

# Module PHP

## Plan de cours

1. Intro : découverte et explications autour de PHP
2. Instructions d'affichage
3. Variable (et Constante) : Type / Déclaration / Affectation
4. Syntaxe et concaténation
5. Opérateurs arithmétiques
6. Conditions
7. Fonction prédéfinie
8. Fonction utilisateur
9. Portée des variables
10. Boucles
11. Inclusions de fichiers
12. Arrays
13. Classes et objets
14. Les Superglobales
15. Lien GET et Formulaire POST
16. Cookie
17. Session
18. PDO et requêtes SQL
19. Réalisation d'un espace de dialogue (Cas Concret)
20. Approche Sécurité
21. Réalisation d'un site web (Cas concret : CRUD - tendance e-commerce)

### 1. Intro : découverte et explications autour de PHP

Pour pouvoir afficher une page PHP en local, il est indispensable de travailler avec Mamp ou Xampp.

Sous Wamp, tous mes fichiers devront être mis dans le dossier /www/

Sous Xampp, ça sera dans /htdocs/

Dans les deux cas, par souci de méthodologie et d'organisation, je crée un sous-dossier nommé /php/, et c'est dans ce dernier que seront logés tous mes fichiers .php

## 2. Instructions d'affichage

Pour pouvoir travailler en **PHP**, désormais mon fichier devra posséder l'extension *.php* (et non plus *.html*)

Cette extension me permettra néanmoins de le combiner au **HTML** (et son **CSS** relié)

Dans ce fichier en *.php*, je devrai ouvrir des passages **PHP** (un ou plusieurs, selon mes besoins)

Tout script **PHP** devra être inclus dans les balises

```
<?php
// code
?>
```

Si le passage **PHP** clôt mon fichier, je pourrai alors ne pas le fermer, et cela est même conseillé

Cela sera le cas pour celui qui appelle mon footer en bas de page (nous verrons cela lors des inclusions de fichiers)

La première instruction que nous allons voir ensemble permet d'afficher du texte

J'en ai deux à ma disposition ( `echo` et `print` )

```
<?php
echo "Je m'appelle Aziz ";
print 'et je réside à Créteil';
```

Pour l'instant, en tant que débutants, on va considérer qu'elles servent strictement à la même chose. On va privilégier l'instruction `echo` durant notre apprentissage (d'autant plus qu'elle est légèrement plus rapide à l'exécution (car elle ne fait pas de return)

`print` pourrait nous servir plus tard à par exemple évaluer une expression (mais je clos rapidement cet aspect et concentrons nous sur `echo`)

Noter que pour afficher mon texte, j'ai dû l'écrire en apostrophes (`quotes`) ou guillemets (`double quotes`). Là aussi, c'est indifférent, l'un où l'autre seront plus avantageux selon les cas de figure que nous allons rencontrer progressivement

Je peux aussi, avec un passage php contracté (entre du code html), afficher sans l'instruction `echo`.

```
<?= "Je m'appelle Aziz et je réside à Créteil" ?>
```

Mais il ne sera pas pertinent dans ce cas de figure (autant continuer à coder en html). Il sera cependant utile lorsque je voudrais afficher la valeur d'une variable (ou constante) au sein du code html

Pour me dé buguer, j'ai à ma disposition deux fonctions prédéfinies. Il s'agit de `print_r` et `var_dump`.

Elles nous seront très utiles tout le long de la construction de notre site web.

Je reviendrai plus tard, de manière plus détaillée sur leur utilisation

## Les commentaires en php

Selon que mon commentaire se limite à une ligne, je le débute par //

Si en revanche il se développe sur plusieurs, alors je le déclarerai entre `/*` et `*/`

```
// commentaire sur une seule ligne  
  
/* commentaire sur  
plusieurs lignes */
```

Noter qu'avec Visual Studio, je peux générer un commentaire avec le raccourci clavier `ctrl /`

Et ce, pour tous les langages.

