

PHP et MySQL

Table des matières

I.	Introduction.....	5
I.1.	Définition et rôle du PHP.....	5
I.2.	Sites statiques et sites dynamiques.....	5
I.3.	Définition et rôle du MySQL.....	6
II.	Langages client side et server side	7
III.	Comment écrire un script PHP.....	8
III.1.	La balise PHP	8
III.2.	Où écrire le code PHP ?.....	8
III.3.	Affichez du texte.....	10
III.3.1.	L'instruction echo	10
III.3.2.	L'instruction print	11
IV.	Les variables en PHP	12
IV.1.	Qu'est-ce qu'une variable	12
IV.2.	déclaration des variables en PHP	12
V.	Concaténation.....	15
V.1.	Avec des guillemets simples	15
V.2.	Avec des guillemets doubles.....	15
V.3.	Bonne Pratique.....	15
VI.	Les types de données PHP.....	17
VI.1.	Le type string.....	17
VI.3.	Le type float.....	18
VI.4.	Le type bool	18
VI.5.	Le type Null.....	18
VII.	Les calculs en PHP.....	19
VII.1.	Les opérateurs arithmétiques.....	19
VIII.	Les structures conditionnelles	22
VIII.1.	Les opérateurs de comparaison.....	22
VIII.2.	La structure if ... else	23
VIII.2.1.	Le cas des booléens	24
VIII.3.	Les conditions composées.....	25
VIII.3.1.	Les opérateurs logiques	25
VIII.4.	La structure switch	26
IX.	Les boucles.....	28
IX.1.	Les opérateurs d'incrémentation et de décrémentation.....	29
IX.2.	La boucle while.....	30
IX.3.	La boucle do...while	31
IX.4.	La boucle for	32
X.	Les tableaux :	33

X.1.	Les tableaux indexés	33
X.1.2.	Afficher les valeurs d'un tableau indexé	34
X.1.2.1.	Avec la boucle for	34
X.2.	Les tableaux associatifs	35
X.2.1.	Créer un tableau associatif	35
X.2.2.	Récupérer et afficher les valeurs d'un tableau associatif	36
X.3.	Les tableaux multidimensionnels	37
X.3.1.	Créer un tableau multidimensionnel	37
X.3.2.	Récupérer ou afficher une valeur en particulier d'un tableau multidimensionnel	37
X.3.3.	Parcourir et afficher les valeurs d'un tableau multidimensionnel	38
X.4.	Afficher rapidement la structure d'un tableau en PHP	38
X.5.	Manipuler les tableaux avec les fonctions prédéfinies	40
XI.	Les fonctions	40
XI.1.	Définition des fonctions et fonctions internes ou prédéfinies	40
XI.2.	Les fonctions définies par l'utilisateur	41
XI.3.	Les paramètres et arguments des fonctions	41
XI.4.	Les valeurs de retour d'une fonction	44
XII.	La portée des variables	44
XII.2.	Accéder à une variable définie localement depuis l'espace global	46
XIII.	Les constantes	47
XIII.1.	Définir une constante	47
XIII.2.	Les constantes prédéfinies et les constantes magiques	48
XIV.	Inclure des fichiers dans d'autres	48
XIV.1.	Présentation de include et de require	48
XIV.2.	Les différences entre include, include_oncé, require et require_oncé	49
XV.	Transmettre des données de page en page	50
XV.1.	Envoyez des paramètres dans l'URL	50
XV.1.1.	Créez un lien avec des paramètres	50
XV.1.2.	Créez un formulaire avec la méthode HTTP GET	51
XV.2.	Récupérez les paramètres en PHP	52
XV.3.	Envoyez des données par la méthode POST	54
XV.4.1.	Paramétrez le formulaire d'envoi de fichier	56
XV.4.2.	Traiter l'envoi du fichier	56
XV.4.2.1.	Tester si le fichier a bien été envoyé	58
XV.4.2.2.	Vérifier la taille du fichier	58
XV.4.2.3.	Vérifier l'extension du fichier	58
XV.4.2.4.	Valider l'upload du fichier	59
XVI.	La variable superglobale \$_SERVER	60
XVII.	La variable superglobale \$_REQUEST	60
XIX.	Les sessions	61
XIX.1.	création d'une session	61

XIX.2.	création de variables pour la session.....	61
XIX.3.	suppression de la session.....	62
XX.	Les cookies.....	63
XX.1.1.	fonctionnement d'un cookie.....	63
XX.1.2.	Écrire un cookie.....	63
XX.1.3.	Sécuriser un cookie.....	64
XX.1.4.	Affichez et récupérez un cookie	64
XX.1.5.	Modifiez un cookie existant.....	65
XX.1.6.	Supprimer un cookie :	65
XXI.	Travailler avec une base de données.....	66
XXI.1.	Découverte des bases de données.....	66
XXI.1.1.	Le langage SQL.....	66
XXI.1.2.	Comment PHP fait le lien entre vous et MySQL.....	66
XXI.2.	Structurez votre base de données.....	67
XXI.2.1.	Comment sont enregistrées les données.....	69
XXII.	Mettez en place une base de données avec phpMyAdmin.....	70
XXII.1.	Créer une table.....	70
XXII.1.1.	Les types de champs MySQL.....	71
XXII.1.2.	Les clés primaires.....	71
XXII.2.	Modifier une table	72
XXII.3.	Importez et exportez des données.....	73
XXIII.	Accédez aux données en PHP avec PDO	75
XXIII.1.	Connectez-vous à la base de données en PHP.....	75
XXIII.1.1.	Connectez PHP à MySQL avec PDO	75
XXIII.1.2.	Testez la présence d'erreurs.....	76
XXIII.2.	Effectuez une première requête SQL.....	77
XXIII.2.1.	Construisez votre première requête SQL.....	77
XXIII.2.2.	Affichez le résultat d'une requête SQL.....	77
XXIII.3.1.	Filtrer les résultats.....	79
XXIII.3.2.	Trier les résultats.....	79
XXIII.3.3.	Afficher certains résultats.....	80
XXIII.4.	Utiliser des variables dans les requêtes	80
XXIII.4.1.	Utiliser des marqueurs.....	80
XXIII.4.2.	Lier les variables avec la méthode bindValue().....	81
XXIV.	Ajouter, modifier et supprimer des données.....	82
XXIV.1.	Ajouter des données.....	82
XXIV.2.	Modifier des données	83
XXIV.3.	Supprimez des données	85
XXV.	Les fonctions SQL.....	86
XXV.1.	Les fonctions MIN() et MAX().....	86
XXV.2.	La fonction COUNT()	86

XXV.3.	La fonction AVG().....	86
XXV.4.	La fonction SUM().....	86
XXV.5.1.	Fonction REPLACE dans un UPDATE	87
XXVI.	Les fonctions d'agrégation et les critères de sélection.....	89
XXVI.1.	L'instruction GROUP BY.....	89
XXVI.2.	La clause SQL HAVING.....	89
XXVII.	Travailler avec plusieurs tables	90
XXVII.1.	Les clés étrangères.....	90
XXVII.1.2.	Qu'est-ce qu'une clé étrangère ?	91
XXVII.1.2.1.	La clé étrangère met en relation deux tables :	91
XXVII.1.2.2.	Clé étrangère et intégrité référentielle de la base de données	92
XXVII.1.3.	Définir une clé étrangère avec phpMyAdmin.....	92
XXVII.1.4.	Définir une clé étrangère en SQL.....	93
XXVII.1.5.	Limitation et contraintes de la clé étrangère.....	94
XXVIII.	Les jointures.....	94
XXVIII.1.	Les différents types de jointures.....	94
XXVIII.2.	L'INNER JOIN	94
XXVIII.3.	Les Alias	95
	Voici la requête sans les AS :	95
XXVIII.4.	Le LEFT JOIN.....	96
XXVIII.5.	Le RIGHT JOIN.....	96
XXIX.	L'opérateur SQL UNION.....	97

I. Introduction

I.1. Définition et rôle du PHP

Le terme **PHP** est l'acronyme récursif de « **PHP : Hypertext Preprocessor** ». Le premier « P » de PHP est en effet lui-même l'abréviation de « PHP ».

Le PHP a été créé en 1994. C'est un langage qui permet de créer des pages qui vont être générées dynamiquement. En d'autres mots, grâce au PHP, nous allons pouvoir afficher des contenus différents sur une même page en fonction de certaines variables : l'heure de la journée, le fait que l'utilisateur soit connu et connecté ou pas, etc.

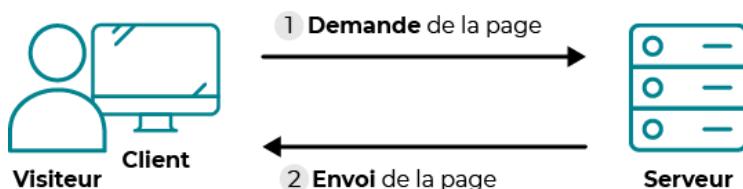
Le PHP va s'exécuter côté serveur. Il fait ainsi partie des langages qu'on nomme « **server side** » en opposition aux langages « **client side** » qui s'exécutent côté client.

I.2. Sites statiques et sites dynamiques

Les langages de programmation axés web peuvent être catégorisés selon deux grands types de classement :

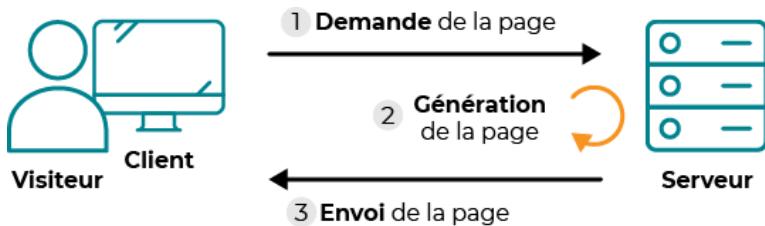
- Langages statiques VS langages dynamiques ;
- Langages avec exécution côté client VS langages avec exécution côté serveur.

Les sites dits statiques se caractérisent par le fait qu'ils sont... statiques : ils ne possèdent ni interaction, ni la capacité de s'adapter aux visiteurs. Le code des différentes pages ne va pas changer en fonction d'un utilisateur ou d'une autre variable.



Un site créé uniquement en HTML et en CSS par exemple sera toujours statique.

Les sites dynamiques, en revanche, vont pouvoir fournir des pages différentes pour chaque visiteur ou selon différentes contraintes et vont nous permettre d'interagir avec l'utilisateur en lui permettant de nous envoyer des données par exemple.



Dans le cas du couple PHP / MySQL :

- PHP va être utile pour tout ce qui est calcul / traitement des données,
- MySQL va nous servir à gérer nos bases de données.

A retenir :

- Un site web créé uniquement avec des langages qui s'exécutent côté client sera statique
- Un site web créé avec des langages qui s'exécutent côté client et des langages qui s'exécutent côté serveur sera généralement dynamique.

I.3. Définition et rôle du MySQL

MySQL est un **Système de Gestion de Bases de Données** relationnelles (SGBD). Une base de données est un ensemble structuré de données. Les données vont pouvoir être des informations clients (nom, adresse, mot de passe, etc.), la liste des commentaires de notre blog, le texte de nos articles, etc.

On ne peut pas directement interagir avec les bases de données car les données sont stockées d'une manière illisible pour un humain. Pour manipuler les données stockées dans les bases de données, nous allons devoir utiliser un langage de bases de données.

Le langage de bases de données utilisé par MySQL est le **SQL** (**Structured Query Language**, *en français Langage de Requêtes Structurées*).

Les avantages du MySQL sont sa simplicité d'utilisation, sa fiabilité, ses performances et qu'on va pouvoir l'utiliser conjointement avec PHP.

On ne peut pas créer ni manipuler de bases de données sans système de gestion de bases de données.

II. Langages client side et server side

Un navigateur (côté client) et un serveur (côté serveur) ne vont pouvoir chacun effectuer que certaines opérations et lire certains langages.

Les navigateurs ne sont capables de comprendre et de n'exécuter que du code HTML, CSS et JavaScript. Un navigateur est ainsi incapable de comprendre du code PHP.

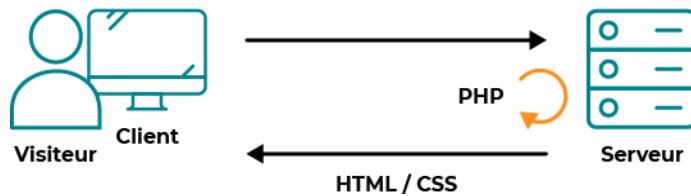
Lorsqu'un navigateur demande à un serveur de lui servir une page, le serveur va donc se charger d'exécuter tout code qui ne serait pas compréhensible par le navigateur.

Une fois ces opérations effectuées, le serveur va renvoyer le résultat sous forme de code compréhensible par le navigateur, c'est-à-dire principalement en HTML. Le navigateur va alors afficher la page au visiteur. Cette page va être unique puisqu'elle a été générée par le serveur pour un utilisateur spécifiquement et en tenant compte de données particulières.

Si en revanche la page demandée par le visiteur ne contient aucun code nécessitant l'intervention du serveur, alors le serveur va la renvoyer telle quelle au navigateur qui va l'afficher au visiteur.

Notez bien ici que les opérations réalisées côté serveur vont être transparentes pour le visiteur et que celui-ci ne va jamais avoir accès ni pouvoir voir le code exécuté côté serveur tout simplement car une fois les opérations terminées, le serveur ne va renvoyer que du code compréhensible par le navigateur.

C'est la raison pour laquelle lorsque vous analyser le code d'un page, vous ne trouverez jamais d'instructions PHP mais seulement du code HTML, CSS et JavaScript qui sont des langages qui s'exécutent côté client.



III. Comment écrire un script PHP

III.1. La balise PHP

Pour utiliser du PHP, on va une balise spécifique :

- Elle commence par `<?php`
- Et se termine par `?>`

C'est à l'intérieur que l'on mettra du code PHP.

Voici une balise PHP vide :

```
<?php ?>
```

À l'intérieur, on écrira du code source PHP :

```
<?php // Ceci est un commentaire sur une ligne ?>
```

La plupart du temps, le code PHP fera plusieurs lignes :

```
<?php
/* Ceci est
un commentaire
multiligne */
?>
```

Vous remarquez ici les deux syntaxes des **commentaires PHP**.

III.2. Où écrire le code PHP ?

Nous allons pouvoir écrire nos scripts PHP soit dans des fichiers dédiés, c'est-à-dire des fichiers qui ne vont contenir que du PHP, soit intégrer le PHP au sein de nos fichiers HTML.

Les fichiers qui contiennent du PHP vont devoir être enregistrés avec l'extension **.php**. Dans le cas où on intègre du code PHP dans un fichier HTML, il faudra également changer son extension en **.php**.

Le code PHP peut s'insérer n'importe où dans le code HTML.

Exemples :

Dans le <body> :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ceci est une page de test avec des balises PHP</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h2>Page de test</h2>

    <p>
      Cette page contient du code HTML avec des balises PHP.<br />
      <?php /* Insérer du code PHP ici */ ?>
      Voici quelques petits tests :
    </p>

    <?php
    /* Encore du PHP
    Toujours du PHP */
    ?>
  </body>
</html>
```

Dans le <head> :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ceci est une page de test <?php /* Code PHP */ ?></title>
    <meta charset="utf-8" />
  </head>
```

Au milieu d'une balise HTML :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ceci est une page de test</title>
    <meta <?php /* Code PHP */ ?> charset="utf-8" />
  </head>
```

Cela fonctionne tout simplement parce que le PHP génère du code HTML. On peut donc insérer du PHP absolument n'importe où dans le code HTML, et même avant le DOCTYPE.

III.3. Affichez du texte

Tout langage de programmation contient ce qu'on appelle des **instructions**. On en écrit une par ligne en général, et en PHP elles se terminent toutes par **un point-virgule**. Une instruction commande à l'ordinateur d'effectuer une action précise.

III.3.1. L'instruction echo

La première instruction que nous allons découvrir permet d'insérer du texte dans la page web. Il s'agit de l'instruction **echo** :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ceci est une page de test avec des balises PHP</title>
        <meta charset="utf-8" />
    </head>
    <body>
        <h1>Page de test</h1>

        <?php
            // Affiche "Hello World" avec un retour à la ligne
            echo 'Hello World <br>';

            /* Affiche
                "Bonjour le Monde"
            */
            echo "Bonjour le Monde <br>";
        ?>

        <p>Un paragraphe de texte</p>

        <!--Instruction echo avec des parenthèses et l'utilisation du caractère
d'échappement \ -->
        <?php echo ("Cette ligne utilise le \"caractère d'échappement\" en
PHP."); ?>

    </body>
</html>
```

Remarques sur l'instruction **echo** :

- On peut utiliser les guillemets simples ' ou doubles "
- Si l'on veut afficher des guillemets on les fait précédé du caractère d'échappement \ (antislash)
- On peut y placer des balises HTML qui seront interprétées correctement
- Elle peut s'écrire avec des parenthèses comme une fonction, mais ce n'en est pas une, c'est une structure de langage

Exercice :

(Dans le répertoire www de Wamp, créer un dossier PHP avec deux sous dossiers Cours et Exercices PHP, placez votre fichier .php dans Cours PHP et démarrer Wamp. Accéder à votre fichier par l'adresse <http://localhost/PHP/Cours/exercice1.php>) :

Ecrivez le code PHP permettant d'afficher ceci, et affichez le dans votre navigateur:

Page de test

Hello World !

Bienvenue sur cette "**Page de test**" en PHP.

III.3.2. L'instruction print

L'instruction **print** produit un affichage identique à l'instruction **echo**, mais comporte quelques différences :

- **echo** ne possède pas de valeur de retour à la différence de **print** qui va toujours retourner **1**,
- on va pouvoir passer plusieurs valeurs à **echo** tandis qu'on ne va pouvoir en passer qu'une à **print**, dans ce cas les différentes valeurs sont séparées par des virgules,
- **echo** s'exécute un peu plus rapidement que **print**.

Exemple :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Document</title>
    </head>
    <body>
        <h1>Titre principal</h1>
        <?php
            echo '<h2>Première instruction PHP avec echo</h2>';
            echo 'Bonjour à tous !<br>';
            echo 29;
            echo "<br>J'ai ", 29, " ans. <br>";

            print '<h2>Première instruction PHP avec print</h2>';
            print 'Bonjour à tous !<br>';
            print 29;
        ?>
    </body>
</html>
```

Produit le résultat suivant :

Titre principal

Première instruction PHP avec echo

Bonjour à tous !

29

J'ai 29 ans.

Première instruction PHP avec print

Bonjour à tous !

29

Remarque : Dans cet exemple, on s'est passé du caractère d'échappement pour afficher **J'ai** simplement en utilisant des guillemets doubles pour l'instruction **echo**, on peut aussi faire l'inverse, si l'on veut afficher des guillemets doubles, on utilise les guillemets simples pour l'instruction **echo**.

IV. Les variables en PHP

IV.1. Qu'est-ce qu'une variable

Une variable est un conteneur servant à stocker des informations de manière temporaire, comme une chaîne de caractères (un texte) ou un nombre par exemple.

Le propre d'une variable est de pouvoir varier, c'est-à-dire de pouvoir stocker différentes valeurs au fil du temps.

En PHP, les variables ne servent à stocker une information que temporairement

IV.2. déclaration des variables en PHP

Lorsqu'on crée une variable en PHP, on dit également qu'on « déclare » une variable.

On va pouvoir choisir le nom qu'on souhaite donner à chacune de nos variables. Cependant, il y a quelques règles à respecter et à connaître lors de la déclaration d'une nouvelle variable :

- Toute variable en PHP doit commencer par le signe \$ qui sera suivi du nom de la variable ;
- Le nom d'une variable doit obligatoirement commencer par une lettre ou un underscore (_) et ne doit pas commencer par un chiffre ;
- Le nom d'une variable ne doit contenir que des lettres, des chiffres et des underscores mais pas de caractères spéciaux ;
- Le nom d'une variable ne doit pas contenir d'espace.

De plus, notez que le nom des variables est sensible à la casse en PHP. Cela signifie que l'usage de majuscules ou de minuscules va créer des variables différentes. Par exemple, les variables \$texte, \$TEXTE et \$tEXTe vont être des variables différentes.

Remarque : Comme on ne peut pas mettre d'espace dans le nom d'une variable, on respectera la convention dite **camelCase**.

Par exemple : \$maVariable

Exemple :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Page de test</title>
    </head>
    <body>
        <h1>Titre principal</h1>
        <?php
            $prenom = "Matthieu";
            $age = 30;
        ?>
    </body>
</html>
```

Dans le script ci-dessus, nous déclarons donc nos deux variables \$prenom et \$age.

On assigne ou on affecte la valeur Matthieu à la variable \$prenom. Cette variable va donc stocker la valeur Matthieu.

