>
<body>
    <p><a href="index.php">Home</a></p>
    <p><a href="page1.php">Page1</a></p>
    <p><a href="deco.php">Déconnexion</a></p>
</body>
</html>
<?php
    //démarrage de la session (à utiliser sur toutes les pages)
    session_start();
    //création super globale $_SESSION['page1']
    $_SESSION['page2'];
    //affichage du contenu de la super globale $_SESSION['name']
    echo ''.$_SESSION['name'].' est connecté';
    //test si on à parcouru la page page1.php
    if(isset($_SESSION['page1'])){
        echo '<p>Nous avons visité la page1.php</p>';
    }
    if(isset($_SESSION['page2'])){
        echo '<p>Nous avons visité la page2.php</p>';
    }
?>
```

PHP

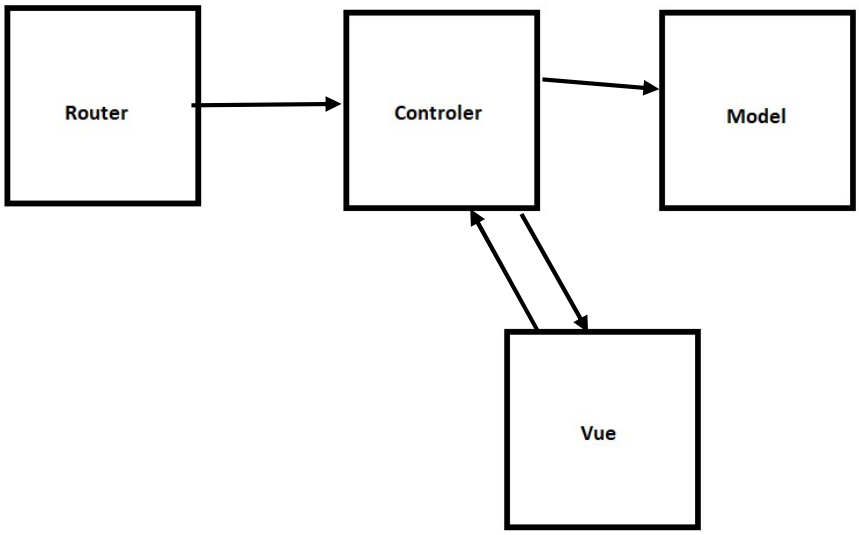
Création d'une page **deco.php** qui va contenir le code suivant :

```
<?php
    //démarrage de la session (à utiliser sur toutes les pages)
    session_start();
    //destruction de la session
    session_destroy();
    //redirection index.php
    header('Location: ./index.php?deco');
?>
```

20 Le Routing :

Le **routing** consiste à rediriger les différentes URL vers le bon controller. Il vient en complément du modèle **MVC** et se traduit de la sorte :

20.1Exemple de schéma de routing :



Toutes nos requêtes **http**, **https** passeront par le **router** (**index.php**). Celui-ci redirigera vers le bon **controller**.

PHP

20.1 Réécriture des URL :

Pour la mise en place de notre **router** (redirection de toutes les requête vers **index.php**), nous allons utiliser la fonction de **réécriture** des **url** du serveur **apache**. Pour cela nous allons déposer un fichier **.htaccess** à la racine de notre projet.

Si notre projet se trouve dans un dossier **projet** à la racine du serveur **web** (*Apache*), nous ajouterons un fichier .**htaccess** à la racine de celui-ci qui va contenir le code suivant :

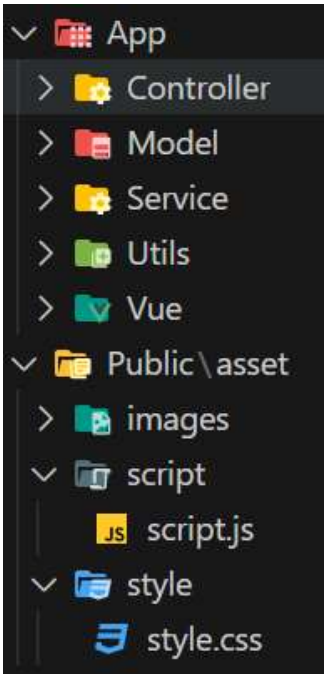
```
#Activation du rewrite des URL
RewriteEngine On
#base du projet (emplacement à partir de la racine du serveur)
RewriteBase /projet
#si ce n'est pas un répertoire
RewriteCond %{REQUEST_FILENAME} !-d
# Si ce n'est pas un fichier
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.+)$ index.php [QSA,L]
```

Le code ci-dessus active la réécriture des url du serveur apache, il prend le dossier **/projet** comme base du projet (**racine**).
L'adresse http du projet sera la suivante :

http://localhost/projet.
la page **router** sera **index.php** (défini dans le fichier **.htaccess**).

20.2 Structure du projet en MVC Objet :

Nous allons structurer notre projet en modèle **MVC** (**Model** (*data*), **Vue** (*interface*), **Controller** (*logique*),



Le dossier **Public** contiendra l'ensemble des ressources accessibles par l'utilisateur final (*navigateur web*),
Le dossier **asset** contiendra tous les **fichiers ressources** (images, script JS, style CSS etc…),

Le dossier **APP** (ou src) contiendra l'ensemble du code source du projet,
Le dossier **Controller** contiendra tous les **classes controller** (**logique**),

Le dossier **Model** contiendra toutes les **classes** (**data**),

Le dossier **Utils** contiendra la classe de connexion à la **bdd et les classes utilitaires**,

Le dossier **Vue** contiendra toutes les **interfaces** (**vues et templates HTML**).

A la racine du projet nous retrouverons le **router** (**index.php**) ainsi que le fichier de réécriture des url **.htaccess Apache**.

En plus de ces répertoires nous retrouverons d'autre répertoire comme **vendor**, **Services**, **Tests etc** …

20.3 Création du routeur :

Nous allons créer un fichier (**index.php**) qui va rediriger toutes les requêtes de la sorte :

-Exemple :

Racine du projet :

Fichier **index.php** (routeur).

projet/index.php -> http://localhost/projet (*url à saisir dans le navigateur*)

-Exemple :

Racine du projet :

Fichier **test.php** (fichier de test, code ci-dessous) :