## Afficher en mélangeant php et html

Je peux donc écrire du html dans un fichier en .php ( `mais pas le contraire` )

```
echo "<h1>Bonjour</h1>";  
echo "Nous sommes <strong>lundi</strong> matin <br>";  
echo "<div class='horaire'>Le cours commence à 9h</div>";
```

Ci dessus, dans l'ordre, j'ai une balise titre `h1`, une autre pour mettre en avant du texte (`strong`) puis une div avec une `classe`

**Attention:** si la syntaxe ci-dessous est valide, elle est néanmoins considérée comme «sale» car elle va générer trop d'entrées et sorties d'un langage vers l'autre

```
<h1><?php echo 'Bonjour'; ?></h1>  
<?php echo 'Nous sommes'; ?> <strong>lundi</strong> <?php echo ' et il est 9h !'; ?  
> <br>  
<div class="mazonne"><?php echo 'Je crée une zone html dans du php'; ?> </div>
```

En effet, ligne 1, je passe du html vers du php pour ré entrer dans du html

Ligne deux, c'est php vers html pour sortir de php pour entrer dans du html puis entrer dans du php puis dans du html

Ligne 3 étant similaire à la ligne 1

Sans parler du fait qu'il est plutôt long à développer. Néanmoins, vous pourrez le rencontrer, si c'est un débutant qui code, ou alors dans le cas des CMS, car les développeurs vont penser aux intégrateurs (qui ne maîtrisent pas php), et multipliant les niches de php dans du html.

Ceci dit, le cas de figure ci-dessus est à différencier des passages php contractés, fréquents et utiles pour afficher des valeurs au sein du code html.

**Noter** aussi que si je peux mêler du php et du html dans un même fichier, si je tente afficher le code source de ma page sur le navigateur, seul le html y figurera. Le navigateur ne gère pas php, cqfd !

### 3. Variable (et Constante) : Type / Déclaration / Affectation

[Documentation officielle](#)

Une **variable** peut être décrite comme un contenant (en fait un espace mémoire), dans lequel je range/stocke diverses valeurs/données. Cela pourra être un mot, une phrase, un chiffre, le résultat d'une requête (SQL) ou d'une opération arithmétique etc.

J'**affecte** cette valeur dans ma **variable** avec l'**opérateur d'affectation** = . Et, une fois "rangée" dans ma variable, cette donnée sera récupérable, réutilisable lorsque j'en aurai besoin

Cela sera notamment utile dans le cas ci-dessous, pour éviter des répétitions

```
echo "<p>Bonjour Fred</p>";  
echo "<p>Comment vas-tu Fred ?</p>";  
echo "<p>Voici tes informations de profil Fred ...</p>";
```

Il sera plus pratique de déclarer une variable prénom, y affecter la valeur Fred pour ensuite l'utiliser autant de fois que j'en aurai besoin

```
$prenom = "Fred";  
echo "<p>Bonjour $prenom</p>";  
echo "<p>Comment vas-tu $prenom ?</p>";  
echo "<p>Voici tes informations de profil $prenom ...</p>";
```

Elle doit obligatoirement débuter par le signe \$, qui ne pourra être suivi d'un chiffre (elle ne sera pas validée ... **erreur PHP**). Par convention, elle ne peut contenir d'accent ou débuter par un underscore (tiret du 8) et de manière générale les caractères spéciaux.

Si le nom de la variable est un mot composé, j'utiliserais le **camelCase** ou le **snake\_case**

Par convention, on utilise le camelCase pour les Variables et le snake\_case pour les fonctions (et constantes)

Penser aussi à nommer la variable de manière pertinente. Ci-dessus, la variable qui contenant la valeur Fred, a été nommée \$prenom. Si elle a vocation a contenir la valeur rouge, il faudra la nommer \$couleur etc.

**Attention:** le nom de la variable est sensible à la casse. Cela signifie que si je n'orthographe pas bien ma variable au moment où je l'appelle (\$prenoms au lieu de \$prenom) dans mon script, je ne pourrais afficher sa valeur (j'aurai même droit à un warning php Undefined variable \$prenoms )

Enfin, comme le suggère son nom, une variable peut...varier

Cela veut dire qu'à tout moment de mon script, je peux lui affecter une nouvelle valeur et c'est cette dernière qui s'affichera désormais suite à un echo

```
$prenom = "Aziz";  
echo "<p>Bonjour $prenom</p>";  
echo "<p>Comment vas-tu $prenom ?</p>";  
echo "<p>Voici tes informations de profil $prenom ...</p>";
```

Vérifier sur le navigateur. Le changement a du s'effectuer.