De la même façon, on assigne la valeur 35 à la variable \$age.

Notez qu'il va falloir utiliser des guillemets ou des apostrophes pour stocker une chaîne de caractères dans une variable. En revanche, nous n'en utiliserons pas pour assigner un nombre à une variable.

Ici le signe = est l'opérateur d'affectation, et n'a absolument rien à voir avec l'égal mathématique.

IV.3. Afficher une variable

De la même manière que l'on peut afficher du texte avec `echo`, on va aussi s'en servir pour afficher la valeur d'une variable :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Page de test</title>
    </head>
    <body>
        <h1>Titre principal</h1>
        <?php
            $prenom = "Matthieu";
            $age = 30;

            echo $prenom;
            echo "<br>";
            echo $age;
            echo " ans";
        ?>
    </body>
</html>
```

Comme vous le voyez, il suffit d'écrire le nom de la variable que vous voulez afficher après `echo` et **sans guillemets**.

Exercice :

Créez un fichier PHP avec ce code source en mettant votre nom et votre age

V. Concaténation

La méthode dépendra de si l'on utilise des guillemets simples ou des guillemets doubles avec `echo` :

V.1. Avec des guillemets simples :

On utilise l'opérateur de concaténation `.` (le point) :

```
<?php
    $prenom = "Matthieu";
    $age = 30;
    echo 'Je m\'appelle ' . $prenom . ' et j\'ai ' . $age . " ans."
?>
```

V.2. Avec des guillemets doubles

On peut insérer directement les variables et elles seront interprétées par PHP :

```
<?php
    $prenom = "Matthieu";
    $age = 30;
    echo "Je m'appelle $prenom et j'ai $age ans."
?>
```

Ce qui affiche le même résultat.

V.3. Bonne Pratique

Bien que la première méthode ait l'air bien plus compliqué, c'est cette méthode qu'utilisent la plupart des programmeurs expérimentés en PHP. En effet, le code est plus lisible, et on repère bien les variables, alors que dans le 2^e cas, elles sont comme « noyée » dans le texte.

Il est donc conseillé de toujours utiliser l'opérateur de concaténation lorsqu'on souhaite mettre bout-à-bout plusieurs chaînes de caractères avec des variables, et ceci qu'on utilise des guillemets simples ou doubles.

Exercice :

Ecrivez un script PHP qui dans un premier temps stocke tous les prénoms, le noms et les âges de chaque étudiants du groupe dans des variables, puis affiche :

Liste des étudiants :

- Jean DUPONT à 35 ans,
- Aline LAVOISIER a 45 ans,
- Etc.
-

Exercice :

Reprenez le script précédent en rajoutant la ville, et en mettant en gras l'affichage des variables :

Liste des étudiants :

- **Jean DUPONT** est âgé de **35** ans et habite à **METZ**,
- **Aline LAVOISIER** est âgée de **45** ans et habite à **NANCY**,
- Etc.

VI. Les types de données PHP

Les variables PHP vont pouvoir stocker différents types de valeurs, comme du texte ou un nombre par exemple.

En PHP, contrairement à d'autres langages de programmation, nous n'avons pas besoin de préciser à priori le type de valeur qu'une variable va pouvoir stocker. Le PHP va en effet automatiquement détecter quel est le type de la valeur stockée dans telle ou telle variable, et nous allons ensuite pouvoir faire différentes opérations selon le type de la variable.

Une conséquence directe de cela est qu'on va pouvoir stocker différents types de valeurs dans une variable au fil du temps sans se préoccuper d'une quelconque compatibilité.

Les variables en PHP vont pouvoir stocker 8 grands types de données différents :

- **Les chaînes de caractères** (`string`) : c'est le nom informatique qu'on donne au texte,
- **Les nombres entiers** (`int`) : ce sont les nombres du type 1, 2, 3, 4, etc. On compte aussi parmi eux les entiers relatifs : -1, -2, -3...,
- **Les nombres décimaux** (`float`) : ce sont les nombres à virgule, comme 14,738. Attention, les nombres doivent être écrits avec un **point** au lieu de la virgule (c'est la notation anglaise),
- **Les booléens** (`bool`) : c'est un type très important qui permet de stocker soit vrai soit faux,
- **Le type null** (`NULL`) : ce n'est pas vraiment un type de données, mais plutôt l'**absence** de type,
- Le type « tableau » (`array`),
- Le type « objet » (`object`),
- Le type « ressource » (`resource`).

Nous allons laisser ces trois derniers de côté pour le moment.

VI.1. Le type string

Ce type permet de stocker du texte.

Pour cela, vous devez entourer votre texte de :

- Guillemets doubles "" ;
- Ou de guillemets simples "

Exemple :

```
<?php
    $fullname = "Jacques DUCHEMIN";
    $email = 'jacques.duchemin@gmail.com';
?>
```

VI.2. Le type int

Il permet de stocker des nombres entiers positifs ou négatifs.

Il suffit d'écrire le nombre que vous voulez stocker, sans guillemets :

```
<?php  
    $userAge = 17;  
    $nb = -3;  
?>
```

VI.3. Le type float

Il permet de stocker des nombres à virgule positifs ou négatifs.

Vous devez écrire votre nombre avec un **point** au lieu d'une virgule.

```
<?php  
    $price = 57.3;  
?>
```

VI.4. Le type bool

Pour dire si une variable vaut vrai ou faux, vous devez écrire le mot **true** ou **false** sans guillemets autour (ce n'est pas une chaîne de caractères !) :

```
<?php  
    $isAuthor = true;  
    $isAdmin = false;  
?>
```

VI.5. Le type Null

Le type de données **Null** est un type un peu particulier puisqu'il correspond à l'absence de valeur et sert donc à représenter des variables vides en PHP.

Ce type de valeur ne contient qu'une seule valeur : la valeur **NULL** qui correspond elle-même à l'absence de valeur.

Notez que si vous déclarez une nouvelle variable sans lui affecter de valeur (ce qui est déconseillé de manière générale), cette variable sera automatiquement de type **Null**.

On peut aussi l'initialiser avec le mot clé **NULL**.

Exemples :

```
<?php  
    $null = NULL;  
    $null2;  
?>
```

VII. Les calculs en PHP

VII.1. Les opérateurs arithmétiques

Les opérateurs arithmétiques vont nous permettre d'effectuer toutes sortes d'opérations mathématiques entre les valeurs contenues dans différentes variables lorsque ces valeurs sont des nombres.

En PHP, nous allons pouvoir utiliser les opérateurs arithmétiques suivants :

Opérateur	Nom de l'opération associée
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste d'une division euclidienne)
**	Exponentielle (élévation à la puissance d'un nombre par un autre)

Quelques précisions :

Le modulo

Le modulo correspond au reste entier d'une division euclidienne. Par exemple, lorsqu'on divise 5 par 3, le résultat est 1 et il reste 2 dans le cas d'une division euclidienne. Le reste, 2, correspond justement au modulo.

L'exponentielle

L'exponentielle correspond à l'élévation à la puissance d'un nombre par un autre nombre. La puissance d'un nombre est le résultat d'une multiplication répétée de ce nombre par lui-même. Par exemple, lorsqu'on souhaite calculer 2 à la puissance de 3 (qu'on appelle également « 2 exposant 3 »), on cherche en fait le résultat de 2 multiplié 3 fois par lui-même c'est-à-dire $2 \times 2 \times 2 = 8$.

Exemples :

```
<?php
    $number = 2 + 4; // $number prend la valeur 6
    $number = 5 - 1; // $number prend la valeur 4
    $number = 3 * 5; // $number prend la valeur 15
    $number = 10 / 2; // $number prend la valeur 5

    // Allez on rajoute un peu de difficulté
    $number = 3 * 5 + 1; // $number prend la valeur 16
    $number = (1 + 2) * 2; // $number prend la valeur 6
    $number = 5 % 3; // $number prend la valeur 2
    $number = 2 ** 3; // $number prend la valeur 8
?>
```

Règles de calcul

Concernant les règles de calcul, c'est-à-dire l'ordre de priorité des opérations, celui-ci va être le même qu'en mathématiques : l élévation à la puissance va être prioritaire sur les autres opérations, tandis que la multiplication, la division et le modulo vont avoir le même ordre de priorité et être prioritaires sur l addition et la soustraction qui ont également le même niveau de priorité.

Si deux opérateurs ont le même ordre de priorité, alors c'est leur sens d'association qui va décider du résultat. Pour les opérateurs arithmétiques, le sens d'association correspond à l'ordre de leur écriture à l'exception de l élévation à la puissance qui sera calculée en partant de la fin.

Ainsi, si on écrit `$x = 1 - 2 - 3`, la variable `$x` va stocker la valeur -4 (les opérations se font de gauche à droite). En revanche, si on écrit `$x = 2 ** 3 ** 2`, la variable `$x` stockera 512 qui correspond à 2 puissance 9 puisqu'on va commencer par calculer $3^{**} 2 = 9$ dans ce cas.

Comme en mathématique, on peut définir soi-même les priorités à l'aide de parenthèses.

VII.2. Les opérateurs d'affectation et opérateurs combinés

Les opérateurs d'affectation vont nous permettre, comme leur nom l'indique, d'affecter une certaine valeur à une variable.

Nous connaissons déjà bien l'opérateur d'affectation le plus utilisé qui est le signe `=`.

Cependant, vous devez également savoir qu'il existe également des opérateurs combinés notamment pour les opérateurs arithmétiques et l'opérateur de concaténation et qui sont les suivants :

Opérateur	Définition
<code>.=</code>	Concatène puis affecte le résultat
<code>+=</code>	Additionne puis affecte le résultat
<code>-=</code>	Soustrait puis affecte le résultat
<code>*=</code>	Multiplie puis affecte le résultat
<code>/=</code>	Divise puis affecte le résultat
<code>%=</code>	Calcule le modulo puis affecte le résultat
<code>**=</code>	Élève à la puissance puis affecte le résultat

Exemples :

```
<?php
    $a = "Bonjour";
    $a .= " le monde"; // $a stocke "Bonjour le monde"
    echo '$a stocke : ' . $a . '<br>';

    $x = 5;
    $x -= 3; // $x stocke maintenant 2
    echo '$x stocke : ' . $x . '<br>';

    $y = 3;
    $y **= $x; // $y stocke 3^2 soit 9
    echo '$y stocke : ' . $y . '<br>';

    $b = 5;
    $b *= 4 + 2; // $b stocke 30
    echo '$b stocke : ' . '<br>';

    $y = 2;
    $z = 3;
    $y /= $z -= 2; // $z stocke 1 et $y stocke 2
    echo '$y stocke : ' . $y . ' et $z stocke : ' . $z . '<br>';
?>
```

Remarque : Tous les opérateurs d'affectation ont une priorité de calcul égale mais qui est inférieure à celle des opérateurs arithmétiques ou de concaténation.

Lorsque des opérateurs ont des ordres de priorité égaux, c'est le sens d'association de ceux-ci qui va décider du résultat. Pour les opérateurs d'affectation, l'association se fait par la droite.

VIII. Les structures conditionnelles

VIII.1. Les opérateurs de comparaison

Pour comparer des valeurs, nous allons devoir utiliser des opérateurs de comparaison :

Opérateur	Définition
<code>==</code>	Permet de tester l'égalité sur les valeurs
<code>==</code>	Permet de tester l'égalité en termes de valeurs et de types
<code>!=</code>	Permet de tester la différence en valeurs
<code><></code>	Permet également de tester la différence en valeurs
<code>!==</code>	Permet de tester la différence en valeurs ou en types
<code><</code>	Permet de tester si une valeur est strictement inférieure à une autre
<code>></code>	Permet de tester si une valeur est strictement supérieure à une autre
<code><=</code>	Permet de tester si une valeur est inférieure ou égale à une autre
<code>>=</code>	Permet de tester si une valeur est supérieure ou égale à une autre

VIII.2. La structure if ... else

Voici ce qu'on doit écrire, dans l'ordre, pour utiliser cette condition.

- Pour introduire une condition, on utilise le mot **if** qui signifie « si », en anglais.
- On ajoute à la suite entre parenthèses la condition en elle-même (vous allez voir que vous pouvez inventer une infinité de conditions).
- Enfin, on ouvre des accolades à l'intérieur desquelles on placera les instructions à exécuter si la condition est remplie.

Exemple :

```
<?php
    $isEnabled = true; // La condition d'accès

    if ($isEnabled == true) {
        echo "Vous êtes autorisé(e) à accéder au site !";
    }
?>
```

Ici, on demande à PHP :

Si la variable **\$isEnabled** est vraie, affiche « Vous êtes autorisé(e) à accéder au site ! ».

Ce qui compte ici, c'est qu'il y a deux possibilités :

- Soit la condition est remplie et alors on affiche quelque chose.
- Sinon, on saute les instructions entre accolades, on ne fait rien.

On peut améliorer notre exemple :

```
<?php
    $isEnabled = true; // La condition d'accès

    if ($isEnabled == true) {
        echo "Vous êtes autorisé(e) à accéder au site !";
    }
    else {
        echo "Accès refusé !! ";
    }
?>
```

Essayez ce code en modifiant la valeur de **\$isEnabled** (sur la première ligne).

En ajoutant le **else**, on a ajouté un autre message qui s'affiche dans le cas où la condition n'est pas vérifiée.

Si l'on veut avoir plus de cas possibles, on utilise la structure **if ... elseif ... else** :

```
<?php
    $isAllowedToEnter = "Oui";

    // SI on a l'autorisation d'entrer
    if ($isAllowedToEnter == "Oui") {
        // instructions à exécuter quand on est autorisé à entrer
    } // SINON SI on n'a pas l'autorisation d'entrer
    elseif ($isAllowedToEnter == "Non") {
        // instructions à exécuter quand on n'est pas autorisé à entrer
    } // SINON (la variable ne contient ni Oui ni Non, on ne peut pas agir)
    else {
        echo "Choix inconnu, merci de recommencer.";
    }
?>
```

Dans l'ordre, PHP rencontre les conditions suivantes :

- Si **\$isAllowedToEnter** est égale à « Oui », tu exécutes ces instructions...
- Sinon, si **\$isAllowedToEnter** est égale à « Non », tu exécutes ces autres instructions...
- Sinon, on redemande quel choix a été fait pour savoir si on a ou non l'autorisation d'entrer.

Exercice :

Déclarer une variable **\$genre** qui contient une chaîne de caractères.(« homme », « femme, ou une autre valeur.)

Créer une condition qui affiche un message différent en fonction de la valeur de la variable :

- « Bonjour Madame. »
- « Bonjour Monsieur. »
- « Bonjour à vous. »

VIII.2.1. Le cas des booléens

Si on regarde bien le dernier code, il serait plus judicieux d'utiliser des booléens.

Voici comment on teste une variable booléenne :

```
<?php
    $isAllowedToEnter = true;

    if ($isAllowedToEnter) {
        echo "Vous êtes autorisé(e) à accéder au site !";
    }
    else {
        echo "Accès refusé !!";
    }
?>
```

L'un des avantages des booléens, c'est qu'ils sont particulièrement adaptés aux conditions :

- On n'est pas obligé d'ajouter le `== true`.
- PHP comprend qu'il faut qu'il vérifie si `$isAllowedToEnter` vaut `true`.

Les avantages des booléens :

- c'est plus rapide à écrire pour vous ;
- ça se comprend bien mieux.

Et si l'on veut vérifier si le booléen vaut `false` ?

On utilise le `!` (l'opérateur NON) :

```
<?php
    $isAllowedToEnter = true;

    // Si pas autorisé
    if (!$isAllowedToEnter) {

    }
?>
```

VIII.3. Les conditions composées

Pour créer des conditions composées, nous allons avoir besoin d'utiliser les opérateurs logiques.

VIII.3.1. Les opérateurs logiques

En PHP, nous pouvons utiliser les opérateurs logiques suivants :

Opérateur	Equivalent en Algo	Définition
AND	ET	Renvoie <code>true</code> si toutes les comparaisons valent <code>true</code>
&&		Renvoie <code>true</code> si toutes les comparaisons valent <code>true</code>
OR	OU	Renvoie <code>true</code> si une des comparaisons vaut <code>true</code>
		Renvoie <code>true</code> si une des comparaisons vaut <code>true</code>
XOR	XOR	Renvoie <code>true</code> si une des comparaisons exactement vaut <code>true</code>
!	NON	Renvoie <code>true</code> si la comparaison vaut <code>false</code> (et inversement)

A noter : Les opérateurs avec des signes ont un ordre de priorité supérieur à ceux avec des mots clés.

Exemples :

```
<?php
    $isEnabled = true;
    $isOwner = false;

    if ($isEnabled && $isOwner) {
        echo 'Accès à la recette validé.';
    } else {
        echo 'Accès à la recette interdit !';
    }
?>
```

```
<?php
    $isEnabled = true;
    $isOwner = false;
    $isAdmin = true;

    if ((isEnabled && $isOwner) || $isAdmin) {
        echo 'Accès à la recette validé.';
    } else {
        echo 'Accès à la recette interdit !';
    }
?>
```

Ici on a ajouté une condition supplémentaire : soit la condition précédente s'applique, soit l'utilisateur concerné est un administrateur.

Exercice :

Les citoyens paient l'impôt selon les règles suivantes :

- les hommes de plus de 20 ans paient l'impôt
- les femmes paient l'impôt si elles ont entre 18 et 35 ans
- les autres ne paient pas d'impôt

Ecrire un script capable de réaliser ces tests avec des conditions composées.

VIII.4. La structure switch

Dans l'absolu, les structures à base de **if... elseif... else** suffisent pour traiter n'importe quelles conditions.

Mais quand on a de multiples cas à traiter on peut utiliser la structure **switch** qui peut rendre le code plus clair.

Exemple :

```
<?php
    $x = 2;

    switch($x) {
        case 0 :
            echo '$x vaut 0';
            break;
        case 1 :
            echo '$x vaut 1';
            break;
        case 2 :
            echo '$x vaut 2';
            break;
        case 3 :
            echo '$x vaut 3';
            break;
        case 4 :
            echo '$x vaut 4';
            break;
        default:
            echo '$x ne contient pas de valeur entre 0 et 4';
    }
?>
```

Remarques :

- On indique au début du switch la variable à tester,
- Le switch ne peut tester que l'égalité,
- Le mot clé **default** à la fin est l'équivalent du **else**,
- Il est impératif de terminer chaque **case** par **break**, sinon, PHP continue l'exécution.

VIII.5. Les structures ternaires

Il existe une autre forme de condition, qui est une structure conditionnelle condensée. Il s'agit de ce qu'on appelle les **ternaires**.

Une ternaire est une condition condensée qui sert à faire deux choses sur une seule ligne :

- Tester la valeur d'une variable dans une condition.
- Affecter une valeur à une variable selon que la condition est vraie ou non.

Prenons cet exemple à base de **if... else** qui met un booléen **\$isAdult** à vrai ou faux selon l'âge du visiteur :

```
<?php
$userAge = 24;

if ($userAge >= 18) {
    $isAdult = true;
}
else {
    $isAdult = false;
}
?>
```

On peut faire la même chose en une seule ligne grâce à une structure ternaire :

```
<?php
$userAge = 24;

$isAdult = ($userAge >= 18) ? true : false;
?>
```

Ici, tout notre test précédent a été fait sur une seule ligne.

La condition testée est **\$userAge >= 18**.

- Si c'est vrai, alors la valeur indiquée après le point d'interrogation (ici **true**) sera affectée à la variable **\$isAdult**.
- Sinon, c'est la valeur qui suit le symbole **:** (ici **false**) qui sera affectée à **\$isAdult**.

IX. Les boucles

Les boucles vont nous permettre d'exécuter plusieurs fois un bloc de code, c'est-à-dire d'exécuter un code « en boucle » tant qu'une condition donnée est vérifiée.

Nous disposons de quatre boucles différentes en PHP :

- La boucle **while** (« tant que »);
- La boucle **do... while** (« faire... tant que »);
- La boucle **for** (« pour »);
- La boucle **foreach** (« pour chaque »).

Nous verrons celle-ci plus tard car elle sert à parcourir des tableaux

Pour éviter de rester bloqué à l'infini dans une boucle, vous pouvez donc déjà noter qu'il faudra que la condition donnée soit fausse à un moment donné (pour pouvoir sortir de la boucle).

Pour que notre condition devienne fausse à un moment, on incrémentera ou on décrémentera la valeur de notre variable à chaque nouveau passage dans la boucle.

IX.1. Les opérateurs d'incrémantation et de décrémantation

Incrémenter une valeur signifie ajouter 1 à cette valeur tandis que décrémenter signifie enlever 1.

Les opérations d'incrémantation et de décrémantation vont principalement être utilisées avec les boucles en PHP. Elles vont pouvoir être réalisées grâce aux opérateurs d'incrémantation **++** et de décrémantation **--**.

Il y a deux façons d'incrémenter ou de décrémenter une variable :

- incrémenter/décrémenter la valeur de la variable puis retourner la valeur de la variable incrémentée ou décrémentée (on parle alors de **pré-incrémantation** et de **pré-décrémantation**),
- retourner la valeur de la variable avant incrémantation ou décrémantation puis l'incrémenter ou la décrémenter (on parle alors de **post-incrémantation** et de **post-décrémantation**).

Le tableau ci-dessous présente les différentes façons d'utiliser les opérateurs d'incrémantation et de décrémantation ainsi que le résultat associé :

Exemple	Résultat
<code>++\$x</code>	Pré-incrémantation : incrémente la valeur contenue dans la variable \$x, puis retourne la valeur incrémentée
<code>\$x++</code>	Post-incrémantation : retourne la valeur contenue dans \$x avant incrémantation, puis incrémente la valeur de \$x
<code>--\$x</code>	Pré-décrémantation : décrémente la valeur contenue dans la variable \$x, puis retourne la valeur décrémentée
<code>\$x--</code>	Post-décrémantation : retourne la valeur contenue dans \$x avant décrémantation, puis décrémente la valeur de \$x

Exemple :