```
< ?php
        echo "test" ;
?>
```

projet/test.php -> http://localhost/projet/test (*url à saisir dans le navigateur*)

-Exemple :

Controller ajouter un utilisateur :

projet/controller/controller_add_user.php -> http://localhost/projet/addUser (*url à saisir dans le navigateur*)

Toutes les requêtes seront redirigées vers la page **index.php**.

PHP

Code PHP :

```
<?php
    //Analyse de l'URL avec parse_url() et retourne ses composants
    $url = parse_url($_SERVER['REQUEST_URI']);
    //test soit l'url a une route sinon on renvoi à la racine
    $path = isset($url['path']) ? $url['path'] : '/';
    /*--------------------------ROUTER -----------------------------*/
    //test de la valeur $path dans l'URL et import de la ressource
    switch($path){
        //route /projet/test -> ./test.php
        case $path === "/projet/test" :
            include './test.php';
            break ;
        //route /projet/addUser -> ./controller/controller_add_user.php
        case $path === "/projet/addUser ":
            include './controller/controller_add_article.php';
            break ;
    }
?>
```

NB : Attention tous les liens (HTML, CSS, JS) devront faire référence à la page **index.php**.

Exemple : page ***view_add_user.php*** (dans le dossier ***view***) si je souhaite lier un fichier css le chemin sera :

```
<link rel="stylesheet" href="./asset/css/style.css">
```

20.4 Script de génération de base de projet MVC Object :

Vous trouverez ci-dessous un repository **github** dans lequel je vous met à disposition un **script bash** qui va automatiser la création d'une **base de projet**. Un tutoriel est disponible en plus du script.

https://github.com/mithridatem/scriptmvc

NB : Ce script est à utiliser pour vous simplifier la création du projet. Par contre je vous conseille de créer vos premiers projets à la main.

PHP

21 Héritage (Objet) :

L'un des intérêts de la **POO** est de rendre notre code modulable, réutilisable (nous l'avons vu avec les **classes** et le modèle **MVC**) et de nous permettre de mettre à jour et d'étendre notre code facilement.

L'héritage va nous permettre **d'étendre** une **classe** (*récupérer tout ce qui est en **public** ou **protected** dans la **classe parente***) par une autre **classe**. Les classes **étendues** pourront bénéficier d'**attributs**, **constructeurs**, **méthodes propres**,

Pour étendre une classe nous utiliserons le mot clé **extends**.

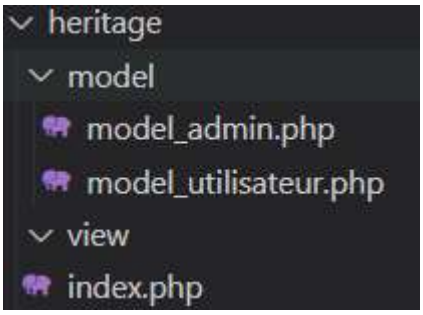
21.1 Exemple :

```php
< ?php
class Admin extends Utilisateur{
}
?>
```

Notre classe **Admin** va étendre la classe **Utilisateur**, elle va pouvoir utiliser toutes les **méthodes** et **attributs** de la classe **Utilisateur** qui ne sont pas en **private**.

NB : Attention Pour qu'une classe puisse **hériter** d'une autre celle-ci doit être **existante** dans le projet et être **include**, une classe ne peut hériter que d'une seule classe à la fois.

Structure des classes :



PHP

Classe Utilisateur :