### 3-1 Les types de variables

PHP n'est pas un langage typé (contrairement à Java, par exemple). En ce sens, au moment de la déclaration de la variable, ma seule obligation est son nommage. Je ne suis pas obligé de lui donner une valeur immédiatement, ou de préciser son type.

Pour connaître le type d'une variable, j'ai à ma disposition une fonction prédéfinie `gettype()` ([documentation officielle](#)). Elle fait partie des [fonctions de gestion des variables](#).

Je peux ainsi passer au filtre n'importe quelle variable pour connaître son type

```
$identite = "Aziz";  
echo gettype($identite) . "<br>";  
  
$nombre_entier = 32;  
echo gettype($nombre_entier) . "<br>";  
  
$nombreDecimal = 5.2;  
echo gettype($nombreDecimal) . "<br>";  
  
$booleen = TRUE;  
echo gettype($booleen) . "<br>";
```

Le résultat de ces echo sera respectivement, `string` (chaîne de caractères), `integer` (nombre entier), `double` (nombre décimal) et `boolean` (un booléen)

Dans le cas du booléen, si je veux afficher sa valeur

```
echo $booleen;
```

Sa valeur retournée sera 1 (pour True) et 0 pour False.

Ce sont les 4 types avec lesquels nous travaillerons pour l'instant, même si il en existe [d'autres](#).

## **3-2 Les constantes**

Une constante a le même rôle qu'une variable, sauf que, comme l'indique son nom, cette valeur ne pourra être modifiée.

Par convention, son nom devra être écrit en majuscules, et s'il est composé, je devrais utiliser le `snake_case`

Pour la déclarer je devrai aussi utiliser une syntaxe particulière.

Elle se fait avec `define()`, fonction prédéfinie ([documentation officielle](#)), qui prend entre parenthèses deux arguments. Le nom de la constante (en majuscules donc), puis sa valeur

```
define('VILLE', 'Créteil');  
echo VILLE;
```

Logiquement, contrairement à une *variable*, je ne peux réaffecter lui une nouvelle valeur. Cela entraînera un warning PHP

**Warning:** Constant VILLE already defined

La question que je peux me poser, c'est quand utiliser une variable, et quand plutôt une constante ?

De manière générale, nous utiliserons massivement des variables, et plutôt une constante lorsque je serais sûr à 100% que la valeur que je veux transporter est unique, et doit être protégée d'une malencontreuse modification. Cela pourrait être le chemin vers un dossier fixe (pour uploader une image, par exemple) ou alors l'URL du site

Nous verrons cela lorsque nous créerons la boutique.

## **3-3 Les constantes magiques**

Il en existe de nombreuses ([documentation officielle](#))

Nous allons juste en aborder deux, `__FILE__` et `__LINE__`

La première donnera le chemin physique du fichier dans lequel nous codons. La seconde, la ligne où nous codons

Noter leur syntaxe particulière avec les deux underscores au début et à la fin (4 underscores au total donc)

```
echo __FILE__;
```

```
echo __LINE__;
```

## 4. Syntaxe et concaténation

En PHP je peux concaténer de deux manières, avec une virgule ( , ) (mais pas avec print) ou un point ( . ) (fonctionne avec print et echo)

Première concaténation avec des virgules

```
$prenom = "Aziz";  
$nom = "Tobbal";  
  
echo "Je suis ", $prenom, " ", $nom , "<br>";
```

(quasi) Strictement la même, avec des points

```
echo "Je suis ". $nom. " ". $prenom . "<br>";
```

Dans ce cours, nous utiliserons le point pour toutes nos concaténations.

### 4-1 La concaténation par affectation

Je peux concaténer deux valeurs à une même variable grâce à l'opérateur combiné `.=`

```
$nombre = 36;  
echo $nombre .= 15;
```

Le résultat sera `3615`

Je constate deux choses: `.=` sert à concaténer et non pas à additionner (nous verrons comment plus tard)

Le second constat est que cet opérateur combiné ne me sera pas utile si je veux n'afficher que 36 ou 15. Désormais c'est une nouvelle valeur unique.