```
<?php
$x = 2; $y = 2; $a = 2; $b = 2;

echo 'Post incrémantation pour $x : ' . $x++. '<br>';
echo '$x contient maintenant : ' . $x. '<br>';

echo 'Pré incrémantation pour $y : ' . ++$y. '<br>';
echo '$y contient maintenant : ' . $y. '<br>';

echo 'Post décrémantation pour $a : ' . $a--. '<br>';
echo '$x contient maintenant : ' . $a. '<br>';

echo 'Pré décrémantation pour $a : ' . --$b. '<br>';
echo '$x contient maintenant : ' . $b. '<br>';

?>
```

Ce qui affiche :

Incrémantion / décrémantion

```
Post incrémentation pour $x : 2
$x contient maintenant : 3
Pré incrémentation pour $y : 3
$y contient maintenant : 3
Post décrémentation pour $a : 2
$x contient maintenant : 1
Pré décrémentation pour $a : 1
$x contient maintenant : 1
```

Cet exemple montre les priorités de traitement du PHP selon que l'on place les signes ++ et -- avant ou après le nom de notre variable.

IX.2. La boucle while

La boucle **while** (« tant que ») est la boucle PHP la plus simple à appréhender.

La boucle **while** va nous permettre d'exécuter un certain bloc de code « tant qu'une » condition donnée est vérifiée.

Voici sa syntaxe :

```
<?php
    $x =0;

    while($x <= 10) {
        echo '$x contient la valeur ' . $x. '<br>';
        $x++;
    }
?>
```

La boucle while

```
$x contient la valeur 0
$x contient la valeur 1
$x contient la valeur 2
$x contient la valeur 3
$x contient la valeur 4
$x contient la valeur 5
$x contient la valeur 6
$x contient la valeur 7
$x contient la valeur 8
$x contient la valeur 9
$x contient la valeur 10
```

Remarque : Il faut toujours initialiser la variable utilisée dans la boucle en dehors de la boucle, sinon elle serait réinitialisée à chaque passage !

Exercice :

En utilisant la boucle **while**, afficher tous les codes postaux possibles pour le département **57**.

IX.3. La boucle do...while

La boucle PHP **do... while** (« faire... tant que » en français) ressemble à priori à la boucle **while** mais va fonctionner « en sens inverse » par rapport à **while**.

En effet, la boucle PHP **do... while** va également nous permettre d'exécuter un code tant qu'une condition donnée est vraie, mais cette fois-ci le code dans la condition sera exécuté avant que la condition soit vérifiée.

Ainsi, même si une condition est fausse dès le départ, on effectuera toujours au moins un passage au sein d'une boucle **do...while**, ce qui n'est pas le cas avec une boucle **while**.

Exemple :

```
<?php
    $x = 0;
    $y = 20;

    do{
        echo '$x contient la valeur ' . $x. '<br>';
        $x++;
    }while($x <= 5);

    do{
        echo '$y contient la valeur ' . $y. '<br>';
        $y++;
    }while($y <= 5);
?>
```

Ce qui affiche :

La boucle do...while

```
$x contient la valeur 0
$x contient la valeur 1
$x contient la valeur 2
$x contient la valeur 3
$x contient la valeur 4
$x contient la valeur 5
$y contient la valeur 20
```

Remarquez que la deuxième boucle s'est bien exécutée une fois alors que la condition était fausse dès le départ.

IX.4. La boucle for

La boucle PHP **for** (« pour ») est plus complexe à appréhender à priori que les boucles précédentes.

Cependant, cette boucle est très puissante et c'est celle qui sera majoritairement utilisée dans nos scripts PHP.

Voici sa syntaxe :

```
<?php
    for($x = 0; $x <=5; $x++){
        echo '$x contient la valeur ' . $x. '<br>';
    }
?>
```

Ce qui donne :

La boucle for

```
$x contient la valeur 0
$x contient la valeur 1
$x contient la valeur 2
$x contient la valeur 3
$x contient la valeur 4
$x contient la valeur 5
```

Analysons un peu cette syntaxe :

Après le mot **for**, il y a des parenthèses qui contiennent trois éléments, séparés par des points-virgules ; :

1. Le premier sert à l'**initialisation**. C'est la valeur que l'on donne au départ à la variable (ici, elle vaut 0).
2. Le second, c'est la **condition**. Comme pour le **while** : tant que la condition est remplie, la boucle est réexécutée. Dès que la condition ne l'est plus, on en sort.
3. Enfin, le troisième c'est l'**incrémantation**. Cela permet d'ajouter 1 à la variable à chaque tour de boucle.

Comment choisir entre while et for ?

Si vous hésitez entre les deux, il suffit de vous poser la question suivante : « **Est-ce que je sais d'avance combien de fois je veux que mes instructions soient répétées ?** ».

Si la réponse est oui, alors la boucle **for** est tout indiquée.

Sinon, alors il vaut mieux utiliser la boucle **while**.

X. Les tableaux :

Les tableaux en PHP sont des variables spéciales qui peuvent stocker plusieurs valeurs en même temps.

Dans un tableau, chaque valeur va être associée à une clef unique. Cette clef va nous permettre notamment de récupérer la valeur associée.

Il existe trois types de tableaux en PHP :

- Des tableaux numérotés ou indexés (les clefs vont être des nombres) ;
- Des tableaux associatifs (nous allons définir la valeur que l'on souhaite pour chaque clef) ;
- Des tableaux multidimensionnels (tableaux qui stockent d'autres tableaux en valeur).

Pour créer un tableau, on peut soit utiliser la structure de langage `array()`, soit la syntaxe `[]`.

X.1. Les tableaux indexés

X.1.1. Crédit d'un tableau indexé

Les tableaux indexés sont le type de tableaux le plus simple à créer en PHP : il suffit d'indiquer une série de valeurs et le PHP associera automatiquement une clef unique à chaque valeur, en commençant par 0.

Voici un exemple de code pour créer les tableaux \$prenoms et \$ages en utilisant les deux syntaxes :

```
<?php
    $prenoms = array("Marie", "Eric", "Jeanne", "Théodore", "Lisa")
    $ages = [25, 28, 19, 37, 22]
?>
```

On peut aussi créer un tableau « case » par « case » :

```
<?php
    $prenoms[0] = "Marie";
    $prenoms[1] = "Eric";
    $prenoms[2] = "Jeanne";
    $prenoms[3] = "Théodore";
    $prenoms[4] = "Lisa";
?>
```

On peut tout à fait créer des tableaux qui vont stocker des chaînes de caractères, des nombres, des booléens, etc.

X.1.2. Afficher les valeurs d'un tableau indexé

Pour afficher les valeurs d'un tableau numéroté une à une, il suffit d'**echo** notre variable tableau en précisant l'indice (entre crochets) correspondant à la valeur que l'on souhaite afficher :

```
<?php
    $prenoms = array("Marie", "Eric", "Jeanne", "Théodore", "Lisa");

    echo $prenoms[0] . "<br>";
    echo $prenoms[2];
?>
```

Tableaux

Marie
Jeanne

Pour parcourir tout le contenu d'un tableau, on utilise les boucles.

X.1.2.1. Avec la boucle for

Nous allons boucler sur les indices de notre tableau et nous allons récupérer la valeur associée à l'indice en question à chaque nouveau passage dans la boucle.

Mais il faut que nous connaissons la taille du tableau.

Pour cela on utilise la fonction **count()**.

Exemple :

```
<?php
    $prenoms = array("Marie", "Eric", "Jeanne", "Théodore", "Lisa");
    $taille = count($prenoms);

    for ($i=0; $i < $taille; $i++) {
        echo $prenoms[$i]. "<br>";
    }
?>
```

Tableaux

Marie
Eric
Jeanne
Théodore
Lisa

X.1.2.2. Avec la boucle foreach

Il est tout à fait possible d'attribuer les indices manuellement et de sauter certains indices pour stocker nos valeurs dans nos tableaux.

Dans ce cas-là, on ne pourra pas récupérer toutes les valeurs en bouclant sur les indices comme ci-dessus mais on utilisera plutôt une boucle **foreach** qui est une boucle spécialement créée pour les tableaux :

```
<?php
    $prenoms = array("Marie", "Eric", "Jeanne", "Théodore", "Lisa");
    $resultat = "";

    foreach ($prenoms as $valeur) {
        $resultat .= $valeur . '<br>';
    }
    echo $resultat;
?>
```

(cela affichera le même résultat que précédemment).

X.2. Les tableaux associatifs

Un tableau associatif est un tableau qui va utiliser des clefs textuelles qu'on va associer à chaque valeur. Ils vont être intéressants lorsqu'on voudra donner du sens à nos clefs.

Par exemple si on souhaite stocker les âges de nos différents utilisateurs dans un tableau, il est plus d'utiliser un tableau associatif en utilisant par exemple les pseudonymes de nos membres comme clefs.

X.2.1. Créer un tableau associatif

Les tableaux associatifs vont être différents des tableaux indexés au sens où nous allons devoir définir chacune des clefs : le PHP ne va pas ici pouvoir nommer automatiquement nos clefs.

Tout comme pour les tableaux numérotés, on va pouvoir créer notre tableau en une fois en utilisant la structure de langage **array()** ou la syntaxe **[]** ou le construire clef par clef et valeur par valeur :

```
<?php
    $ages = array("Marie" => 25, "Eric" => 28, "Jeanne" => 19, "Théodore" =>
37, "Lisa" => 22);
    $ages = [ "Marie" => 25, "Eric" => 28, "Jeanne" => 19, "Théodore" => 37,
"Lisa" => 22];

    // ou encore clé par clé :
    $mails["Marie"] = "marie@toto.com";
    $mails["Eric"] = "eric@tata.fr";
    $mails["Jeanne"] = "damejeannette@tutu.fr";
?>
```

X.2.2. Récupérer et afficher les valeurs d'un tableau associatif

On va pouvoir afficher une valeur en particulier d'un tableau associatif très simplement de la même façon que pour les tableaux indexé :

```
$ages = ["Marie" => 25, "Eric" => 28, "Jeanne" => 19, "Théodore" => 37,  
"Lisa" => 22];  
  
echo "Marie a " . $ages["Marie"]. " ans.<br>";
```

Pour parcourir un tableau associatif et par exemple afficher les valeurs les unes après les autres, nous allons en revanche être obligés d'utiliser la boucle **foreach** de la manière suivante :

```
$ages = ["Marie" => 25, "Eric" => 28, "Jeanne" => 19, "Théodore" => 37, "Lisa"  
=> 22];  
  
foreach ($ages as $cle => $valeur) {  
    $resultat .= $cle. ' a ' . $valeur . ' ans.<br>';  
}  
echo $resultat;
```

Ce qui affiche :

Tableaux

```
Marie a 25 ans.  
Eric a 28 ans.  
Jeanne a 19 ans.  
Théodore a 37 ans.  
Lisa a 22 ans.
```

Si l'on veut ne récupérer que la valeur des éléments du tableau on utiliser foreach de la même manière que pour un tableau indexé :

```
$ages = ["Marie" => 25, "Eric" => 28, "Jeanne" => 19, "Théodore" => 37, "Lisa"  
=> 22];  
  
foreach ($ages as $valeur) {  
    $resultat .= $valeur . ' ans.<br>';  
}  
echo $resultat;
```

Exercice :

Déclarer une variable de type **array** qui stocke les informations suivantes :

- France : Paris
- Allemagne : Berlin
- Italie : Rome

Afficher les valeurs de tous les éléments du tableau en utilisant la boucle **foreach**.

X.3. Les tableaux multidimensionnels

Un tableau multidimensionnel est un tableau qui va lui-même contenir d'autres tableaux en valeurs.

On appelle ainsi tableau à deux dimensions un tableau qui contient un ou plusieurs tableaux en valeurs, tableau à trois dimensions un tableau qui contient un ou plusieurs tableaux en valeurs qui contiennent eux-mêmes d'autres tableaux en valeurs et etc.

PHP ne limite pas le nombre de dimension d'un tableau, mais trop de dimensions rendent le code difficile à comprendre.

X.3.1. Créer un tableau multidimensionnel

On crée les tableaux multidimensionnels de la même manière que les autres :

```
<?php
/* Tableau multidimensionnel indexé stockant
des tableaux indexés*/
$suite = [
    [1, 2, 4, 8, 16],
    [1, 3, 9, 27, 81]
];
/* Tableau multidimensionnel indexé stockant
des tableaux associatifs*/
$users = [
    ["prenom" => "Marie", "age" => 25, "mail" => "marie@toto.com"],
    ["prenom" => "Eric", "age" => 28, "mail" => "eric@tata.fr"],
    ["prenom" => "Marie", "age" => 19, "mail" => "damejeannette@tutu.fr"],
];
/* Tableau multidimensionnel associatif stockant
des tableaux associatifs*/
$products = [
    "Livres" => ["poids" => 200, "quantite" => 10, "prix" => 15],
    "Stickers" => ["poids" => 10, "quantite" => 100, "prix" => 1.5],
];
?>
```

X.3.2. Récupérer ou afficher une valeur en particulier d'un tableau multidimensionnel

Pour récupérer une valeur en particulier dans un tableau multidimensionnel, on procède de la même manière qu'avec un tableau à une dimension, mais il faudra autant d'indices que de dimensions dans le tableau :

```
echo $suite[0][0] .'  
.' . $suite[0][2].'  
.' ;
echo $users[2]["prenom"]. '  
' ;
echo $products["Livres"]["prix"];
```

Tableaux

1
4
Marie
15

X.3.3. Parcourir et afficher les valeurs d'un tableau multidimensionnel

Pour parcourir un tableau multidimensionnel on utilise aussi la boucle **foreach**.

Si l'on veut ne récupérer que certaines valeurs des sous-tableaux :

```
foreach($users as $user) {  
    echo $user['prenom'] . ' à ' . $user['age'] . ' ans.<br>';  
}
```

Marie à 25 ans.
Eric à 28 ans.
Marie à 19 ans.

Si l'on souhaite parcourir toutes les valeurs d'un tableau multidimensionnel on va utiliser autant de boucles **foreach** imbriquées que de dimensions dans le tableau :

```
foreach($products as $clef => $product) {  
    echo 'Produit : ' . $clef. '<br>';  
    foreach($product as $propertie => $value) {  
        echo $propertie . ' : ' . $value. '<br>';  
    }  
    echo '<br>';  
}
```

Produit : Livres
poids : 200
quantite : 10
prix : 15

Produit : Stickers
poids : 10
quantite : 100
prix : 1.5

X.4. Afficher rapidement la structure d'un tableau en PHP

Parfois, on voudra simplement afficher la structure d'un tableau PHP sans mise en forme pour vérifier ce qu'il contient ou pour des questions de débogage.

Le PHP nous fournit deux possibilités de faire cela : on va pouvoir soit utiliser la fonction **print_r()**, soit la fonction **var_dump()** que nous connaissons déjà pour afficher n'importe quel type de tableaux (numérotés, associatifs ou multidimensionnels).

Notez que **var_dump()** va nous fournir davantage d'informations que **print_r()** :

On va généralement utiliser cette fonction avec l'élément HTML `pre` pour avoir un meilleur affichage de la structure du tableau qu'on souhaite afficher (je vous rappelle que `pre` va permettre de conserver la mise en forme de notre code) :

```
echo '<pre>';
print_r($products);
var_dump($products);
echo '</pre>';

Array
(
    [Livres] => Array
        (
            [poids] => 200
            [quantite] => 10
            [prix] => 15
        )

    [Stickers] => Array
        (
            [poids] => 10
            [quantite] => 100
            [prix] => 1.5
        )
)

C:\wamp64\www\PHP\Cours\exempleTableauxMulti.php:49:
array (size=2)
'Livres' =>
    array (size=3)
        'poids' => int 200
        'quantite' => int 10
        'prix' => int 15
'Stickers' =>
    array (size=3)
        'poids' => int 10
        'quantite' => int 100
        'prix' => float 1.5
```

Exercice 9

Reprenez le script de l'exercice 4, en créant un tableau associatif multidimensionnel pour stocker la liste des étudiants et leurs informations de la forme suivante :

```
$stagiaires = [
"Jean" => [ "prenom" => "Jean", "nom" => "DUPONT", "age" => 35, "ville" =>
"METZ"]
]
```

et utiliser des boucles foreach pour obtenir l'affichage suivant :

Liste des étudiants

- Jean
 - prenom : Jean
 - nom : DUPONT
 - age : 35
 - ville : METZ

X.5. Manipuler les tableaux avec les fonctions prédéfinies

PHP inclue beaucoup de fonctions pour manipuler les tableaux. En voici quelques-unes :

Rechercher dans un tableau :

- `array_key_exists()` pour vérifier si une clé existe dans le tableau,
- `in_array()` pour vérifier si une valeur existe dans le tableau,
- `array_search()` pour récupérer la clé d'une valeur dans le tableau.

Ajouter et retirer des valeurs dans un tableau :

- `array_push()` : Ajoute un ou plusieurs éléments à la fin d'un tableau,
- `array_unshift()` Ajoute un ou plusieurs éléments au début d'un tableau,
- `array_pop()` Supprime un élément de la fin d'un tableau,
- `array_shift()` Supprime un élément au début d'un tableau.

Il en existe beaucoup d'autres qui permettent par exemple de fusionner des tableaux, de les trier etc.

La documentation officielle vous apportera toutes les réponses :

<https://www.php.net/manual/fr/ref.array.php>

XI. Les fonctions

Comme les boucles, les fonctions permettent d'éviter d'avoir à répéter du code PHP que l'on utilise souvent. Mais alors que les boucles sont de bêtes machines tout juste capables de répéter deux cents fois la même chose, les fonctions sont des robots « intelligents » qui s'adaptent en fonction de ce que vous voulez faire, et qui automatisent grandement la plupart des tâches courantes.

XI.1. Définition des fonctions et fonctions internes ou prédéfinies

Une fonction correspond à une série cohérente d'instructions qui ont été créées pour effectuer une tâche précise. Pour exécuter le code contenu dans une fonction, il va falloir appeler la fonction.

Nous avons déjà croisé des fonctions par exemple la fonction `var_dump()` dont le rôle est de renvoyer les informations d'une variable.

La fonction `var_dump()` fait partie des fonctions dites « internes » au PHP ou « prêtées à l'emploi » tout simplement car sa définition fait partie du langage PHP.

En effet, vous devez bien comprendre ici que la fonction `var_dump()` contient une série d'instructions qui permettent de renvoyer les informations d'une variable, mais nous n'avons pas besoin de savoir ce qu'elle contient, juste comment l'utiliser.

Les fonctions prédéfinies sont l'une des plus grandes forces du PHP puisqu'il en existe plus de 1000 qui vont couvrir quasiment tous nos besoins.

XI.2. Les fonctions définies par l'utilisateur

En plus des fonctions internes, PHP nous laisse la possibilité de définir nos propres fonctions. Cela va s'avérer très pratique dans le cas on va souvent devoir répéter les mêmes opérations.

Pour utiliser nos propres fonctions, nous allons déjà devoir les définir, c'est-à-dire préciser une première fois la série d'instructions que chaque fonction devra exécuter lors de son appel.

Pour déclarer une fonction, il faut déjà commencer par préciser le mot clef **function** qui indique au PHP qu'on va définir une fonction personnalisée.

Ensuite, nous allons devoir préciser le **nom** de notre fonction. Le nom des fonctions va suivre les mêmes règles que celui des variables, à savoir qu'il devra commencer par une lettre ou un underscore et ne devra pas être un nom déjà pris par le langage PHP.

A la différence des variables le nom des fonctions est insensible à la casse. Cela signifie que l'utilisation de majuscules et des minuscules ne servent pas à différencier une fonction d'une autre.

Exemple :