```php
<?php
    class Utilisateur{
        //attributs
        private $name;
        private $firstName;
        //constructeur
        public function __construct($name, $first){
            $this->name = $name;
            $this->firstName = $first;
        }
        //getter and setter
        public function getName():string{
            return $this->name;
        }
        public function getFirstName():string{
            return $this->firstName;
        }
        public function setName($name):void{
            $this->name = $name;
        }
        public function setFirstName($first):void{
            $this->firstName = $first;
        }
        //Méthodes
        public function showUser():void{
            echo 'Nom : '.$this->getName().'
            Prénom : '.$this->getFirstName().'';
        }
    }
?>
```

PHP

21.2 Appel des classes et méthodes :

```php
<?php
    //imports
    include './model/model_utilisateur.php';
    include './model/model_admin.php';

    //instances des objets :
    $util = new Utilisateur("Dupond", "Marc");
    //admin utilise le constructeur d'Utilisateur
    $admin = new Admin("Durand", "Marie");

    echo '<p>Utilisateur : '.$util->getName().'</p>';
    //admin utilise le getter public d'utilisateur
    echo '<p>Admin : '.$admin->getName().'</p>';
?>
```

NB : notre classe **Admin** utilise le **constructeur** ainsi que **les getters setters** de la classe **Utilisateur**.

21.2 Redéfinition d'une méthode dans la classe enfant :

Si nous essayons de redéfinir dans la classe **Admin** le **getter getName** comme ci-dessous et que nous l'appelons dans la page **index** (code précédent), nous aurons une erreur :

Car le **paramètre name** est en **Private** (dans la classe **Utilisateur**) et donc **inaccessible** en dehors de la classe **Utilisateur**.

```php
<?php
    class Admin extends utilisateur{
        //getter and setter
        //redéfinition dans la classe
        public function getName():string{
            return $this->name;
        }
    }
?>
```

PHP

22 Etendu des classes Héritage (Objet) :

Pour pouvoir étendre des classes et redéfinir leurs attributs, méthodes dans la classe étendu nous allons devoir passer les **attributs** et ou **méthodes** à redéfinir en **protected** dans la classe parente (classe **Utilisateur**).

22.1 Correction classe Utilisateur :

```php
<?php
    class Utilisateur{
        //attributs
        protected $name;
        protected $firstName;
?>
```

Le code de la section précédente (**getNom** dans la classe **Admin**) est alors **utilisable**. C'est le **getter getNom** de la classe **Admin** qui est appelé (il peut résoudre le **$this->name** car l'attribut est en **protected**).

22.2 Surcharge de méthode :

Redéfinition (surcharge) de la méthode getNom dans la classe Admin :

```php
//redéfinition dans la classe
public function getName():string{
      //retourne le nom en Majuscule
      return strtoupper($this->name);
}
```

NB : On modifie dans la classe **Admin** (surcharge) le fonctionnement de la méthode **getName** qui est déjà définie dans la classe **Utilisateur**.

22.3 Méthodes de la classe Admin qui utilisent des attributs de la classe Utilisateur :

```php
<?php
    class Admin extends Utilisateur{
        //attributs
        protected $activate;
        //getter and setter
        //redéfinition dans la classe
        public function getName():string{
            //retourne le nom en Majuscule
            return strtoupper($this->name);
        }
        //méthodes
        //activer un compte Utilisateur (objet)
        public function setActiveUser($obj):void{
```

PHP

```
            //ajouter le nom dans la liste activate(tableau)
            $this->activate[] = $obj;
        }
        //affichage de la liste des comptes activés
        public function getActiveUser():void{
            echo 'Liste des comptes Utilisateurs activés :';
            //boucle affichage des comptes(objets)
            foreach($this->activate as $value){
                echo '<p>'.$value->name.' '.$value->firstName.',</p>';
            }
        }
    }
?>
```

22.4 Appel dans index.php des méthodes de la classe Admin (setActivateUser et getActivateUser) :

```
<?php
    //imports
    include './model/model_utilisateur.php';
    include './model/model_admin.php';
    //instances des objets :
    $util = new Utilisateur("Dupond", "Marc");
    $util2 = new Utilisateur("Albert", "Patricia");
    //admin utilise le constructeur d'Utilisateur
    $admin = new Admin("Durand", "Marie");
    echo '<p>Utilisateur : '.$util->getName().'</p>';
    echo '<p>Utilisateur : '.$util2->getName().'</p>';
    //admin utilise le getter public d'utilisateur
    echo '<p>Admin : '.$admin->getName().'</p>';
    //appel de la méthode setActiveUser
    $admin->setActiveUser($util);
    $admin->setActiveUser($util2);
    //affichage de la liste des utilisateurs activés
    $admin->getActiveUser();
?>
```

Repository github héritage :

https://github.com/mithridatem/extends.git