Par moments, dans mon script, j'aurai besoin de `.=`, mais je m'aperçois aussi que je dois garder à l'esprit que pour chaque valeur j'ai besoin d'une variable

### 4-2 La syntaxe (différence entre [quotes](#) et [doubles quotes](#))

Si je code ceci

```
$exemple = 'test';  
  
echo "Ma variable $exemple est de type string";
```

```
echo 'Ma variable $exemple n\'est pas interprétée';
```

Je peux voir que ma variable, lorsqu'elle est déclarée en double quotes est interprétée. Par contre, entre simples quotes, elle n'est pas évaluée/interprétée.

Je dois donc faire attention à cette différence. Dans le second cas il aurait fallut faire de la concaténation

```
echo 'Ma variable ' . $exemple . ' n\'est pas interprétée';
```

Remarquer d'ailleurs les codes couleurs différents.

Autre différence. L'anti-slash que je dois utiliser

```
n\'est
```

Sans, l'apostrophe qui suit le n serait interprétée comme une fermeture de quote , entraînant une erreur PHP pour tout ce qui suit

De manière globale, nous utiliserons les simples quotes au moment d'affecter une valeur à une variable, et plutôt les doubles pour afficher les chaînes de caractères (mais il faudra s'adapter selon les cas de figure)

## 5. Les opérateurs arithmétiques

[Documentation officielle](#)

+, -, \* et / sont les **opérateurs arithmétiques** de base avec lesquels je vais pouvoir additionner, soustraire, multiplier ou diviser

Exemple avec les deux variables suivantes qui ont respectivement pour valeurs 4 et 2

```
$premierNombre = 4;
$secondNombre = 2;

echo $premierNombre + $secondNombre;
echo $premierNombre - $secondNombre;
echo $premierNombre * $secondNombre;
echo $premierNombre / $secondNombre;
```

Il existe deux autres opérateurs arithmétiques; le **modulo %** et l'**exponentiation \*\*** (ce dernier depuis PHP 5.6)

Le modulo permet de connaître le reste d'une division. L'exponentiation permet d'affecter la puissance du second nombre au premier

```
echo $premierNombre % $secondNombre;
echo $premierNombre ** $secondNombre;
```



En dernier, nous allons voir les opérateurs d'affectation ( [documentation officielle](#) ). Il s'agit de += , -= , \*= et /=

\$a += \$b est équivalent à \$a = \$a + \$b

Concrètement, si je reprends mes deux variables du dessus, cela donnera

```
echo $premierNombre += $secondNombre;  
echo "<br>";
```

Résultat: 6 (ne pas concaténer le <br> en fin du premier echo, warning php **A non-numeric value encountered** les <br> concaténés en fin des autres opérations fonctionnent, mais par avec les opérateurs d'affectation)

**Attention:** Jusqu'à présent je pouvais réutiliser mes deux variables pour toutes les opérations en conservant leur valeur initiale.

Désormais, la valeur de \$premier nombre n'est plus 2, mais 6. Cela rejoint le cas vu précédemment avec .=

C'est une nouvelle valeur qui a été affectée. Bien garder cela en mémoire en vue d'autres opérations.

Il serait peut-être (selon les cas) plus pertinent de créer une nouvelle variable pour garder le résultat de l'addition de deux autres variables

## 6. Conditions

Les **conditions** sont incontournables dans le fonctionnement d'un site dynamique. On les retrouve d'ailleurs dans tous les langages de programmation

J'en aurai par exemple besoin pour vérifier que le mot de passe inséré dans le formulaire de connexion est bien le même que celui enregistré en BDD (base de données)

Si ( **if** ) le mot de passe est similaire, alors l'utilisateur sera connecté. Sinon ( **else** ), je l'en avertis, et ce processus de connexion ne sera pas validé

Pour un site qui fait de la vente, je pourrai aussi vérifier la quantité commandée par rapport au stock disponible en BDD

Si ( **if** ) mon stock est inférieur à la quantité commandée, la vente ne pourra être validée. Je pourrai alors proposer à l'utilisateur une nouvelle quantité (égale ou inférieure au stock) qui permettra de continuer le processus de vente

Il ne faut pas oublier qu'entre le moment où le produit aura été mis dans le panier et sa validation pour le paiement (plusieurs minutes, heures voire jours auront pu s'écouler), le stock aura pu décroître

Dans la syntaxe que nous allons voir, **if** signifie si, **else** signifie sinon et **elseif** signifie sinon si