```
<?php

function bonjour(){
    echo "Bonjour tout le monde !<br>";
}
?>
```

Une fonction ne s'exécute pas au chargement de la page. Pour l'exécuter, il va falloir l'appeler par son nom, suivi de parenthèses, et on peut l'appeler autant de fois que nécessaire :

```
<?php
    function bonjour(){
        echo "Bonjour tout le monde !<br>";
    }
    bonjour();
    bonjour();
?>
```

Fonctions

Bonjour tout le monde !
Bonjour tout le monde !

XI.3. Les paramètres et arguments des fonctions

Souvent, les fonctions vont avoir besoin d'informations externes.

Par exemple, notre fonction **var_dump()** va avoir besoin qu'on lui fournisse ou qu'on lui « passe » une variable pour pouvoir récupérer ses informations.

Les informations qu'on va passer à une fonction sont appelées des **arguments**. Nous allons toujours passer les arguments dans les parenthèses de notre fonction. Certaines fonctions ne vont pas avoir besoin d'argument pour fonctionner, d'autres vont avoir besoin d'un argument, d'autres encore de deux, etc. Par ailleurs, certains arguments vont parfois être facultatifs tandis que d'autres seront obligatoires.

Quelles différences entre un **paramètre** et un **argument**? On parlera de **paramètre** lors de la **définition d'une fonction**. Un **paramètre** sert à indiquer qu'une fonction va avoir besoin d'un **argument** pour fonctionner mais ne correspond pas à une valeur effective : ce n'est qu'un prête nom. Un **argument** en revanche correspond à **la valeur effective passée** à une fonction.

Exemples :

```
<?php
$prenom = "Pierre";
$x = 4;
$y = 5;

function bonjour($p){
    echo "Bonjour " . $p. "<br>";
}

function addition($p1, $p2){
    echo $p1. " + ". $p2. " = ". ($p1+$p2). "<br>";
}

bonjour($prenom);
bonjour("Mathilde");
addition($x, $y);
addition(1,1);
?>
```

Fonctions

Bonjour Pierre
Bonjour Mathilde
 $4 + 5 = 9$
 $1 + 1 = 2$

XI.3.1. Passer des arguments par référence

Jusqu'à présent, nous avons passé nos arguments par valeur à nos fonctions ce qui correspond au passage par défaut en PHP.

Lorsqu'on passe une variable comme argument par valeur à une fonction, le fait de modifier la valeur de la variable à l'intérieur de la fonction ne va pas modifier sa valeur à l'extérieur de la fonction :

```
<?php
    $x = 0;
    function plus3($p){
        $p = $p +3;
        echo 'Valeur dans la fonction : ' . $p;
    }

    plus3($x);
    echo '<br> Valeur en dehors de la fonction : ' . $x;
?>
```

Fonctions

```
Valeur dans la fonction : 3
Valeur en dehors de la fonction : 0
```

Parfois, on voudra cependant que nos fonctions puissent modifier la valeur des variables qu'on leur passe en argument. Pour cela, il va falloir passer ces arguments par référence.

Pour indiquer qu'on souhaite passer un argument par référence à une fonction, il suffit d'ajouter le signe **&** devant le paramètre en question dans la définition de la liste des paramètres de la fonction.

```
<?php
    $x = 0;

    function plus3(&$p){
        $p = $p +3;
        echo 'Valeur dans la fonction : ' . $p;
    }

    plus3($x);
    echo '<br> Valeur en dehors de la fonction : ' . $x;
?>
```

Fonctions

```
Valeur dans la fonction : 3
Valeur en dehors de la fonction : 3
```

XI.4. Les valeurs de retour d'une fonction

Jusqu'à présent, le seul moyen que nous avions d'avoir accès au résultat d'une fonction qu'on avait créée était d'afficher ce résultat en utilisant un **echo** dans la définition de la fonction.

L'instruction **return** va nous permettre de demander à une fonction de retourner un résultat qu'on va ensuite pouvoir stocker dans une variable ou autre pour le manipuler.

Attention cependant : l'instruction **return** va terminer l'exécution d'une fonction, ce qui signifie qu'on placera généralement cette instruction en fin de fonction puisque le code suivant une instruction **return** dans une fonction ne sera jamais lu ni exécuté.

```
<?php
    function multEcho($a, $b){
        echo $a. ' x ' . $b. ' = ' . $a*$b. '<br>';
    }

    function multReturn($a, $b){
        return $a*$b;
    }

    multEcho(2, 3);

    $res = multReturn(4, 5);
    echo $res;
?>
```

Fonctions

2 x 3 = 6
20

Utiliser **return** dans nos fonctions permet donc une bien plus grande souplesse. D'ailleurs, on privilégiera toujours un **return** à la fin d'une fonction.

XII. La portée des variables

En PHP, nous pouvons déclarer des variables n'importe où dans notre script : au début du script, à l'intérieur de boucles, au sein de nos fonctions, etc.

Les variables peuvent avoir deux portées différentes : soit une portée globale, soit une portée locale.

Toute variable définie en dehors d'une fonction a une portée globale. Par définition, une variable qui a une portée globale est accessible « globalement », c'est-à-dire dans tout le script sauf dans les espaces locaux d'un script.

Au contraire, toute variable définie à l'intérieur d'une fonction va avoir une portée locale à la fonction. Cela signifie que la variable ne sera accessible qu'au sein de la fonction et notre variable sera par ailleurs par défaut détruite dès la fin de l'exécution de la fonction.

Exemples :

```
<?php
$x = 10;

function portee1(){
    echo 'La valeur de $x globale est : ' . $x. '<br>';
}
function portee2(){
    $x=5;
    echo 'La valeur de $x locale est : ' . $x. '<br>';
}
function portee3(){
    $y=0;
    $y++;
    echo 'La valeur de $y est : ' . $y. '<br>';
}
function portee4(){
    $z=1;
}

portee1();
portee2();
portee3();
portee3();
portee4();
echo 'La variable locale $z contient : ' . $z;
?>
```

Portée des variables

(!) Notice: Undefined variable: x in C:\wamp64\www\PHP Cours\exemplePortée.php on line 13				
Call Stack				
#	Time	Memory	Function	Location
1	0.0064	409304	{main}()	...\exemplePortée.php:0
2	0.0064	409304	portee1()	...\exemplePortée.php:28

La valeur de \$x globale est :

La valeur de \$x locale est : 5

La valeur de \$y est : 1

La valeur de \$y est : 1

(!) Notice: Undefined variable: z in C:\wamp64\www\PHP Cours\exemplePortée.php on line 33				
Call Stack				
#	Time	Memory	Function	Location
1	0.0064	409304	{main}()	...\exemplePortée.php:0

La variable locale \$z contient :

Comme vous pouvez le constater :

\$x est définie dans l'espace global,

Dans portee1(), on ne peut pas y accéder,

Dans portee2(), on définit une variable locale \$x (différente de \$x globale),

Dans portee3(), qui est appelée 2 fois on voit bien que \$y reste à 1, car la variable est détruite à chaque exécution,

Et dans portee4(), on définit une variable locale \$z, et on voit bien que l'on ne peut pas y accéder depuis l'espace global.

XII.1. Accéder à une variable de portée globale depuis un espace local

Parfois, nous voudrons nous servir de variables possédant une portée globale (c'est-à-dire définies en dehors d'une fonction) à l'intérieur d'une fonction.

Pour cela, on va pouvoir utiliser le mot clef **global** avant la déclaration des variables qu'on souhaite utiliser dans notre fonction.

Pour être tout à fait précis, on dit que les variables globales sont importées dans le contexte local par référence :

```
$a=10;
function portee(){
    global $a;
    echo 'La valeur de $a globale est : '.$a.'  
>';
    $a+=5;
}

portee();
echo '$a contient maintenant : ' . $a;
```

La valeur de \$a globale est : 10
\$a contient maintenant : 15

XII.2. Accéder à une variable définie localement depuis l'espace global

Il n'y a aucun moyen d'accéder à une variable définie localement depuis l'espace global. La seule manière de récupérer la valeur finale d'une variable définie localement et la stocker dans une nouvelle variable globale en utilisant l'instruction **return** :

```
function portee5(){
    $toto = 5;
    return $toto;
}

$titi = portee5();
echo '$toto contient la valeur : ' . $titi;
$stoto contient la valeur : 5
```

Notez cependant bien ici qu'une variable locale n'aura toujours qu'une portée locale et que sa portée ne pourra pas être étendue dans l'espace global.

XII.3. Le mot clef static

Une variable définie localement va être supprimée ou détruite dès la fin de l'exécution de la fonction dans laquelle elle a été définie.

Pour qu'une fonction de « souvienne » de la dernière valeur d'une variable définie dans la fonction, nous allons pouvoir utiliser le mot clef **static** devant la déclaration initiale de la variable.

```
function compteur(){
    static $x =0;
    echo '$x contient : ' . $x . '<br>';
    $x++;
}
compteur();
compteur();
compteur();
compteur();
compteur();
```

\$x contient : 0
\$x contient : 1
\$x contient : 2
\$x contient : 3
\$x contient : 4

XIII. Les constantes

XIII.1. Définir une constante

Pour définir une constante en PHP, nous allons pouvoir utiliser la fonction **define()** ou le mot clef **const**.

Pour créer une constante en utilisant **define()**, on passe le nom et la valeur de la constante en arguments de cette fonction. Une fois qu'une constante est définie, elle ne peut jamais être modifiée ni détruite.

Si vous souhaitez créer une constante en utilisant le mot clef **const**, alors vous devez noter que seules des données de type **string**, **integer**, **float**, **boolean** et **array** peuvent être stockées dans notre constante.

Par convention on écrit toujours le nom d'une constante en majuscules et à la différence des variables, nous ne devrons pas préfixer le nom d'une constante avec le signe **\$**.

Pour ensuite utiliser la valeur d'une constante ou l'afficher, il suffit d'appeler la constante par son nom comme on en a l'habitude avec les variables :

```

define('DIX', 10);
echo "la constante DIX contient : " .DIX.<br>';

const DOUZE = 12;
echo 'la constante DOUZE contient : '.DOUZE;

```

XIII.2. Les constantes prédéfinies et les constantes magiques

Le PHP nous fournit également un certain nombre de constantes prédéfinies qui vont généralement nous donner des informations de type « meta » comme la version de PHP actuellement utilisée, le plus petit entier supporté par PHP, des constantes de rapport d'erreur etc.

Parmi ces constantes prédéfinies, il y en a neuf qui vont retenir notre attention et qu'on appelle des constantes « magiques ». Ces constantes se distinguent des autres puisque leur valeur va changer en fonction de l'endroit dans le script où elles vont être utilisées.

Les constantes magiques sont reconnaissables par le fait que leur nom commence et termine par deux underscores.

En voici quelques-unes :

Nom de la constante	Description
<u>_FILE_</u>	Contient le chemin complet et le nom du fichier
<u>_DIR_</u>	Contient le nom du dossier dans lequel est le fichier
<u>_LINE_</u>	Contient le numéro de la ligne courante dans le fichier
<u>_FUNCTION_</u>	Contient le nom de la fonction actuellement définie ou {closure} pour les fonctions anonymes

XIV. Include des fichiers dans d'autres

Inclure des fichiers php dans d'autres est très utile pour construire un site web. Pour cela, on dispose de 4 fonctions :

- `include()`
- `require()`
- `include_once()`
- `require_once()`

XIV.1. Présentation de include et de require

Les instructions PHP `include` et `require` vont nous permettre toutes deux d'inclure des fichiers de code (ou plus exactement le contenu de ces fichiers) à l'intérieur d'autres fichiers de code.

Par exemple, sur votre site, plutôt que de réécrire tout le code html à chaque page, vous pourrez inclure le menu et le footer, ou bien faire un gabarit contenant le header et le footer et inclure le contenu principal.

XIV.2. Les différences entre `include`, `include_once`, `require` et `require_once`

La seule différence entre les instructions `include` et `require` est la réponse du PHP dans le cas où le fichier ne peut pas être inclus pour une quelconque raison (fichier introuvable, indisponible, etc.).

Dans ce cas-là, si l'inclusion a été tentée avec `include`, le PHP renverra un simple avertissement et le reste du script s'exécutera alors qu'avec `require`, une erreur fatale sera retournée par PHP et l'exécution du script s'arrêtera immédiatement.

L'instruction `require` est donc plus « stricte » que `include`.

La différence entre les instructions `include` et `require` et leurs variantes `include_once` et `require_once` est qu'on va pouvoir inclure plusieurs fois un même fichier avec `include` et `require` tandis qu'en utilisant `include_once` et `require_once` un même fichier ne pourra être inclus qu'une seule fois dans un autre fichier.

Exemples :

```
<?php
    require 'functions.php';
    include 'menu.php';
?>
<main>
    ...
</main>
<?php
    include 'footer.php'
?>
```

Ou

```
<header>
    ...
</header>
<main>
    <?php
        include 'contact.php'
    ?>
</main>
<footer>
    ...
</footer>
```

Dans la pratique, on utilisera plutôt `require` ou `require_once` pour des fichiers contenant du traitement, et `include` ou `include_once` pour des fichiers destinés à être affichés.

XV. Transmettre des données de page en page

XV.1. Envoyez des paramètres dans l'URL

Les URL permettent de transmettre des informations. Voyons comment cela fonctionne.

- Imaginons que votre site s'appelle **monsite.com** ;
- et possède une page PHP de contact : **contact.php** .

Pour accéder à la page de contact, vous devez aller à l'URL suivante :

http://www.monsite.com/contact.php

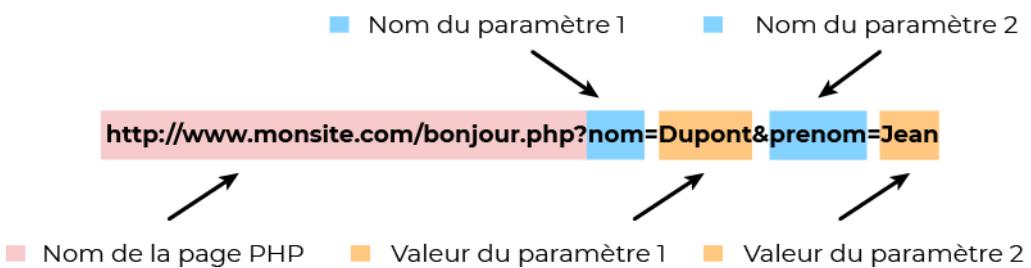
Voyons comment envoyer des informations à la page **contact.php** .

Ces informations vont figurer à la fin de l'URL, comme ceci :

http://www.monsite.com/contact.php?nom=Dupont&prenom=Jean

Ce que vous voyez après le point d'interrogation, ce sont des **paramètres** que l'on envoie à la page PHP. Celle-ci peut récupérer ces informations dans des variables.

Voici comment on peut découper cette URL :



Le point d'interrogation sépare le nom de la page PHP des paramètres.

Les paramètres s'enchaînent selon la forme **nom=valeur** et sont séparés les uns des autres par le symbole **&** .

On peut transmettre autant de paramètres que l'on veut, la seule limite est la longueur de l'URL. En général, il n'est pas conseillé de dépasser les 256 caractères.

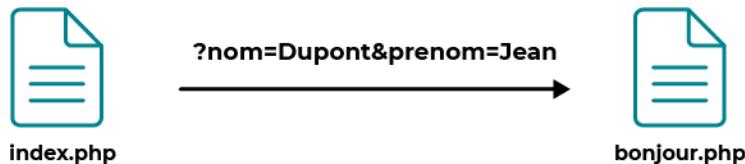
XV.1.1. Créez un lien avec des paramètres

Maintenant que nous savons cela, nous pouvons créer des liens en HTML, qui transmettent des paramètres d'une page vers une autre.

Imaginons que vous ayez deux fichiers sur votre site :

- index.php (l'accueil) ;
- bonjour.php .

Nous voulons faire un lien de **index.php** à **bonjour.php** pour transmettre des informations dans l'URL :



Pour cela, vous allez insérer le code suivant dans index.php :

```
<a href="bonjour.php?prenom=Jean&nom=Dupont">Dis-moi bonjour !</a>
```

Ce lien appelle la page **bonjour.php** et lui envoie deux paramètres :

- nom : Dupont ;
- prenom : Jean.

XV.1.2. Créez un formulaire avec la méthode HTTP GET

La deuxième solution pour faire passer des informations dans l'URL, c'est de proposer à l'utilisateur de soumettre un formulaire avec la méthode HTTP GET.

Nous pouvons faire cela avec une balise **form** qui a pour attribut **method** la valeur GET.

```
<form action="bonjour.php" method="GET">
    <div>
        <label for="prenom">Prénom</label>
        <input type="text" name="prenom">
    </div>
    <br>
    <div>
        <label for="nom">Nom</label>
        <input type="text" name="nom">
    </div>
    <br>
    <button type="submit">Envoyer</button>
</form>
```

Les données soumises à l'aide du formulaire se retrouveront dans l'URL, comme pour un lien.

XV.2. Récupérez les paramètres en PHP

Pour notre besoin, nous avons un formulaire dans `index.php` que nous allons soumettre sur une autre page, et qui affichera un message avec le nom et le prénom dans `bonjour.php`.

Le formulaire va alors être converti en lien vers

`bonjour.php? prenom=Jean&nom= Dupont`

Et ces informations pourront être récupérées par PHP dans le fichier `bonjour.php`.

Lors de la soumission, une variable superglobale appelée `$_GET` va contenir les données envoyées. Il s'agit d'un tableau associatif dont les clés correspondent aux noms des paramètres envoyés dans l'URL.

On peut donc :

- récupérer ces informations ;
- les traiter
- les afficher ;

Pour l'exemple, dans `bonjour.php`, placez le code suivant :

```
<html>
  <head>
    <meta charset="UTF-8">
    <title>Bonjour</title>
  </head>
  <body>
    <h1>Informations bien reçues !</h1>

    <p>Bonjour <?php echo $_GET['prenom']. " ". $_GET['nom']; ?></p>

  </body>
</html>
```

Mais c'est loin d'être suffisant, pour l'instant les données transmises ne sont absolument pas sécurisées, et nous ne vérifions pas la présence et le contenu des paramètres.

De plus, transmettre des formulaires avec la méthode GET est une mauvaise pratique, ici, ce n'était que pour l'exemple !

Pour sécuriser un peu cet échange de données, voyons le code suivant :

```
<?php
if (isset($_GET['prenom']) && isset($_GET['nom']))
    && !empty($_GET['prenom']) && !empty($_GET['nom'])) {
    $prenom = htmlspecialchars($_GET['prenom']);
    $nom = htmlspecialchars($_GET['nom']);
    $msg = "Bonjour " . $prenom. " ". $nom. " !";
} else {
    $msg = "Paramètres invalides";
}
?>

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Bonjour</title>
    </head>
    <body>
        <h1>Informations bien reçues !</h1>

        <p><?php echo $msg; ?></p>

    </body>
</html>
```

- On commence par vérifier que les paramètres existent avec **isset()** ;
- On s'assure qu'ils ne soient pas vides avec **!empty()** ;
- Convertit les caractères spéciaux en entités HTML, protège contre du code malveillant avec **htmlspecialchars()** ;

Tout ce qui est affiché et qui vient, à la base, d'un visiteur, vous devez penser à le protéger avec **htmlspecialchars()**, afin de se protéger des failles XSS.

La **faille XSS** (pour cross-site scripting) est vieille comme le monde (ehu, comme le Web) et on la trouve encore sur de nombreux sites web, même professionnels ! C'est une technique qui consiste à injecter du code HTML contenant du JavaScript dans vos pages, pour le faire exécuter à vos visiteurs.

En résumé, ne faites jamais confiance aux informations provenant d'un visiteur, d'où l'expression :

Never Trust User Inputs.

XV.3. Envoyez des données par la méthode POST

Comme indiqué précédemment, l'envoi de formulaire par la méthode GET n'est pas une bonne pratique. Il faut toujours privilégier la méthode POST pour envoyer des formulaires.

Avec cette méthode, les données ne transiteront pas par l'URL, l'utilisateur ne les verra donc pas passer dans la barre d'adresse.

Cette méthode permet d'envoyer autant de données que l'on veut, ce qui fait qu'on la privilégie le plus souvent.

Néanmoins, les données ne sont pas plus sécurisées qu'avec la méthode GET , et il faudra toujours vérifier si tous les paramètres sont bien présents et valides, comme on l'a fait dans le chapitre précédent. **On ne doit pas plus faire confiance aux formulaires qu'aux URL.**

Lors de l'envoi d'un formulaire avec la méthode POST, c'est la superglobale `$_POST` qui recevra les données. Comme pour `$_GET`, `$_POST` est un tableau associatif qui aura pour clés les attribut name des champs du formulaire.

On a vu que la fonction `htmlspecialchars()` permettait de neutraliser les balises HTML envoyées dans un formulaire. Par contre elle les affichera quand même au format texte. Si vous voulez les retirer, vous pouvez utiliser la fonction `strip_tags()`.

Exemple d'un formulaire de contact en POST :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Contact</title>
    </head>
    <body>
        <h1>Contact</h1>
        <form action="contact.php" method="POST">
            <div>
                <label for="email">Email</label>
                <input type="email" name="email">
            </div>
            <div>
                <label for="message">Votre message</label>
                <textarea placeholder="Exprimez vous"
                    name="message"></textarea>
            </div>
            <button type="submit">Envoyer</button>
        </form>
    </body>
</html>
```

Et la page submit_contact :

```
<?php

if (!isset($_POST['email']) || (!filter_var($_POST['email'],
FILTER_VALIDATE_EMAIL)) || (!isset($_POST['message']) ||
empty($_POST['message'])))
{ echo 'Il faut un email et un message valides pour
soumettre le formulaire.';

} else {
    $email = $_POST['email'];
    $message = strip_tags($_POST['message']);
}
?>

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Demande de contact reçue</title>
        <link
            href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
            rel="stylesheet"
        >
    </head>
    <body>
        <div class="container">

            <h1>Message bien reçu !</h1>

            <div class="card">

                <div class="card-body">
                    <h5 class="card-title">Rappel de vos informations</h5>
                    <p class="card-text"><b>Email</b> : <?php echo $email;
?></p>
                    <p class="card-text"><b>Message</b> : <?php echo $message;
?></p>
                </div>
            </div>
        </body>
    </html>
```

Vous remarquez ici une autre fonction, filter_var() qui permet de filtrer des variables (voir la doc, beaucoup de filtres existent avec cette fonction).

XV.4. Envoyer des fichiers avec un formulaire

XV.4.1. Paramétrez le formulaire d'envoi de fichier

Si l'on souhaite offrir au visiteur la possibilité d'envoyer des fichiers avec un formulaire, il faut ajouter l'attribut `enctype="multipart/form-data"` à la balise `<form>` :

```
<form action="submit_contact.php" method="POST" enctype="multipart/form-data">
    <!-- Formulaire -->
</form>
```

Grâce à `enctype`, le navigateur du visiteur sait qu'il s'apprête à envoyer des fichiers.

Maintenant, nous pouvons ajouter à l'intérieur du formulaire une balise permettant d'envoyer un fichier.

C'est une balise très simple de type `<input type="file" />`.

Il faut donner un nom à ce champ de formulaire (grâce à l'attribut `name`) pour que PHP puisse reconnaître le champ par la suite :

```
<form action="submit_contact.php" method="POST" enctype="multipart/form-data">
    <!-- Formulaire -->
    <div>
        <label for="Screenshot">Votre capture d'écran</label>
        <input type="file" name="Screenshot" />
    </div>

    <button type="submit">Envoyer</button>
</form>
```

XV.4.2. Traiter l'envoi du fichier

C'est maintenant que ça devient important. Il faut que l'on ajoute du code dans la page `submit_contact.php` pour traiter l'envoi du fichier.

Au moment où la page PHP s'exécute, le fichier a été envoyé sur le serveur mais il est stocké dans un dossier temporaire.

C'est à vous de décider si vous acceptez définitivement le fichier ou non.

Vous pouvez par exemple vérifier si le fichier a la bonne extension (si vous demandez une image et qu'on vous envoie un « .txt », vous devrez refuser le fichier).

Si le fichier est bon, vous l'accepterez grâce à la fonction `move_uploaded_file` et ce, d'une manière définitive.

Pour chaque fichier envoyé, une variable `$_FILES['nom_du_champ']` est créée.

Dans notre cas, la variable s'appellera `$_FILES['Screenshot']`

Cette variable est un tableau qui contient plusieurs informations sur le fichier :

Variable	Signification
<code>\$_FILES['Screenshot']['name']</code>	Contient le nom du fichier envoyé par le visiteur.
<code>\$_FILES['Screenshot']['type']</code>	Indique le type du fichier envoyé. Si c'est une image gif par exemple, le type sera <code>image/gif</code>
<code>\$_FILES['Screenshot']['size']</code>	Indique la taille du fichier envoyé. Attention : cette taille est en octets. Il faut environ 1 000 octets pour faire 1 Ko, et 1 000 000 d'octets pour faire 1 Mo. La taille de l'envoi est limitée par PHP. Par défaut, impossible d'uploader des fichiers de plus de 8 Mo.
<code>\$_FILES['Screenshot']['tmp_name']</code>	Juste après l'envoi, le fichier est placé dans un répertoire temporaire sur le serveur en attendant que votre script PHP décide si oui ou non il accepte de le stocker pour de bon. Cette variable contient l'emplacement temporaire du fichier (c'est PHP qui gère ça).
<code>\$_FILES['Screenshot']['error']</code>	Contient un code d'erreur permettant de savoir si l'envoi s'est bien effectué ou s'il y a eu un problème et si oui, lequel. La variable vaut 0 s'il n'y a pas eu d'erreur.

Si vous avez mis un second champ d'envoi de fichier dans votre formulaire, il y aura une seconde variable `$_FILES['nom_de_votre_autre_champ']` découpée de la même manière que le tableau qu'on vient de voir ici.

Nous allons faire les vérifications suivantes pour décider si l'on accepte le fichier ou non.

- Vérifier tout d'abord si le visiteur a bien envoyé un fichier, en testant la variable `$_FILES['Screenshot']` avec `isset()` et s'il n'y a pas eu d'erreur d'envoi, grâce à `$_FILES['Screenshot']['error']` .
- Vérifier si la taille du fichier ne dépasse pas 1 Mo par exemple (environ 1 000 000 d'octets), grâce à `$_FILES['Screenshot']['size']` .
- Vérifier si l'extension du fichier est autorisée (il faut interdire à tout prix que les gens puissent envoyer des fichiers PHP, sinon ils pourraient exécuter des scripts sur votre serveur). Dans notre cas, nous autoriserons seulement les images (fichiers .png, .jpg, .jpeg et .gif).

Nous analyserons pour cela la variable `$_FILES['Screenshot']['name']` .

Nous allons donc faire une série de tests dans notre page `submit_contact.php` .

XV.4.2.1. Tester si le fichier a bien été envoyé

On commence par vérifier qu'un fichier a été envoyé.

Pour cela, on va tester si la variable `$_FILES['screenshot']` existe avec `isset()`.

On vérifie dans le même temps s'il n'y a pas d'erreur d'envoi :

```
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['screenshot']) && $_FILES['screenshot']['error'] == 0){}
```

XV.4.2.2. Vérifier la taille du fichier

On veut interdire que le fichier dépasse 1 Mo, soit environ 1 000 000 d'octets. On doit donc tester `$_FILES['screenshot']['size']` :

```
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['screenshot']) && $_FILES['screenshot']['error'] == 0){
    // Testons si le fichier n'est pas trop gros
    if ($_FILES['screenshot']['size'] <= 1000000)
```

XV.4.2.3. Vérifier l'extension du fichier

On peut récupérer l'extension du fichier dans une variable grâce à ce code :

```
$fileInfo = pathinfo($_FILES['screenshot']['name']);
$extension = $fileInfo['extension'];
```

La fonction `pathinfo` renvoie un tableau contenant entre autres l'extension du fichier dans `$fileInfo['extension']`.

On stocke ça dans une variable `$extension`.

Une fois l'extension récupérée, on peut la comparer à un tableau d'extensions autorisées, et vérifier si l'extension récupérée fait bien partie des extensions autorisées à l'aide de la fonction `in_array()` :

```
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['screenshot']) AND $_FILES['screenshot']['error'] == 0)
{
    // Testons si le fichier n'est pas trop gros
    if ($_FILES['screenshot']['size'] <= 1000000)
    {
        // Testons si l'extension est autorisée
        $fileInfo = pathinfo($_FILES['screenshot']['name']);
        $extension = $fileInfo['extension'];
        $allowedExtensions = ['jpg', 'jpeg', 'gif', 'png'];
        if (in_array($extension, $allowedExtensions))
        {
        }
    }
}
```

XV.4.2.4. Valider l'upload du fichier

Si tout est bon, on accepte le fichier en appelant `move_uploaded_file()`.

Cette fonction prend deux paramètres :

1. Le nom temporaire du fichier (on l'a avec `$_FILES['Screenshot']['tmp_name']`).
2. Le chemin qui est le nom sous lequel sera stocké le fichier de façon définitive. On peut utiliser le nom d'origine du fichier `$_FILES['Screenshot']['name']` ou générer un nom au hasard.

On va placer le fichier dans un sous-dossier « Uploads ». On gardera le même nom de fichier que celui d'origine.

Comme `$_FILES['Screenshot']['name']` contient le chemin entier vers le fichier d'origine (`C:\dossier\fichier.png` , par exemple), il nous faudra extraire le nom du fichier.

On peut utiliser pour cela la fonction `basename` qui renverra juste « `fichier.png` » :

```
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['Screenshot']) && $_FILES['Screenshot']['error'] == 0)
{
    // Testons si le fichier n'est pas trop gros
    if ($_FILES['Screenshot']['size'] <= 1000000)
    {
        // Testons si l'extension est autorisée
        $fileInfo = pathinfo($_FILES['Screenshot']['name']);
        $extension = $fileInfo['extension'];
        $allowedExtensions = ['jpg', 'jpeg', 'gif', 'png'];
        if (in_array($extension, $allowedExtensions))
        {
            // On peut valider le fichier et le stocker
            // définitivement
            move_uploaded_file($_FILES['Screenshot']['tmp_name'],
                'uploads/' . basename($_FILES['Screenshot']['name']));
            echo "L'envoi a bien été effectué !";
        }
    }
}
```

Remarques :

- Il faut que le dossier `Uploads` existe sur le serveur et qu'il ait les droits d'écriture (Mettre les droits à 733 avec le client FTP pour que PHP puisse enregistrer les fichiers),
- Il vaut mieux redéfinir les noms de fichiers pour éviter qu'il y ait 2 fichiers avec le même nom, sinon l'ancien serait écrasé. On peut par exemple utiliser un compteur,
- Il faut toujours restreindre les extensions autorisées pour éviter les fichiers malveillants et en particulier les fichiers PHP.

Exercice : Créer une copie de vos fichiers `contact.php` et `submit_contact.php` de manière à rajouter l'envoi d'une image, et affichez la sur la page `submit_contact`.

XVI. La variable superglobale `$_SERVER`

La superglobale `$_SERVER` contient des variables définies par le serveur utilisé ainsi que des informations relatives au script.

Cette superglobale est à nouveau un tableau associatif dont les clefs sont les noms des variables qu'elle stocke et les valeurs sont les valeurs des variables liées.

Voici quelques clefs qu'il peut être intéressant de connaître :

```
<?php
    // Renvoie le nom du script courant
    echo $_SERVER['PHP_SELF']. '<br>';

    // Renvoie le nom du serveur qui héberge le fichier
    echo $_SERVER['SERVER_NAME']. '<br>';

    // Renvoie l'adresse IP du serveur
    echo $_SERVER['SERVER_ADDR']. '<br>';

    // Renvoie l'adresse IP du visiteur
    echo $_SERVER['REMOTE_ADDR']. '<br>';

    // Renvoie une valeur non vide
    // si le script a été appelé via le protocole HTTPS.
    echo $_SERVER['HTTPS']. '<br>';

    // Renvoie le temps Unix du début de la requête
    echo $_SERVER['REQUEST_TIME']. '<br>';
?>
```

XVII. La variable superglobale `$_REQUEST`

La variable superglobale `$_REQUEST` va contenir toutes les variables envoyées via HTTP GET, HTTP POST et par les cookies HTTP.

Cette variable, qui est un tableau associatif, va ainsi contenir les variables de `$_GET`, `$_POST` et `$_COOKIE`.

XIX. Les sessions

Jusqu'ici, nous étions parvenus à passer des variables de page en page à l'aide d'URL ou de formulaires. On sait ainsi envoyer d'une page à une autre le nom et le prénom du visiteur. Mais dès qu'on charge une autre page, ces informations sont « oubliées ». C'est pour cela qu'on a inventé les sessions.

Les sessions permettent de conserver des variables sur toutes les pages de votre site.

Les sessions se gèrent en trois étapes.

XIX.1. création d'une session

1. Un visiteur arrive sur votre site.
2. On demande à créer une session pour lui.
3. PHP génère alors un numéro unique.

Ce numéro est souvent très grand. Exemple : a02bbfffc6198e6e0cc2715047bc3766f.

On génère une session grâce à la fonction **session_start()**.

Ce numéro sert d'identifiant ; c'est ce qu'on appelle un « ID de session » ou **PHPSESSID** .

PHP transmet automatiquement cet ID de page en page, en utilisant généralement un cookie.

XIX.2. création de variables pour la session

Une fois la session générée, on peut créer une infinité de variables de session pour nos besoins.

Par exemple, on peut créer :

- une variable qui contient le nom du visiteur : `$_SESSION['nom']`
- une autre qui contient son prénom : `$_SESSION['prenom']`
- une pour l'email : `$_SESSION[email]`
- etc.

Le serveur conserve ces variables même lorsque la page PHP a fini d'être générée. Autrement dit : quelle que soit la page de votre site, vous pourrez récupérer le nom et le prénom du visiteur via la superglobale `$_SESSION`.

XIX.3. suppression de la session

Lorsque le visiteur se déconnecte de votre site, la session est fermée et PHP « oublie » alors toutes les variables de session que vous avez créées.

Il est en fait difficile de savoir précisément quand un visiteur quitte votre site. En effet, lorsqu'il ferme son navigateur ou va sur un autre site, le vôtre n'en est pas informé.

Soit le visiteur clique sur un bouton « Déconnexion » (que vous aurez créé) avant de s'en aller, soit on attend quelques minutes d'inactivité pour le déconnecter automatiquement : on parle alors de "timeout". Le plus souvent, le visiteur est déconnecté par un timeout.

Pour activer ou détruire une session, deux fonctions sont à connaître :

1. **session_start()** : démarre le système de sessions. Si le visiteur vient d'arriver sur le site, alors un numéro de session est généré pour lui.
2. **session_destroy()** : ferme la session du visiteur. Cette fonction est automatiquement appelée lorsque le visiteur ne charge plus de page de votre site pendant plusieurs minutes (c'est le timeout), mais vous pouvez aussi créer une page « Déconnexion » si le visiteur souhaite se déconnecter manuellement.

Il faut appeler **session_start()** sur chacune de vos pages AVANT d'écrire le moindre code HTML ou PHP (avant même la balise **<!DOCTYPE>**).

Si vous oubliez de lancer **session_start()** , vous ne pourrez pas accéder à la variable superglobale **\$_SESSION**

On peut aussi supprimer une valeur en particulier dans la session avec **unset(\$_SESSION['exemple'])**

Les sessions servent dans de nombreux cas sur le web. Voici quelques exemples :

- Imaginez un script qui demande un identifiant et un mot de passe pour qu'un visiteur puisse se « connecter » (s'authentifier). On peut enregistrer ces informations dans des variables de session et se souvenir de l'identifiant du visiteur sur toutes les pages du site !
- Puisqu'on retient son identifiant et que la variable de session n'est créée que s'il a réussi à s'authentifier, on peut l'utiliser pour restreindre certaines pages de notre site à certains visiteurs uniquement. Cela permet de créer toute une zone d'administration sécurisée : si la variable de session login existe, on affiche le contenu, sinon on affiche une erreur. Cela devrait vous rappeler l'exercice sur la protection d'une page par mot de passe, sauf qu'ici, on peut se servir des sessions pour protéger automatiquement plusieurs pages.
- On se sert activement des sessions sur les sites de vente en ligne. Cela permet de gérer un « panier » : on retient les produits que commande le client quelle que soit la page où il est. Lorsqu'il valide sa commande, on récupère ces informations et... on le fait payer.

XX. Les cookies

Travailler avec des cookies revient à peu près à la même chose qu'avec des sessions, à quelques petites différences près.

XX.1.1. fonctionnement d'un cookie

Un cookie, c'est un petit fichier que l'on enregistre sur l'ordinateur du visiteur.

Ce fichier contient du texte et permet de « retenir » des informations sur le visiteur.

Par exemple, vous inscrivez dans un cookie le pseudo du visiteur. Comme ça, la prochaine fois qu'il viendra sur votre site, vous pourrez lire son pseudo en allant regarder ce que son cookie contient.

On fait souvent l'erreur de penser que les cookies sont « dangereux ».

Ce ne sont pas des virus, juste de petits fichiers texte qui permettent de retenir des informations.

Au pire, un site marchand peut retenir que vous aimez les appareils photos numériques et vous afficher uniquement des pubs pour des appareils photos, mais c'est tout ; ces petites bêtes sont inoffensives pour votre ordinateur.

Chaque cookie stocke généralement une information à la fois.

Si vous voulez stocker le pseudonyme du visiteur et sa date de naissance, il est donc recommandé de créer deux cookies.

Les cookies sont stockées sur le disque dur du visiteur et leur emplacement varie en fonction du navigateur utilisé.

XX.1.2. Écrire un cookie

Comme une variable, un cookie a un nom et une valeur.

Pour écrire un cookie, on utilise la fonction PHP setcookie (qui signifie « Placer un cookie », en anglais).

On lui donne en général trois paramètres, dans l'ordre suivant :

- Le nom du cookie (exemple : LOGGED_USER).
- La valeur du cookie (exemple : utilisateur@example.com).
- La date d'expiration du cookie, sous forme de "timestamp" (exemple : 1090521508).

Si vous voulez supprimer le cookie dans un an, il vous faudra donc écrire :`time() + 365*24*3600`

Cela aura pour effet de supprimer votre cookie dans exactement un an.

XX.1.3. Sécuriser un cookie

Pour sécuriser un cookie, nous allons configurer les options `httpOnly` et `secure` sur le cookie.

Sans rentrer dans les détails, cela rendra votre cookie inaccessible en JavaScript sur tous les navigateurs qui supportent cette option (c'est le cas de tous les navigateurs récents). Cette option permet de réduire drastiquement les risques de faille XSS sur votre site, au cas où vous auriez oublié d'utiliser `htmlspecialchars` à un moment.

Voici comment créer un cookie de façon sécurisée :

```
// retenir l'email de la personne connectée pendant 1 an
setcookie(
    'LOGGED_USER',
    'utilisateur@example.com',
    [
        'expires' => time() + 365*24*3600,
        'secure' => true,
        'httponly' => true,
    ]
);
```

Une particularité notable de la fonction `setcookie()` est qu'il va falloir l'appeler avant d'écrire tout code HTML pour qu'elle fonctionne puisque les cookies doivent être envoyés avant toute autre sortie.

XX.1.4. Affichez et récupérez un cookie

Avant de commencer à travailler sur une page, PHP lit les cookies du client pour récupérer toutes les informations qu'ils contiennent. Ces informations sont placées dans la superglobale `$_COOKIE` sous forme d'un tableau (array).

De ce fait, si je veux ressortir l'e-mail du visiteur que j'avais inscrit dans un cookie, il suffit d'écrire : `$_COOKIE['LOGGED_USER']`

Ce qui nous donne un code PHP très simple pour afficher de nouveau le pseudo du visiteur :

```
Bonjour <?php echo strval($_COOKIE['LOGGED_USER']); ?> !
```

À noter que si le cookie n'existe pas, la variable superglobale n'existe pas. Il faut donc faire un `isset` pour vérifier si le cookie existe ou non.

Remarque :

Les cookies viennent du visiteur. Comme toute information qui vient du visiteur, **elle n'est pas sûre**. N'importe quel visiteur peut créer des cookies et envoyer ainsi de fausses informations à votre site. Il est donc impératif de traiter les informations provenant de cookies.

XX.1.5. Modifiez un cookie existant

Il suffit appeler à `setcookie` en gardant le même nom de cookie, ce qui « écrasera » l'ancien.

Par exemple, si c'est Laurène Castor qui se connecte au site, je ferai :

```
setcookie(
    'LOGGED_USER',
    'laurene.castor@exemple.com',
    [
        'expires' => time() + 365*24*3600,
        'secure' => true,
        'httponly' => true,
    ]
);
```

Notez qu'alors le temps d'expiration du cookie est remis à zéro pour un an.

XX.1.6. Supprimer un cookie :

Pour supprimer un cookie, on va de nouveau faire appel à `setcookie()`, et définir une date d'expiration dans le passé :

```
setcookie(
    'LOGGED_USER',
    '',
    [
        'expires' => time() - 3600,
    ]
);
```

XXI. Travailler avec une base de données

XXI.1. Découverte des bases de données

La **base de données (BDD)** est un système qui enregistre des informations.