## 6-1 La condition if/else

Je déclare 5 variables dont je vais avoir besoin au fur et a mesure

```
$a = 22;  
$b = 27;  
$c = 35;  
$d = 48;  
$e = 55;  
  
if($a < $b){  
    echo "Vrai, a est bien inférieur à b";  
}else{  
    echo "Faux, a n'est pas supérieur à b";  
}
```

Pour ce premier test, je vérifie dans les parenthèses du if si la valeur de \$a est inférieure à celle de \$b

Si c'est TRUE, alors c'est le premier message qui s'affichera. Dans le cas contraire, c'est le second

Noter que pour le else, je n'ai jamais de parenthèse. Je n'ai aucun cas à vérifier car il s'agit de TOUS les autres cas

Autre remarque, j'aurai pu aussi coder (avec le même résultat)

```
if($a < $b){  
    echo "Vrai, a est bien inférieur à b <br>";  
}
```

Je ne suis pas obligé de coder le **else** . Dit autrement, si la condition est vérifiée, affiche le message. Si elle n'est pas vérifiée, tu ne fais rien (tu n'affiche rien).

## 6-2 If/else avec AND (&&)

Je vais faire une double vérification, et pour que ma condition soit TRUE, il faudra que les deux conditions soient vérifiées

```
if($a < $b && $b > $c){  
    echo "Vrai, les deux conditions sont respectées";  
}else{  
    echo "Faux, une des deux conditions n'est pas respectée";  
}
```

Comme la seconde condition n'est pas vérifiée et que **&&** m'oblige à ce qu'elles le soient toutes, je tombe dans le **else**

## 6-4 If/else avec OR ( || )

Toujours une double vérification, et pour qu'elle soit TRUE, il faut qu'une des deux soit vérifiée (ou les deux)

```
if($a == 22 || $b > if($a < $b || $b > $c){  
    echo "Vrai, une des deux conditions est bien respectée";  
}else{  
    echo "Faux, aucune condition n'est respectée";  
}
```

## 6-5 If/else avec XOR (ou exclusif)

Avec XOR, pour que la vérification soit TRUE, il faut que seule une des conditions soit respectées.

Sinon, c'est le else

```
if($a == 22 XOR $b == 27){  
    echo "Vrai, une des deux conditions est bien respectée";  
}else{  
    echo "Faux, les deux conditions sont respectées";  
}
```

## 6-6 If/elseif/else

[Documentation officielle](#)

**elseif** permet de vérifier plusieurs cas de figure

Ici, deux (puis le **else**), avec introduction de l'élément de comparaison **!=** (différent de)

```
if($a > $b){  
    echo "Vrai, a est bien supérieur à b";  
}elseif($a != 22){  
    echo "Vrai, a n'est pas égal à 22";  
}else{  
    echo "Faux, aucune condition n'est vraie <br>";  
}
```

Pour résumer, si je ne rentre pas dans le if, alors je passe au elseif, et si sa condition n'est toujours pas vérifiée, j'entre dans le else.

A noter que je peux avoir plusieurs elseif.

## 6-7 If/else contracté

Cette syntaxe est aussi appelée **ternaire**

```
echo ($d < $e)? "Vrai, d est bien inférieur à e" : "Faux";
```

Avec cette syntaxe, la condition est toujours placée dans la parenthèse ( $d < e$ ), le **?** remplace le if et les **:** remplacent le else

Durant les révision nous ne l'utiliseront pas beaucoup, mais lorsque arrivera le moment de construire la boutique, nous ferons appel à cette syntaxe de temps en temps, car utile et lisible

## **6-8 if/else avec == et ===**

### [Documentation officielle](#)

En plus de **!=** (différent de), **<** (ou **<=**), **>** (ou **>=**), deux autres opérateurs de comparaison seront très utiles

Le **==** (double égalité) et le **===** (triple égalité)

**Attention** le **=** n'est pas un opérateur de comparaison. Il sert à affecter une valeur, pas à comparer

Pour reprendre l'exemple de la connexion, c'est le **==** qui va me permettre de comparer le mot de passe inséré dans le formulaire avec le mdp stocké en BDD

Dans l'exemple qui suit, je déclare deux variables que je vais comparer de deux manières. D'abord avec le **==**

```
$var = 100;
$var2 = "100";

if($var == $var2){
    echo "Les deux variables ont la même valeur";
}else{
    echo "Les deux variables ont une valeur différente";
}
```

Puis avec **===**

```
if($var === $var2){
    echo "Les deux variables ont la même valeur et le même type";
}else{
    echo "Les deux variables ont un type différent";
}
```

Dans le premier cas, c'est **TRUE** car **==** ne va comparer que la valeur, et malgré la différence de type.