Ces informations sont toujours organisées. Et c'est ça qui fait que la BDD est si pratique : c'est un moyen simple de ranger des informations.

Une base de données, c'est un peu comme une armoire dans laquelle chaque dossier est à sa place.

Quand tout est à sa place, il est beaucoup plus facile de retrouver l'information. En classant les informations que vous collectez (concernant vos visiteurs, par exemple), il vous sera très facile de récupérer plus tard ce que vous cherchez.

Pour interagir avec une BDD, nous avons besoin d'utiliser un **Système de Gestion de Base de Données (SGBD)**. Nous utiliserons le SGBD **MySQL**, mais sachez que l'essentiel de ce que vous allez apprendre fonctionnera de la même manière avec un autre SGBD.

XXI.1.1. Le langage SQL

Vous allez devoir communiquer avec le SGBD pour lui donner l'ordre de récupérer ou d'enregistrer des données. Pour lui "parler", on utilise le langage **SQL**.

La bonne nouvelle, c'est que le langage SQL est un standard, c'est-à-dire que quel que soit le SGBD que vous utilisez, vous vous servirez du langage SQL. La mauvaise, c'est qu'il y a en fait quelques petites variantes d'un SGBD à l'autre, mais cela concerne généralement les commandes les plus avancées.

Comme vous vous en doutez, il va falloir apprendre le langage SQL pour travailler avec les bases de données. Ce langage n'a rien à voir avec le PHP, mais nous allons impérativement en avoir besoin.

Voici un exemple de commande en langage SQL, pour vous donner une idée :

```
SELECT id, auteur, message, datemsg FROM livreor ORDER BY datemsg DESC
```

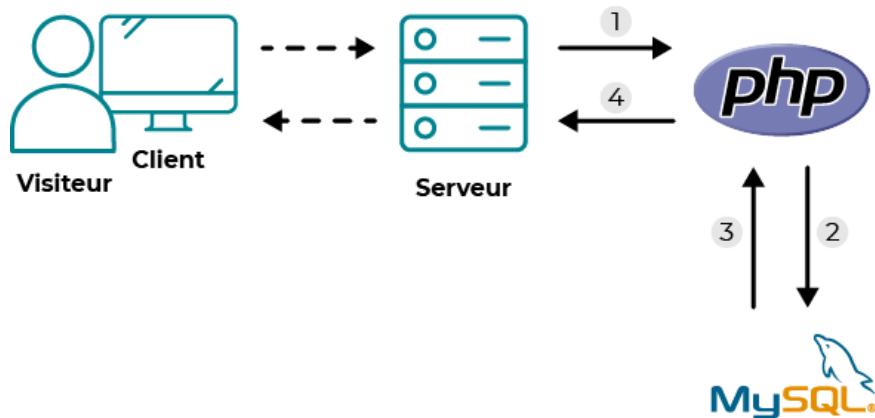
Le principal objectif de cette partie du cours sera d'apprendre les instructions nécessaires à écrire en PHP pour effectuer des requêtes en base de données, et les bases du langage SQL.

XXI.1.2. Comment PHP fait le lien entre vous et MySQL

Ce qu'il faut comprendre, c'est que l'on ne va pas pouvoir parler à MySQL directement, seul PHP peut le faire...

C'est donc PHP qui va faire l'intermédiaire entre vous et MySQL. On devra demander à PHP : "Va dire à MySQL de faire ceci".

Voici un schéma pour illustrer cela :



Voici ce qui peut se passer lorsque le serveur a reçu une demande d'un client qui veut poster un message :

1. Le serveur utilise toujours PHP, il lui fait donc passer le message.
2. PHP effectue les actions demandées et se rend compte qu'il a besoin de MySQL. En effet, le code PHP contient à un endroit "Va demander à MySQL d'enregistrer ce message". Il fait donc passer le travail à MySQL.
3. MySQL fait le travail que PHP lui a soumis et lui répond "OK, c'est bon !".
4. PHP renvoie au serveur que MySQL a bien fait ce qui lui était demandé.

Maintenant, il va falloir découvrir comment est organisée une base de données. Bien en comprendre l'organisation est en effet absolument indispensable.

XXI.2. Structurez votre base de données

Avec les bases de données, il faut utiliser un vocabulaire **précis**. Pour mieux comprendre, on va réutiliser l'image de l'armoire.

Imaginez ceci :

- L'armoire est appelée "**la base**" dans le langage SQL. C'est le gros meuble dans lequel on classe les informations.
- Dans une armoire, il y a plusieurs tiroirs. Un tiroir, en SQL, c'est ce qu'on appelle "**une table**". Chaque tiroir contient des données différentes. Par exemple, on peut imaginer un tiroir qui contient les pseudonymes et informations sur vos visiteurs, un autre qui contient les messages postés sur votre forum, etc.
- Mais que contient une table ? C'est là que sont enregistrées les données, sous la forme d'un tableau. Dans ce tableau, les colonnes sont appelées **des "champs"**, et les lignes sont appelées **des "entrées"**.

Une table est donc représentée sous la forme d'un tableau.

Par exemple, le tableau suivant vous montre à quoi peut ressembler le contenu d'une table appelée « Utilisateurs » :

ID	FullName	Email	Password
1	Gérard MANFAIM	gerard.manfaim@gmail.com	P@ssw0rd
2	Bill KILL	kill.bill@yahoo.fr	s3cr3t
3	Eta GERE	eta.gere@outlokk.fr	123456
4

(sauf qu'on ne stockera jamais de mot de passe en clair 🔑)

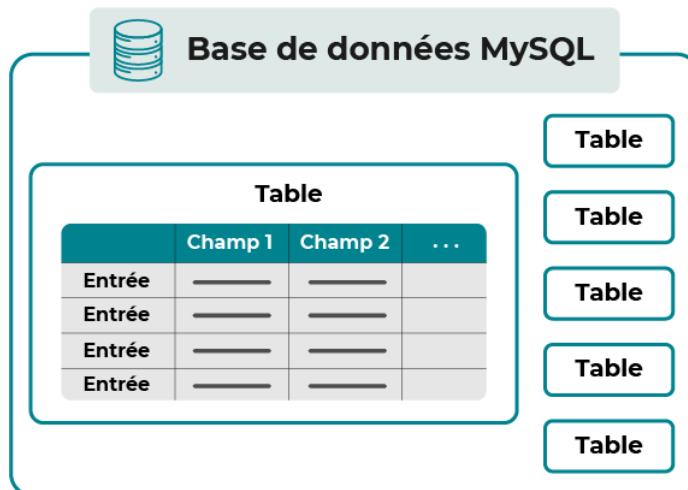
Ce tableau représente le contenu d'une table (c'est-à-dire le tiroir de l'armoire).

Chaque ligne est une entrée. Ici, il y en a quatre, mais une table peut très bien contenir 100, 1 000, ou même 100 000 entrées.

Très souvent, on crée un champ appelé "ID" (identifiant).

Comme nous le verrons plus tard, il est très pratique de numérotter ses entrées, même si ce n'est pas obligatoire.

Un autre schéma pour que tout ça soit clair :



Organisation d'une base de données MySQL

La base de données contient plusieurs tables (on peut en mettre autant que l'on veut à l'intérieur).

Rappel : chaque table est en fait un tableau où les colonnes sont appelées champs et où les lignes sont appelées entrées.

XXI.2.1. Comment sont enregistrées les données

Voici une question que l'on se pose fréquemment, quand on aborde ce genre de chapitre sur les bases de données pour la première fois :

Ils sont bien jolis ces tableaux et ces schémas, ces bases, ces champs... Mais je ne vois pas ce que c'est concrètement ! Où est-ce que MySQL enregistre les données ?

En fait, tout ce que l'on vient de voir, c'est une façon de « visualiser » la chose. Il faut que vous imaginiez que la base de données gère les informations sous forme de tableaux, parce que c'est la meilleure représentation qu'on puisse s'en faire.

Mais concrètement, quand MySQL enregistre des informations, il les écrit bien quelque part. Et comme tout le monde, il les enregistre **dans des fichiers** !

Ces fichiers sont quelque part sur votre disque dur, mais il ne faut jamais les ouvrir et encore moins les modifier directement. Il faut toujours parler avec MySQL qui va se charger d'extraire et de modifier les informations dans ces fichiers.

Chaque SGBD a sa propre façon d'enregistrer les données, mais aucun d'eux ne peut y échapper : pour que les données restent enregistrées, il faut les stocker dans des fichiers sur le disque dur.

Par exemple, avec MySQL sous Windows, si vous utilisez WAMP, vous devriez trouver les fichiers où sont stockées les informations dans C:\wamp\mysql\data .

C'est le même principe si vous utilisez MAMP, XAMPP... : les fichiers de la base de données sont bien quelque part !

Il ne faut jamais toucher à ces fichiers directement. On demandera TOUJOURS à MySQL d'enregistrer, ou d'aller lire des choses. Après, c'est lui qui se débrouille pour classer ça comme il veut dans ses fichiers.

Et c'est justement ça, le gros avantage de la base de données : pas de prise de tête pour le rangement des informations. Vous demandez à MySQL de vous sortir tous les commentaires d'une recette de votre site enregistrées de février à juillet : il va lire dans ses fichiers, et vous ressort les réponses. Vous vous contentez de "dialoguer" avec MySQL. Lui se charge de ranger vos données dans ses fichiers.

XXII. Mettez en place une base de données avec phpMyAdmin

Pour communiquer avec MySQL, nous pouvons utiliser différents logiciels : de l'invite de commandes jusqu'à des logiciels accessibles par le navigateur. Ici nous allons utiliser **phpMyAdmin**, l'un des outils les plus connus permettant de manipuler une base de données MySQL.

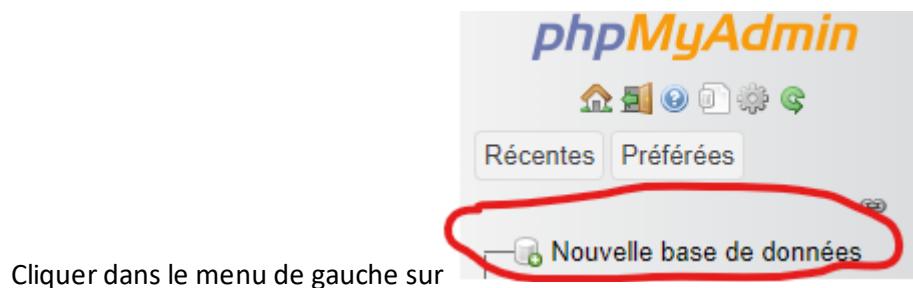
XXII.1. Créer une table

Nous allons créer une première base de données avec une table contenant les informations du groupe de stagiaires de la formation.

Dans phpMyAdmin, accessible à l'adresse : (sous WAMP)

<http://localhost/phpmyadmin/>

identifiant par défaut : root, pas de mot de passe,



Cliquer dans le menu de gauche sur
Sur la page création d'une base de données, donner un nom « formation » et choisir l'interclassement « utf8_general_ci » puis cliquer sur créer :

Bases de données

This screenshot shows a sub-menu titled "Création d'une base de données". It contains two input fields: one for the database name ("formation") and another for the character set ("utf8_general_ci"). To the right of these fields is a "Créer" button.

Sur l'écran suivant choisir le nom de la table « etudiants », et le nombre de colonnes, 5 puis cliquer sur créer :

This screenshot shows a sub-menu titled "Créer une nouvelle table". It has two input fields: "Nom de table" containing "stagiaires" and "Nombre de colonnes" containing "5". To the right of these fields is a "Créer" button.

Nous allons devoir associer chacune des informations de nos étudiants à un champ de la table, et lui associer un type de données. Et, nous devons définir un champ qui servira d'identifiant unique (un peu comme un numéro de Sécurité sociale), de sorte à pouvoir retrouver chaque personne :

Nom	Type	Taille/Valeurs*	Valeur par défaut	Interclassement	Attributs	Null	Index	Actions	Commentaires
id	INT		Aucun(e)			<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>	
nom	VARCHAR	50	Aucun(e)			<input type="checkbox"/>	---	<input type="checkbox"/>	
prenom	VARCHAR	50	Aucun(e)			<input type="checkbox"/>	---	<input type="checkbox"/>	
age	INT		Aucun(e)			<input type="checkbox"/>	---	<input type="checkbox"/>	
ville	VARCHAR	50	Aucun(e)			<input type="checkbox"/>	---	<input type="checkbox"/>	

Commentaires de table : Interclassement : Moteur de stockage : MyISAM

Définition de PARTITION :

Partitionner par : (Expression ou liste de colonn)

Partitions :

[Aperçu SQL](#) **Enregistrer**

XXII.1.1. Les types de champs MySQL

Alors que PHP ne propose que quelques types de données que l'on connaît bien maintenant (int , string , bool ...), MySQL propose une quantité très importante de types de données.

Mais dans la pratique, vous n'aurez besoin de jongler qu'entre les quatre types de données suivants :

1. INT : nombre entier ;
2. VARCHAR : texte court (entre 1 et 255 caractères) ;
3. TEXT : long texte (on peut y stocker un roman sans problème) ;
4. DATE : date (jour, mois, année).

Cela couvrira 99 % de vos besoins, et avec l'expérience vous apprendrez à optimiser vos bases de données, et l'intérêt des autres types de données de MySQL.

XXII.1.2. Les clés primaires

Toute table doit posséder un champ qui joue le rôle de **clé primaire**. La clé primaire permet d'identifier de manière unique une entrée dans la table. En général, on utilise le champ id comme clé primaire, comme on vient de le faire.

Prenez le réflexe de créer à chaque fois ce champ « id » en lui donnant l'index PRIMARY , ce qui aura pour effet d'en faire une clé primaire.

Vous en profiterez en général pour cocher la case AUTO_INCREMENT pour que ce champ gère lui-même les nouvelles valeurs automatiquement (1, 2, 3, 4...).

XXII.2. Modifier une table

Pour ajouter des données dans la table on va cliquer sur l'onglet « Insérer » :



Et ajouter nos informations sur les étudiants :

Colonne	Type	Fonction	Null	Valeur
id	int(11)			
nom	varchar(50)			BULCKAEN
prenom	varchar(50)			Aline
age	int(11)			45
ville	varchar(50)			HAMBACH

On laisse le champs id vide puisqu'on l'a défini en auto-increment, et on clique sur exécuter, et on continue jusqu'à avoir renseigner tous les stagiaires.

Par l'onglet structure, on va pouvoir ajouter ou supprimer des colonnes, modifier les types de champs etc :

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	id	int(11)			Non	Aucun(e)		AUTO_INCREMENT	Modifier Supprimer Plus
2	nom	varchar(50)	utf8_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
3	prenom	varchar(50)	utf8_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
4	age	int(11)			Non	Aucun(e)			Modifier Supprimer Plus
5	ville	varchar(50)	utf8_general_ci		Non	Aucun(e)			Modifier Supprimer Plus

↑ Tout cocher Avec la sélection : Parcourir Modifier Supprimer Primaire Unique Index Spatial Texte entier

Imprimer Suggérer des optimisations de structure Déplacer des colonnes Normaliser

Ajouter 1 colonne(s) après ville Exécuter

Index

Action	Nom de l'index	Type	Unique	Compressé	Colonne	Cardinalité	Interclassement	Null	Commentaire
Éditer Renommer Supprimer	PRIMARY	BTREE	Oui	Non	id	1	A	Non	

Créer un index sur 1 colonnes Exécuter

XXII.3. Importez et exportez des données

Nous allons maintenant nous intéresser à l'onglet "Import" de phpMyAdmin, dont le principal intérêt est de créer une base de données entière avec tables et données :

Importation dans le serveur courant

Fichier à importer :

Le fichier peut être compressé (gzip, bzip2, zip) ou non.
Le nom du fichier compressé doit se terminer par .[format].[compression]. Exemple : .sql.zip
Parcourir les fichiers : (Taille maximale : 128Mio)

Choisir un fichier Aucun fichier choisi

Il est également possible de glisser-déposer un fichier sur n'importe quelle page.

Jeu de caractères du fichier :

utf-8

Il suffit de cliquer sur « choisir un fichier », de rechercher le fichier SQL à importer et de cliquer en

Importer

bas de la page sur

Regardons maintenant l'onglet « Exporter » de phpMyAdmin.

Exportation des tables depuis la base de données « formation »

The screenshot shows the 'Exporter' tab interface. At the top, there's a section titled 'Méthode d'exportation:' with two radio button options: 'Rapide, n'afficher qu'un minimum d'options' (selected) and 'Personnalisée, afficher toutes les options possibles'. Below this is a 'Format:' section with a dropdown menu set to 'SQL'. At the bottom right is a large 'Exporter' button.

C'est ici que vous allez pouvoir récupérer votre base de données sur le disque dur, sous forme de fichier texte .sql (qui contiendra des tonnes de requêtes SQL).

Le fichier que vous allez obtenir grâce à « l'exportation » de phpMyAdmin, c'est un fichier qui dit à MySQL comment recréer votre base de données (avec des requêtes en langage SQL). Et ce fichier-là est tout à fait lisible.

On peut se servir de ce fichier pour deux choses :

- **Transmettre votre base de données sur Internet** : pour le moment, votre base de données se trouve sur votre disque dur. Mais lorsque vous voudrez héberger votre site sur Internet, il faudra utiliser la base de données en ligne de votre hébergeur ! Le fichier .sql que vous allez générer vous permettra de **reconstruire la base de données à l'identique**.
- **Faire une copie de sauvegarde de la base de données** : on ne sait jamais, si vous faites une bêtise ou si quelqu'un réussit à détruire toutes les informations sur votre site, vous serez bien content d'avoir une copie de secours sur votre disque dur !

XXIII. Accédez aux données en PHP avec PDO

XXIII.1. Connectez-vous à la base de données en PHP

Pour pouvoir travailler avec la base de données en PHP, il faut d'abord s'y connecter.

Il va donc falloir que PHP s'authentifie : on dit qu'il établit une connexion avec MySQL.

Une fois que la connexion sera établie, vous pourrez faire toutes les opérations que vous voudrez sur votre base de données.

Pour se connecter à une base de données MySQL, vous allez devoir utiliser une extension PHP appelée **PDO ("PHP Data Objects")**. Cette extension est fournie avec PHP (en français, "les fonctions PDO sont à votre disposition"), mais parfois il vous faudra activer l'extension.

XXIII.1.1. Connectez PHP à MySQL avec PDO

Nous allons avoir besoin de quatre renseignements :

- **Le nom de l'hôte** : c'est l'adresse IP de l'ordinateur où MySQL est installé. Le plus souvent, MySQL est installé sur le même ordinateur que PHP : dans ce cas, mettez la valeur **localhost** .
- **La base** : c'est le nom de la base de données à laquelle vous voulez vous connecter. Dans notre cas, la base s'appelle **formation** .
- **L'identifiant et le mot de passe** : ils permettent de vous identifier. Renseignez-vous auprès de votre hébergeur pour les connaître.

Sur WAMP, la valeur de l'identifiant est root et le mot de passe rest vide.

Voici donc l'instruction PDO pour vous connecter à votre base formation :

```
<?php
// Souvent on identifie cet objet par la variable $conn ou $db
$db = new PDO(
    'mysql:host=localhost;dbname=formation;charset=utf8' ,
    'root' ,
    ''
);
?>
```

La ligne de code qu'on vient de voir crée une connexion à la base de données.

En fait, on crée la connexion en indiquant dans l'ordre dans les paramètres :

- le nom d'hôte : **localhost** ;
- la base de données : **formation** ;
- l'identifiant (login) : **root** ;
- le mot de passe (rappel : sous WAMP, le mot de passe est vide).

Lorsque votre site sera en ligne, vous aurez sûrement un nom d'hôte différent, ainsi qu'un identifiant et un mot de passe, comme ceci :

```
<?php  
$db = new PDO('mysql:host=sql.hebergeur.com;dbname=mabase;charset=utf8',  
'pierre.durand', 's3cr3t');  
?>
```

Il faudra donc penser à changer cette ligne pour l'adapter à votre hébergeur en modifiant les informations en conséquence, lorsque vous enverrez votre site sur le Web.

Le premier paramètre (qui commence par mysql) s'appelle le **DSN : Data Source Name**. C'est généralement le seul qui change en fonction du type de base de données auquel on se connecte.

XXIII.1.2. Testez la présence d'erreurs

Si vous avez renseigné les bonnes informations (nom de l'hôte, de la base, login et mot de passe), rien ne devrait s'afficher à l'écran.

Toutefois, s'il y a une erreur (vous vous êtes trompé de mot de passe ou de nom de base de données, par exemple), PHP risque d'afficher toute la ligne qui pose l'erreur, ce qui inclut le mot de passe !

Vous ne voudrez pas que vos visiteurs puissent voir le mot de passe si une erreur survient lorsque votre site est en ligne. Il est préférable de **traiter** l'erreur.

En cas d'erreur, PDO renvoie ce qu'on appelle une **exception**, qui permet de « capturer » l'erreur.

```
<?php  
try{  
    $db = new PDO('mysql:host=localhost;dbname=formation;charset=utf8',  
'root', '' );  
}  
catch (Exception $e){  
    die('Erreur : ' . $e->getMessage());  
}  
?>
```

Voilà encore un code un peu nouveau pour nous : sans trop rentrer dans le détail, il faut savoir que PHP essaie d'exécuter les instructions à l'intérieur du bloc **try** :

- S'il y a une erreur, il rentre dans le bloc **catch** et fait ce qu'on lui demande (ici, on arrête l'exécution de la page en affichant un message décrivant l'erreur).
- Si au contraire tout se passe bien, PHP poursuit l'exécution du code et ne lit pas ce qu'il y a dans le bloc **catch** . Votre page PHP ne devrait donc rien afficher pour le moment.

XXIII.2. Effectuez une première requête SQL

L'objectif ici consiste à récupérer la liste des stagiaires qui sont maintenant stockées dans votre base de données.

Arrive alors le grand moment que vous attendiez tous : on va parler à MySQL. Pour cela, on va faire ce qu'on appelle une requête, en demandant à MySQL de nous dire ce que contient la table stagiaires.

XXIII.2.1. Construisez votre première requête SQL

Le SQL est un langage. C'est lui qui nous permet de communiquer avec MySQL.

Voici la première requête SQL que nous allons utiliser :

```
SELECT * FROM etudiants
```

Cela peut se traduire par :

« Prendre tout ce qu'il y a dans la table etudiants ».

Analysons chaque terme de cette requête.

- **SELECT** : en langage SQL, le premier mot indique quel type d'opération doit effectuer MySQL. Ce mot-clé demande à MySQL d'afficher ce que contient une table.
- ***** : après le SELECT, on doit indiquer quels champs MySQL doit récupérer dans la table.
 - Si on n'est intéressé que par les champs « nom » et « prenom », il faudra taper :
SELECT nom, prenom FROM stagiaires
 - Si vous voulez prendre tous les champs, tapez ***** . Cette petite étoile peut se traduire par « tout » : « Prendre tout ce qu'il y a... ».
- **FROM** : c'est un mot de liaison qui se traduit par « dans ». **FROM** fait la liaison entre le nom des champs et le nom de la table.
- **etudiants** : c'est le nom de la table dans laquelle il faut aller piocher.

Effectuons la requête à l'aide de l'objet PDO :

```
<?php  
$etudiantsStatement = $db->prepare('SELECT * FROM etudiants');  
?>
```

XXIII.2.2. Affichez le résultat d'une requête SQL

Le problème, c'est que **\$etudiantsStatement** contient quelque chose d'inexploitable directement : un objet **PDOStatement**. Cet objet va contenir la requête SQL que nous devons exécuter, et par la suite, les informations récupérées en base de données.

Pour récupérer les données, demandez à cet objet d'exécuter la requête SQL et de récupérer toutes les données dans un format "exploitable" pour vous, c'est-à-dire sous forme d'un tableau PHP.

```
$etudiantsStatement->execute();
$etudiants = $etudiantsStatement->fetchAll();
```

Une fois qu'on a récupéré les données sous forme d'un tableau PHP, on en revient à ce que vous connaissez déjà ! Pour résumer :

```
<?php
try{
    // On se connecte à MySQL
    $db = new PDO('mysql:host=localhost;dbname=formation;charset=utf8',
'root', '');

}
catch(Exception $e){
    // En cas d'erreur, on affiche un message et on arrête tout
    die('Erreur : '.$e->getMessage());
}

// Si tout va bien, on peut continuer

// On récupère tout le contenu de la table etudiants
$sqlQuery = 'SELECT * FROM etudiants';
$etudiantsStatement = $db->prepare($sqlQuery);
$etudiantsStatement->execute();
$etudiants = $etudiantsStatement->fetchAll();

// On affiche chaque etudiant un à un
foreach ($etudiants as $etudiant) {
?>
    <p><?php echo $etudiant ['nom']; ?></p>
<?php
}
?>
```

XXIII.2.3. Affichez seulement le contenu de quelques champs

Maintenant vous devriez être capable d'afficher ce que vous voulez.

Personne ne vous oblige à afficher tous les champs !

Par exemple, si l'on veut juste lister les noms et les prénoms, on utilise la requête SQL suivante :

```
<?php
$sqlQuery = 'SELECT nom, prenom FROM etudiants';
```

Et... c'est tout, vous récupérerez seulement les informations dont vous avez besoin, ce qui est plus performant.

XXIII.3. Filtrez et trier vos données

En modifiant vos requêtes SQL, il est possible de filtrer et trier très facilement vos données. Vous allez ici découvrir les mots-clés suivants du langage SQL :

- **WHERE** ;
- **ORDER BY** ;
- **LIMIT** .

XXIII.3.1. Filtrer les résultats

Par exemple, si l'on veut n'afficher que les étudiants de 30 ans et plus, on utilise la requête suivante :

```
<?php  
$sqlQuery = 'SELECT * FROM etudiants WHERE age >= 30';
```

WHERE permet donc de filtrer les résultats d'une requête selon une **condition**.

Pour créer ces conditions, le SQL utilise des opérateurs de comparaisons, voici les plus courants :

Opérateur	Description
=	Égale
<>	Pas égale
!=	Pas égale
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égale à
<=	Inférieur ou égale à
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donnée (utile pour les nombres ou dates)
LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot.
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle

On peut aussi créer des conditions combinées avec les opérateurs **AND**, **OR**, **NOT** et **XOR** de la même manière qu'en PHP.

(Voir la documentation officielle pour plus de détails : <https://sql.sh/>)

XXIII.3.2. Trier les résultats

Si l'on souhaite trier les résultats d'une requête, on utilisera la commande **ORDER BY** suivie du nom du champs sur lequel on effectue le tri, puis de des suffixes **ASC** (ascendant) ou **DESC** (descendant), pour spécifier le sens du tri. Par défaut, le tri est ascendant.

Par exemple si l'on veut trier les résultats par ordre alphabétique du nom, on utilisera la requête suivante :

```
$sqlQuery = 'SELECT * FROM etudiants ORDER BY nom ASC';
```

Et si l'on veut trier les résultats par ordre alphabétique inverse (en commençant par Z):

```
$sqlQuery = 'SELECT * FROM etudiants ORDER BY nom DESC';
```

XXIII.3.3. Afficher certains résultats

La clause LIMIT est à utiliser dans une requête SQL pour spécifier le nombre maximum de résultats que l'on souhaite obtenir.

Par exemple si l'on souhaite n'afficher que les 10 premiers résultats, on utilisera la requête suivante :

```
$sqlQuery = 'SELECT * FROM etudiants ORDER BY nom ASC LIMIT 10';
```

Cette clause est souvent associé à un OFFSET, c'est-à-dire effectuer un décalage sur le jeu de résultat. Ces 2 clauses permettent par exemple d'effectuer des système de pagination (exemple : récupérer les 10 articles de la page 4). L'OFFSET Commence toujours à 0

Dans notre cas si l'on veut n'afficher que les résultats de 6 à 15 on utilisera la requête suivante :

```
$sqlQuery = 'SELECT * FROM etudiants ORDER BY nom ASC LIMIT 5, 10';
```

Il faut utiliser les mots-clés dans l'ordre suivant : WHERE puis ORDER BY puis LIMIT , sinon MySQL ne comprendra pas votre requête.

XXIII.4. Utiliser des variables dans les requêtes

Les requêtes que vous avez exécutées jusqu'ici étaient simples et effectuaient toujours la même opération. Or, les choses deviennent intéressantes quand on utilise des variables de PHP dans les requêtes.

XXIII.4.1. Utiliser des marqueurs

Les marqueurs sont des identifiants reconnus par PDO pour être remplacés lors de la préparation de la requête par les variables PHP et qui sera de la forme :ville.

Par exemple si l'on veut prévoir dans notre requête la possibilité de filtrer les résultats par ville, en laissant la possibilité à l'utilisateur de choisir la ville on écrira notre requête ainsi :

```

$sqlQuery = 'SELECT * FROM etudiants WHERE ville = :ville';
$etudiantsStatement = $db->prepare($sqlQuery);
$etudiantsStatement ->execute(array(
    'ville' => $ville,
));

```

Ainsi la variable \$ville pourra être alimenté par un sélect dans un formulaire ou une valeur en GET.

On utilise ces requêtes préparées pour des raisons de sécurité. En effet, On ne concatène JAMAIS une requête SQL pour passer des variables, au risque de créer des injections SQL !

XXIII.4.2. Lier les variables avec la méthode bindValue()

Pour plus de sécurité dans la préparation de nos requêtes, plutôt que directement passer un array à execute(), nous allons utiliser bindValue() pour lier le marqueur à la valeur de notre variable.

La méthode bindValue() prends deux paramètres obligatoires et trois paramètres facultatifs, dont le premier nous sera particulièrement utile :

- Un identifiant (obligatoire) qui sera de la forme :nom si on utilise des marqueurs nommés ou qui sera l'index de base 1 du paramètre si on utilise un marqueur interrogatif ;
- Le nom de la variable PHP (obligatoire) à lier au paramètre de la requête SQL ;
- Le type de données explicite pour le paramètre (facultatif) spécifié en utilisant les constantes **PDO::PARAM_*constants**.

Les constantes prédéfinies les plus utilisées sont les suivantes :

- PDO ::PARAM_STR, qui représente le type de données CHAR, VARCHAR et les autres types de données « chaîne de caractères » SQL ;
- PDO ::PARAM_INT, qui représente le type de données SQL INTEGER (nombre entier) ;
- PDO ::PARAM_NULL, qui représente le type de données SQL NULL ;
- PDO ::PARAM_BOOL, qui représente le type de données booléen.

Notre requête devient donc :

```

$sqlQuery = 'SELECT * FROM etudiants WHERE ville = :ville';
$etudiantsStatement = $db->prepare($sqlQuery);
$etudiantsStatement ->bindValue(':ville', $ville, PDO::PARAM_STR);
$etudiantsStatement ->execute();

```

Bien entendu, on peut lier autant de variables que l'on veut de cette façon.

Liste des stagiaires

ID	Prénom	Nom	Age	Ville
6	Simon	HELLERINGER	25	SARREGUEMINES
10	Jocelin	SCHWARTZ	29	SARREGUEMINES

XXIV. Ajouter, modifier et supprimer des données

Après avoir vu comment lire les données avec SELECT, il nous reste encore à voir comment ajouter, modifier et supprimer des données pour que notre fameux **CRUD** soit complet.

Pour cela nous allons utiliser de nouvelles requêtes : **INSERT**, **UPDATE** et **DELETE**.

Pour cela, nous allons créer une nouvelle table dans la base **formation** que nous appellerons **users** avec les champs suivants :

- « id », type INT, PRIMARY KEY, AUTO_INCREMENT,
- « prenom », type VARCHAR(30),
- « nom », type VARCHAR(30),
- « mail », type VARCHAR(30).

XXIV.1. Ajouter des données

Pour ajouter une entrée, on utilise une requête **INSERT**. Voici un exemple qui ajoute un utilisateur :

```
<?php  
$query = 'INSERT INTO users(prenom, nom, mail) VALUES (:prenom, :nom, :mail)';
```

Étudions un peu cette requête.

- D'abord, vous devez commencer par les mots-clés **INSERT INTO** qui indiquent que vous voulez insérer une entrée.
- Vous précisez ensuite le nom de la table (ici **users**), puis listez entre parenthèses les noms des champs dans lesquels vous souhaitez placer des informations (sauf le champs id qui est en auto-incrémentation).
- Enfin vous inscrivez **VALUES** suivi des valeurs à insérer **dans le même ordre que les champs que vous avez indiqués**.

Utilisez cette requête SQL au sein d'un script PHP :

```
<?php
try{
    // On se connecte à MySQL
    $db = new PDO('mysql:host=localhost;dbname=formation;charset=utf8',
    'root', '');
}
catch(Exception $e){
    // En cas d'erreur, on affiche un message et on arrête tout
    die('Erreur : '.$e->getMessage());
}

// Si tout va bien, on peut continuer

$query = 'INSERT INTO users(prenom, nom, mail) VALUES (:prenom, :nom,
:mail)';
$req = $db->prepare($query);
$req->bindValue(':prenom', $prenom, PDO::PARAM_STR);
$req->bindValue(':nom', $nom, PDO::PARAM_STR);
$req->bindValue(':mail', $mail, PDO::PARAM_STR);
$req->execute();
}
```

Généralement, on récupèrera des variables de `$_POST` (issues d'un formulaire) pour insérer une entrée dans la base de données, dans notre cas, pour alimenter les variables `$prenom`, `$nom` et `$mail`.

Exercice : Créez une page d'inscription avec un formulaire avec les champs prénom, nom et mail pour alimenter notre nouvelle table users et qui affiche un message de confirmation lorsque l'insertion a été effectuée. Vous ajouterez une dizaine d'utilisateurs.

XXIV.2. Modifier des données

Pour modifier une entrée, on utilise une requête **UPDATE**. Voici un exemple qui modifie les informations d'un utilisateur :

En imaginant qu'on fournit un formulaire d'édition et que l'on autorise les utilisateurs à éditer les champs `prenom`, `nom` et `mail`, voici la requête SQL correspondante :

```
$query = 'UPDATE users SET prenom = :prenom, nom = :nom, mail = :mail WHERE id
= :id';
```

Comment cela fonctionne :

- Tout d'abord, le mot-clé **UPDATE** permet de dire qu'on va modifier une entrée.
- Ensuite, le nom de la table (**users**).
- Le mot-clé **SET** sépare le nom de la table de la liste des champs à modifier.
- Viennent ensuite les champs qu'il faut modifier, séparés par des virgules. Ici, on modifie le champ `prenom`, puis on fait de même pour les champs `nom` et `mail`.
- Enfin, le mot-clé **WHERE** est tout simplement indispensable. Il nous permet de dire à MySQL quelle entrée il doit modifier (sinon, toutes les entrées seraient affectées !). On se base très souvent sur le champ **id** pour indiquer **quelle entrée** doit être modifiée.

Exercice : Créez deux pages, exempleUpdate.php et exempleUpdateChoice.php

Dans **exempleUpdateChoice**, vous mettrez un formulaire en GET qui permet de sélectionner l'utilisateur à modifier, et qui renvoie l'id de l'utilisateur choisi vers **exempleUpdate** :

Choisissez l'utilisateur à modifier

A dropdown menu containing the text "Alex TERRIER". To its right is a blue rectangular button with the word "Valider" in white.

(le select est alimenté par une requête SELECT)

Dans **exempleUpdate** vous remettez le formulaire avec les trois champs mais ils seront préremplis avec les informations de la BDD :

Modification

The form has three text input fields. The first field is labeled "Prénom" and contains "Alex". The second field is labeled "Nom" and contains "TERRIER". The third field is labeled "Email" and contains "alexterrier@yahoo.fr". Below the inputs is a blue rectangular button with the word "Valider" in white.

Vous pouvez accéder à ces informations en vous basant sur l'id transmis dans l'URL, en faisant une requête SELECT, et en mettant les différentes infos dans l'attribut value de chaque champ de formulaire.

Quand on clique sur valider, on exécute une requête UPDATE pour mettre à jour les informations de l'utilisateur.

Si l'URL ne contient pas l'id, on redirige sur **exempleUpdateChoice**.

XXIV.3. Supprimez des données

Pour supprimer une ou plusieurs entrées, on utilise une requête **DELETE**.

ATTENTION, cette requête, bien que simple à utiliser est un peu dangereuse : après suppression, il n'y a aucun moyen de récupérer les données, alors faites bien attention !

Voici un exemple qui supprime un utilisateur à partir de son id :

```
$query = 'DELETE FROM users WHERE id=:id';
```

Voici son fonctionnement :

- **DELETE FROM** : pour dire « supprimer dans » ;
- **users** : le nom de la table ;
- **WHERE** : indispensable pour indiquer quelle(s) entrée(s) doi(ven)t être supprimée(s).

ATTENTION : Si vous oubliez le WHERE , toutes les entrées seront supprimées. Cela équivaut à vider la table.

Exercice :

Créer une page `exempleDelete.php`, contenant un tableau avec l'ensemble des utilisateurs de la table `users`, et une dernière colonne avec un bouton supprimer, permettant la suppression d'un utilisateur :

Choisissez l'utilisateur à supprimer

ID	Prénom	Nom	Email	Supprimer
1	Matthieu	OCTAVE	matthieu.octave@gmail.com	Supprimer
9	Luke	SKYWALKER	laforceoitavectoi@jedi.com	Supprimer
3	Tom	SAWYER	tom.sawyer@gmail.com	Supprimer
8	Gérard	MANFAIM	gerard.mamfaim@gmail.com	Supprimer
5	Alex	TERRIER	alexterrieur@yahoo.fr	Supprimer
6	Alain	TERRIER	alainterrieur@yahoo.fr	Supprimer
10	Leïa	PRINCESSE	princesse.leia@alderaan.com	Supprimer

Ensuite, cherchez un moyen de demander une confirmation au moment du clic sur le bouton, avant que la suppression se fasse :



XXV. Les fonctions SQL

XXV.1. Les fonctions MIN() et MAX()

La fonction SQL **MIN()** va retourner la valeur la plus petite dans une colonne sélectionnée.

La fonction SQL **MAX()** va elle retourner la valeur la plus grande dans une colonne sélectionnée.

Nous pouvons utiliser ces fonctions pour voir par exemple quelle est la plus grosse commande passée sur notre site, quel est l'utilisateur le plus âgé, etc. Notez que ces fonctions vont également fonctionner sur des chaînes de caractères. Dans ce cas, le « a » sera considéré plus petit que « b » et etc.

Par exemple, si l'on reprend notre table `stagiaires`, on peut obtenir le stagiaire le plus jeune avec la requête :

```
SELECT MIN(age) FROM stagiaires
```

Ou le stagiaire les plus âgé avec :

```
SELECT MAX(age) FROM stagiaires
```

XXV.2. La fonction COUNT()

La fonction SQL **COUNT()** va retourner le nombre d'entrées d'une colonne. Nous l'utiliserons généralement avec une clause **WHERE** pour qu'elle retourne le nombre d'entrées qui vont satisfont à une certaine condition.

Cette fonction va être très utile d'un point de vue statistique, pour savoir par exemple combien de vos clients sont des hommes, ou habitent à Paris, etc.

Nous allons par exemple pouvoir savoir le nombre de stagiaires ayant plus de 30 ans :

```
SELECT COUNT(age) FROM stagiaires WHERE age > 30
```

XXV.3. La fonction AVG()

La fonction SQL **AVG()** retourne la valeur moyenne d'une colonne contenant des valeurs numériques.

On va donc pouvoir d'un coup d'œil connaître l'âge moyen des stagiaires :

```
SELECT AVG(age) FROM stagiaires
```

XXV.4. La fonction SUM()

La fonction SQL **SUM()** retourne la somme des valeurs d'une colonne contenant des valeurs numériques.

Cette fonction va être utile pour faire un inventaire des produits vendus par exemple.

Pour la tester, nous pouvons dans notre table additionner les âges même si, ça ne sert pas à grand-chose en pratique :

```
SELECT SUM(age) FROM stagiaires
```

Ces 4 fonctions sont ce que l'on appelle des fonctions d'agrégation car elles ne retournent qu'une seule valeur.

XXV.5. La fonction REPLACE()

La fonction REPLACE dans le langage SQL permet de remplacer des caractères alphanumérique dans une chaîne de caractère. Cela sert particulièrement à mettre à jour des données dans une base de données ou à afficher des résultats personnalisés.

La fonction comporte 3 paramètres :

1. Chaîne d'entrée
2. Texte à remplacer
3. Texte qui sert de remplacement

La requête SQL ci-dessous présente un exemple concret d'utilisation de cette fonction SQL.

```
SELECT REPLACE('Hello tout le monde', 'Hello', 'Bonjour')
```

Résultat :

Bonjour tout le monde

XXV.5.1. Fonction REPLACE dans un UPDATE

Pour modifier certains caractères dans une base de données, il est possible d'utiliser une requête avec la syntaxe suivante:

```
UPDATE stagiaires SET ville = REPLACE(ville, 'BACH', 'CITY')
```

Cette requête permet de remplacer le texte dans la colonne ville. Cela peut se révéler très pratique pour corriger des erreurs d'une base de données :

Résultat :

age	ville
45	HAMBACH
29	FORBACH



age	ville
45	HAMCITY
29	FORCITY

XXV.5.2. Fonction REPLACE dans un SELECT

Pour ne remplacer les caractères que lors de l'affichage et ne pas altérer les données contenu dans la table, il est possible d'utiliser cette fonction dans un SELECT, comme le montre cet exemple:

```
SELECT colonne1, colonne2, REPLACE(colonne3, 'exemple insulte', 'CENSURE')  
FROM table
```

Cela peut se révéler très pratique pour modifier des caractères mais garder une bonne intégrité des données.