Par contre, le **===** va aussi comparer le type. C'est pour cela que c'est **FALSE** et que j'entre dans le else

## 6-9 if(isset())/else

[Documentation officielle](#)

Pour vérifier si une *variable* existe, a déjà été déclarée, définie etc...

Très utile pour vérifier si une valeur a bien été fournie dans un formulaire (d'inscription, par exemple), avant de le valider (couplée à `isset($_POST['indice_a_verifier'])`, dans le cas présent)

Même chose avec `isset($_GET['indice_a_verifier'])`, pour s'assurer que des informations ont bien transité par l'URL (ou `isset($_SESSION['indice_a_verifier'])` pour vérifier si la session existe)

Dans ces trois cas de figures, `isset()` pourra être remplacé par `array_key_exists` (syntaxe légèrement différente -> `if(array_key_exists('indice_a_verifier', $_POST))`)

Dans l'exemple ci-dessous je vais vérifier si la variable `$test` a bien été initialisée, déclarée au préalable

```
if(isset($test)){
    echo "La variable test existe";
}else{
    echo "La variable test n'a pas été déclarée";
}
```

Bien entendu, comme elle n'a jamais été déclarée, je vais tomber dans le else

Pour progressivement se familiariser avec, voici cette même condition sous sa forme contractée / ternaire

```
echo (isset($test)) ? "La variable test existe" : "La variable test n'a pas été déclarée" ;
```

**Remarque:** `isset()` ne pourra être utilisé pour tester une constante. Il faudra utiliser `defined()` .

## 6-10 La condition switch

[Documentation officielle](#)

Elle est équivalente à un if/elseif/else, et plutôt plus performante (et donc conseillée) que cette dernière s'il y a plusieurs elseif. Mais il est vrai qu'elle est moins fréquemment utilisée

Peut-être à cause de sa syntaxe un peu plus contraignante (alternance entre les `;` et les `:` les `break` etc...)

```
$couleur = "bleu";

switch($couleur){
    case "vert":
        echo "La couleur est bien vert";
        break;
    case "bleu":
```

```

    echo "La couleur est bien bleu";
    break;
default:
    echo "Pas la bonne couleur";
    break;
}

```

La même en `if/elseif/else`

```

$couleur = "bleu";

if($couleur == "vert"){
    echo "La couleur est bien vert";
}elseif($couleur == "rouge"){
    echo "La couleur est bien rouge";
}else{
    echo "Pas la bonne couleur";
}

```

Retenir essentiellement qu'elles sont équivalentes pour vérifier des conditions, avec de meilleures performances en terme de calcul et donc d'affichage s'il y a plusieurs `elseif`

## 7 Fonctions prédéfinies

Il en existe une multitude, mises à notre disposition ( [documentation officielle](#) )

Nous n'allons pas toutes les détailler, tant il y'en a, juste quelques unes

### 7-1 iconv\_strlen() et strlen()

[iconv\\_strlen\(\)](#) permet de compter le nombre de caractères d'un string/chaîne de caractères donné en argument dans la parenthèse de la fonction

```

$phrase = "Je suis un cristolien d'adoption";
echo iconv_strlen($phrase);

```

[strlen\(\)](#) permet de faire la même chose, à cette nuance près qu'elle va aussi compter les accents ( et autres caractères spéciaux ) en plus.

Exemple ci-dessous avec le mot `étés`

```

$chaine = "étés";
echo strlen($chaine);

```

Le résultat sera 6. Un pour chaque caractère + 2 autres pour les accents

Ainsi, selon que je veuille ou non prendre en compte l'espace mémoire mobilisé par la chaîne de caractères (c'est le cas de `strlen()`), j'utiliserai l'une ou l'autre.