Il existe bien d'autres fonction très pratiques que vous pouvez retrouver sur la doc :

<https://sql.sh/fonctions>

Exercice :

1. Importer la table population fournie dans la base formation.
2. Créer une page population.php, contenant un tableau avec l'ensemble des communes de la table population.
3. Mettez en place un système de pagination avant de tenter d'afficher le résultat complet (risque de bug)
4. Ajouter les boutons suivants :

Retour, Min, Max, Population totale, Nombre de communes, Moyenne par commune

5. Associez à chacun de ces boutons un lien engendrant l'exécution de la requête SQL appropriée (en GET), et affichant le résultat :

Population par communes en France

[Retour](#) [Min](#) [Max](#) [Population totale](#) [Nombre de Communes](#) [Moyenne par commune](#)

ID	Code Postal	Commune	Population
1	01001	L' Abergement-Clémenciat	794
2	01002	L' Abergement-de-Varey	249
3	01004	Ambérieu-en-Bugey	14428
4	01005	Ambérieux-en-Dombes	1723
5	01006	Ambléon	117
6	01007	Ambronay	2841
7	01008	Ambutrix	767
8	01009	Andert-et-Condon	339
9	01010	Anglefort	1132
10	01011	Apremont	387
11	01012	Aranc	330
12	01013	Arandas	148
13	01014	Arbent	3459
14	01016	Arbigny	468
15	01015	Arboys en Bugey	661

XXVI. Les fonctions d'agrégation et les critères de sélection

XXVI.1. L'instruction GROUP BY

L'instruction SQL **GROUP BY** va généralement être utilisée avec les fonctions d'agrégation et en combinaison avec **ORDER BY**.

Cette instruction va nous permettre de grouper les résultats renvoyés selon une colonne.

En pratique, cette instruction va être utile pour regrouper des clients selon leur pays par exemple, ou encore selon le montant de leurs commandes, etc.

Dans notre table « etudiants », nous allons ainsi par exemple pouvoir regrouper nos utilisateurs selon leur âge et afficher le nombre de fois où un même âge a été trouvée en utilisant la fonction **COUNT()** :

```
SELECT COUNT(id), age FROM etudiants GROUP BY age ORDER BY COUNT(id) DESC
```

COUNT(id)	age
3	24
2	29
2	25
1	37
1	30
1	36
1	31
1	45

Ici, on commence par compter combien d'id (et donc combien d'entrées) nous avons dans la table avec **COUNT(id)**. On regroupe ensuite les résultats sélectionnés selon la valeur dans la colonne « âge » de notre table. Finalement, on organise ces résultats selon le nombre de fois où un même âge a été trouvé du plus grand nombre de fois au plus petit nombre de fois.

L'ordre des instructions est très important en SQL : si vous indiquez les différentes instructions dans n'importe quel ordre, vous avez de grandes chances pour que votre requête ne fonctionne pas du tout !

XXVI.2. La clause SQL HAVING

La clause SQL **HAVING** remplace la clause **WHERE** dans le cas d'une utilisation avec les fonctions d'agrégation.

En effet, on ne peut tout simplement pas utiliser de clause **WHERE** avec des fonctions d'agrégation car la clause **WHERE** fonctionne avec les données de chaque ligne mais pas avec des données agrégées.

Nous allons par exemple pouvoir sélectionner uniquement les âges qui ne sont présents qu'une seule fois dans notre table et organiser le tout dans l'ordre croissant :

```
SELECT COUNT(id), age FROM etudiants
GROUP BY age
HAVING COUNT(id) < 2
ORDER BY age ASC
```

XXVII. Travailler avec plusieurs tables

Quand on parle d'une « Base de données », cela sous entend la plupart du temps qu'il y'a plusieurs tables dans cette base. Et si cette base est bien construite, on évite la redondance des données, autrement dit une même information ne se répète pas ni au sein d'une table, ni sur différentes tables.

Mais pour éviter cette redondance il faut qu'un lien existe entre les tables qui contiennent des informations qui ont un rapport entre elles.

Ce lien est ce que l'on appelle une **clé étrangère**, et c'est elle qui sert à mettre en relation deux tables dans une BDD relationnelle et d'en exploiter tout la puissance !

XXVII.1. Les clés étrangères

XXVII.1.1. L'intérêt des clés étrangères :

Par exemple, considérons une base de **données** d'une boutique en ligne. Celle-ci peut être composée de deux types de **données** :

- Les données sur les différentes commandes du magasin : Nom du produit et prix
- Les données sur les clients du magasin : Prénom, nom, adresse.

Si l'on stocke toutes ces informations dans une table, on obtiendrait quelque chose comme :

Nom du produit	Prix	Prenom client	Nom client	Adresse client
PC 16'	599	Jean	Bonneau	50 av St Marc
Souris	29	Ambre	Cerna	13 Grande Rue
Lampe de bureau	30	Ambre	Cerna	13 Grande Rue
Clavier	55	Ambre	Cerna	13 Grande Rue

Informations répétées inutilement

Pour éviter cette répétition on va diviser cette base de données en deux **tables** :

- « Commande » : Contient la liste des différentes commandes du magasin.
- « Client » : Contient la liste des différents clients du magasin.

Table “Commande”

Produit	Prix
PC 16'	599
Souris	29
Lampe de bureau	30
Clavier	55

Table “Client”

Prenom	Nom	Adresse
Jean	Bonneau	50 av St Marc
Ambre	Cerna	13 Grande Rue

On allège ainsi la base de **données** en retirant toutes les répétitions.

Cependant, il faut **mettre en relation les deux tables** pour associer les clients Jean Bonneau et Ambre Cerna aux différentes commandes de la boutique.

C'est ici que la **clé étrangère** entre en jeu.

XXVII.1.2. Qu'est-ce qu'une clé étrangère ?

La **clé étrangère** permet :

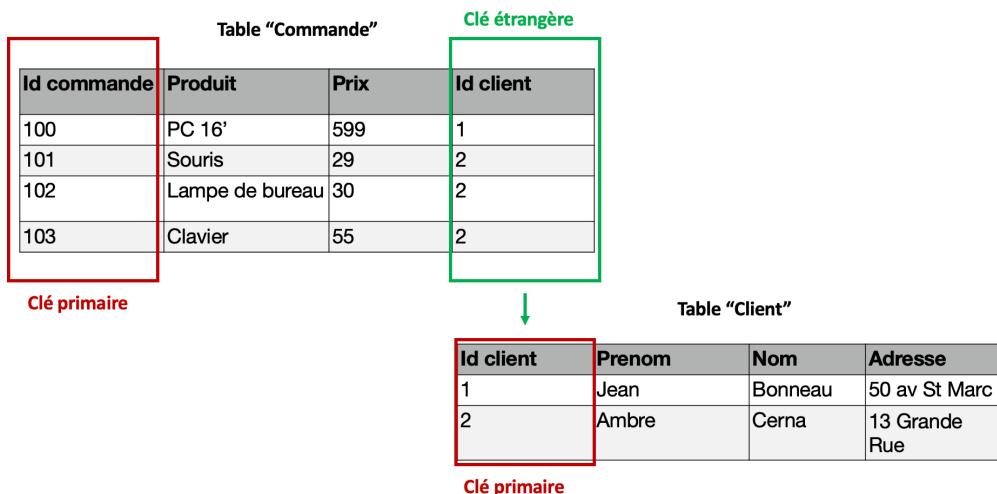
- De mettre en relation deux tables au sein d'une BDD relationnelle.
- D'assurer l'intégrité référentielle des données. Autrement dit, seules les valeurs devant apparaître dans la base de **données** sont permises.

XXVII.1.2.1. La clé étrangère met en relation deux tables :

La **clé étrangère** fait référence à la **clé primaire** d'une autre **table**.

Pour expliquer le rôle de la **clé étrangère** dans la mise en relation de deux **tables**, reprenons l'exemple de la boutique en ligne.

Voici comment mettre en **relation** la **table Commande** avec la **table Client** :



Nous avons ajouté les **clés primaires** de chaque table et une **clé étrangère** dans la table « Commande »

Dans la table « Commande », la **clé étrangère** est l'attribut « Id Client ». En effet, celle-ci référence la **clé primaire** « Id Client » de la table « Client ».

Maintenant que la colonne « Id Client » se trouve dans les deux **tables**, il est facile de retrouver les commandes effectuées par Ambre Cerna. Il nous suffit de ne garder que les commandes dont l'identifiant est 2. Idem pour Jean Bonneau, son identifiant est 1.

La **clé étrangère** « Id client » met donc en **relation** la **table** « Commande » et la **table** « Client ».

XXVII.1.2.2. Clé étrangère et intégrité référentielle de la base de données

La clé étrangère est une contrainte qui s'assure du respect de l'intégrité référentielle de la base de données.

Concrètement, une donnée qui compose la clé étrangère d'une **table A** doit faire référence à une donnée existante dans la clé primaire d'une **table B**.

Dans la boutique en ligne, la contrainte est qu'une commande doit nécessairement être associée à un « Id Client» qui est déjà référencé dans la table « Client ». Sinon, la clé étrangère pointe vers du vide et la mise en relation est impossible !

Pour faire cela, nous avons défini une **clé étrangère** dans la **table « Commande »** : « Id Client». Ainsi, la table « Commande » n'accepte que des « Id Client» qui existe dans la **table « Client »**.

De cette manière, on s'assure que la **table « Commande »** contient uniquement des informations sur des clients existants dans la **table « Client »**.

En définissant une **clé étrangère**, nous avons donc respecté l'intégrité référentielle dans la base de données de la boutique en ligne !

XXVII.1.3. Définir une clé étrangère avec phpMyAdmin

Pour créer une clé étrangère dans phpMyAdmin, il faut tout d'abord vous assurer que les tables utilisent le moteur InnoDB :

Dans le menu structure d'une base :

Table	Action	Lignes	Type	Interclassement	Tai
lexique	★ Parcourir Structure Rechercher Insérer Vider Supprimer	4	InnoDB	utf8_general_ci	16,
photos	★ Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8_general_ci	32,
2 tables	Somme	4	InnoDB	utf8_general_ci	48,

Pour changer le moteur :

Aller sur une table, dans le menu Opérations :

The screenshot shows the 'Options pour cette table' (Options for this table) dialog box. It includes fields for renaming the table ('Renommer la table en' with 'photos') and adjusting privileges ('Ajuster les priviléges'). At the bottom, there is a 'Moteur de stockage' (Storage engine) dropdown set to 'InnoDB'. This dropdown is circled with a red oval.

Pour créer la clé étrangère, il faut d'abord créer une colonne dans la table (id_autretable),

Puis dans structure aller sur « Vue Relationnelle » :



Vous arrivez sur l'écran suivant :

A screenshot of the MySQL Workbench 'Contraintes de clé étrangère' (Foreign Key Constraints) configuration screen. The screen has a header with tabs: 'Structure de table' and 'Vue relationnelle'. Below the header is a table with columns: 'Actions', 'Propriétés de la contrainte', 'Colonne', and 'Contrainte de clé étrangère (INNODB)'. Under 'Actions', there are dropdown menus for 'ON DELETE' (set to 'RESTRICT') and 'ON UPDATE' (set to 'RESTRICT'). Under 'Propriétés de la contrainte', there are dropdown menus for 'Base de données' (set to 'lexique'), 'Table' (set to 'commande'), and 'Colonne' (set to 'id_commande'). At the bottom left are buttons for 'Aperçu SQL' and 'Enregistrer'.

Il ne reste plus qu'à compléter les informations demandées :

- Nom de la contrainte : donner un nom ou laisser vide il sera généré automatiquement,
- ON DELETE : définir le comportement souhaité en cas de suppression de la cible de la relation, dans notre exemple, que faire avec la commande si l'on supprime l'utilisateur à l'initiative de cette commande :
Cette dernière peut prendre une option parmi celles-ci :
 - RESTRICT ou NO ACTION : MySQL va empêcher la suppression tant qu'un utilisateur est référencé sur au moins une commande.
 - SET NULL: MySQL va autoriser la suppression de l'utilisateur, et remplacer "id_client" sur les commandes concernées par la valeur NULL.
 - CASCADE: l'option la plus courante, mais la plus dangereuse. Ici, MySQL va supprimer les commandes liées en même temps que l'utilisateur (il va donc supprimer tous les objets reliés).
- ON UPDATE : Même options mais en cas de mise à jour,
- Colonne : Choisir la colonne qui contiendra la clé étrangère, ici id_client,
- Base de donnée, table et colonne : Choisir à quoi fait référence notre clé étrangère, ici ce sera la colonne id_client de la table commande.

XXVII.1.4. Définir une clé étrangère en SQL

On peut aussi définir une clé étrangère à l'aide d'une commande SQL :

```
ALTER TABLE commande
ADD FOREIGN KEY (id_client) REFERENCES commande (id_commande)
ON DELETE SET NULL;
```

XXVII.1.5. Limitation et contraintes de la clé étrangère

La clé étrangère doit suivre une série de contraintes :

- La clé étrangère ne peut faire référence qu'à une colonne (ou des colonnes) au sein de la même base de données, sur le même serveur.
- La colonne de la clé étrangère et celle qu'elle référence doivent être de même type (INT, VARCHAR etc...).
- Une clé étrangère ne peut pas être appliquée dans des tables temporaires.

Faites donc bien attention à respecter ces contraintes en manipulant une clé étrangère !

XXVIII. Les jointures

Maintenant l'on a vu l'intérêt d'une clé étrangère dans une base de données relationnelle, voyons comment exploiter cet outil grâce à une technique très connue des Data analystes : les jointures.

Concrètement, une jointure SQL permet de fusionner tout ou partie de plusieurs tables afin d'extraire les informations à analyser. Pour y arriver, il faut donc utiliser la ou les clés étrangères à disposition.

XXVIII.1. Les différents types de jointures

Il existe différents types de jointures en SQL mais tous ne sont pas supportés par le MySQL. Nous allons donc simplement nous concentrer sur les types de jointures suivants:

- L'**INNER JOIN** (jointure interne);
- Le **LEFT JOIN** (jointure externe);
- Le **RIGHT JOIN** (jointure externe);

Les jointures de type interne, par exemple, ne vont nous retourner des résultats que lorsqu'il y aura une correspondance entre les deux tables, tandis que les jointures de type externe vont nous retourner des données même lorsqu'il n'y aura pas de correspondance dans la seconde table.

XXVIII.2. L'INNER JOIN

Nous allons pouvoir créer une jointure interne à l'aide du mot clef **INNER JOIN** en SQL.

Une jointure interne est un type de jointures qui va nous permettre de ne sélectionner que les données relatives aux entrées qui ont des valeurs identiques dans les deux tables.

Pour appliquer un **INNER JOIN**, nous allons avoir besoin d'une colonne de référence dans chacune des deux tables, c'est-à-dire de deux colonnes qui ont des valeurs qui représentent les mêmes choses.

Si l'on continue avec notre exemple, si l'on souhaite lister toutes les commandes passées par Ambre CERNA, on utilisera la requête suivante :

```
SELECT c.prenom, c.nom, cmd.produit, cmd.prix
FROM `commandes` as cmd
INNER JOIN client as c
ON cmd.id_client = c.id
WHERE cmd.id_client=2
```

Et l'on obtient le jeu de résultats suivant :

prenom	nom	produit	prix
Ambre	CERNA	Souris	29
Ambre	CERNA	Lampe de Bureau	30
Ambre	CERNA	Clavier	55

XXVIII.3. Les Alias

Vous noterez que dans cette requête, on utilise des alias, spécifiés avec le mot clé **AS** pour simplifier la requête.

Cependant, l'utilisation d'alias n'est pas du tout obligatoire, et quand on utilise des alias on n'es pas obligé d'utilisé AS.

Voici la requête sans les AS :

```
SELECT c.prenom, c.nom, cmd.produit, cmd.prix
FROM `commandes` cmd
INNER JOIN client c
ON cmd.id_client = c.id
WHERE cmd.id_client=2
```

Et la même requête sans alias :

```
SELECT client.prenom, client.nom, commandes.produit, commandes.prix
FROM `commandes`
INNER JOIN client
ON commandes.id_client = client.id
WHERE commandes.id_client=2
```

Attention, nous n'allons toutefois pas pouvoir utiliser les alias de colonnes avec n'importe quelle clause en MySQL. En particulier, il est interdit d'utiliser les alias de colonnes au sein d'une clause **WHERE** car en SQL standard, lorsqu'une clause **WHERE** est évalué la valeur de la colonne en soi peut ne pas encore avoir été déterminée.

Nous allons pouvoir utiliser les alias sans soucis avec les clauses **GROUP BY**, **ORDER BY** et **HAVING**.

XXVIII.4. Le LEFT JOIN

Le **LEFT JOIN** (également appelé **LEFT OUTER JOIN**) est un type de jointures externes.

Ce type de requête va nous permettre de récupérer toutes les données (relatives aux colonnes spécifiées) de la table de gauche c'est-à-dire de notre table de départ ainsi que les données de la table de droite (table sur laquelle on fait la jointure) qui ont une correspondance dans la table de gauche.

Avec ce type de requête, nous allons par exemple pouvoir récupérer tous les noms et prénoms de nos clients dans notre table « client » et SEULEMENT les commandes liés à un utilisateur de notre table « commandes » :

```
SELECT c.nom, c.prenom, cmd.produit, cmd.prix  
FROM `client` c  
LEFT JOIN commandes cmd  
ON c.id = cmd.id_client
```

Résultats :

nom	prenom	produit	prix
BONNEAU	Jean	PC 16"	599
CERNA	Ambre	Souris	29
CERNA	Ambre	Lampe de Bureau	30
CERNA	Ambre	Clavier	55
MAMFAIM	Gérard	NULL	NULL
TERRIEUR	Alex	NULL	NULL

XXVIII.5. Le RIGHT JOIN

Le **RIGHT JOIN**, encore appelé **RIGHT OUTER JOIN** est un autre type de jointures externes qui va fonctionner de la même façon qu'un **LEFT JOIN** à la différence que cette fois-cice sont toutes les données de la table de droite (table sur laquelle on effectue la jointure qui vont être récupérées tandis que seules les entrées de la table de gauche ou table de départ qui vont satisfaire à la condition de jointure seront sélectionnées).

La requête ci-dessous par exemple va bien récupérer le contenu de toutes les commandes dans la table « commandes » sans exception et SEULEMENT les noms et prénoms des personnes qui ont passé une commande :

```
SELECT c.nom, c.prenom, cmd.produit, cmd.prix  
FROM `client` c  
RIGHT JOIN commandes cmd  
ON c.id = cmd.id_client
```

Résultats

nom	prenom	produit	prix
BONNEAU	Jean	PC 16"	599
CERNA	Ambre	Souris	29
CERNA	Ambre	Lampe de Bureau	30
CERNA	Ambre	Clavier	55
NULL	NULL	Ecran 22"	149

XXIX. L'opérateur SQL UNION

Voici un nouvel opérateur SQL : l'opérateur **UNION** et nous allons apprendre à l'utiliser pour combiner les résultats de deux requêtes **SELECT**.

Afin que cet opérateur fonctionne correctement et que les résultats puissent bien être combinés, il va falloir respecter les règles suivantes :

- Chacun des **SELECT** dans un **UNION** doit posséder le même nombre de colonnes ;
- Les colonnes doivent posséder le même type de données une à une dans chaque **SELECT** (la première colonne du premier **SELECT** doit posséder des données de même type que la première colonne du deuxième **SELECT** et etc.).

Notez bien que les colonnes sélectionnées peuvent avoir un nom différent entre les différents **SELECT** du moment que le type de données est le même. En cas de nom différent, et si nous voulons harmoniser les noms, nous pouvons tout à fait utiliser un alias.

Par défaut, l'opérateur SQL **UNION** ne va pas sélectionner les doublons dans les valeurs (les valeurs qui sont trouvées plusieurs fois n'apparaîtront qu'une seule fois).

Par exemple, dans la base d'une application d'entreprise, imaginons deux tables :

- **users**, contenant les utilisateurs de l'application (ne travaillant pas forcément tous au sein de l'entreprise)
- **employes**, contenant les employés de l'entreprise (n'étant pas forcément tous utilisateurs de l'application)

Si l'on souhaite récupérer tous les noms et prénoms des personnes présentes dans ces deux tables, on aurait la requête suivante :

```
SELECT nom, prenom FROM users
UNION
SELECT nom, prenom FROM employes
```