## **7-2 substr()**

`substr()` permet de sélectionner un "morceau" d'une chaîne de caractère et de supprimer le reste

Elle exige trois arguments: la chaîne à découper, le point de départ et le point d'arrivée

Exemple ci-dessous en découplant `$phrase` en son milieu, sachant que grâce à `iconv_strlen` je connais déjà sa longueur (32)

```
echo substr($phrase,0,16);
```

Elle sera pratique pour supprimer la dernière lettre d'une chaîne de caractères (en donnant une valeur négative au point d'arrivée)

Pour un E Commerce, je pourrai ainsi exploiter les catégories de vêtements en BDD (mises au pluriel; les manteaux, les pantalons etc...) dans un fiche produit individuel (manteau trois-quart, pantalon Jean etc...)

Exemple avec le mot étés (dans `$chaîne`)

```
echo substr($chaîne, 0, -1);
```

En donnant une valeur négative au troisième argument (ici -1, pour stopper la chaîne à l'avant dernier caractère. Avec -2, ça sera l'antépénultième etc...) je supprime le s en dernière position

## **7-3 date()**

La fonction prédéfinie `date()` permet de récupérer la date du jour

```
echo date("d / m / Y");
```

Je lui donne trois arguments, `d` pour jour (day), `m` pour mois (month) et `Y` pour année (year). Le `/` n'étant qu'un séparateur pour formater l'affichage. J'aurai pu mettre un `-` à sa place, ne pas mettre d'espace etc ....

Noter aussi que j'aurai pu ne donner que deux arguments, pour n'afficher que le jour et le mois, ou le mois et l'année. Tout dépendra de mon besoin.

Autre remarque; si je décide d'écrire

```
echo date("D / M / y");
```

J'aurais toujours la date d'aujourd'hui, mais formatée autrement.

## 7-4 empty()

Lorsque nous avons travaillé sur les if/else, nous avons vu la fonction prédéfinie `isset` (et rapidement parlé de son équivalent `if_array_key_exists()` ). Elle permettait de vérifier si une variable existait

`empty()` va nous permettre de vérifier si une variable a un contenu (qui lui a été affecté) ou si elle est vide (utile plus tard lorsque nous déciderons qu'un formulaire a été correctement rempli et donc prêt à aller alimenter la base de données, ou s'il y a des erreurs qui doivent d'abord être corrigées avant son envoi...nous verrons cela plus tard)

Elle va prendre en argument (on peut aussi dire paramètre) la variable qu'elle doit analyser

Pour cet exemple, je vais réutiliser ma variable `$phrase`

```
if(empty($phrase)){  
    echo "Vrai, cette variable n'a pas de contenu";  
}else{  
    echo "Faux, cette variable a un contenu";  
}
```

Elle renvoie obligatoirement un **booléen**, VRAI ou FAUX

Je peux aussi fonctionner avec son contraire `!empty()`, c'est à dire demander à vérifier qu'elle possède un contenu.

Je travaille cette fois sur `$chaine`

```
if(!empty($chaine)){  
    echo "Vrai, cette variable a un contenu";  
}else{  
    echo "Faux, elle est vide";  
}
```

Elle renvoie encore un fois un booléen

## 8 Les fonctions utilisateur

[Documentation officielle](#)

Une **fonction utilisateur** est une fonction scriptée pour un besoin ponctuel et précis, par nos soins (par opposition à la fonction prédéfinie)

Ci-dessous, je déclare la fonction `salut()` (avec le mot clé/réserve `function`)

```
function salut($prenom){  
    echo "Bonjour " . $prenom;  
}  
salut('Aziz');
```



Dans sa parenthèse, je lui donne un argument/paramètre (\$prenom). Je ne suis pas obligé de donner un paramètre. Ce n'est pas une obligation, mais dans ce cas là, j'aurai du écrire `function salut()` sans rien préciser à l'intérieur. Avec cette syntaxe, je suis obligé de lui fournir ce paramètre.

J'ouvre ensuite des accolades. A l'intérieur je scripte tout le bloc d'instructions. C'est à dire toutes les lignes pour que cette fonction s'exécute.

Puis en dernier, j'appelle/exécute cette fonction, en lui donnant l'argument qu'elle attend

Selon les fonctionnalités de mon site, j'aurai besoin d'en créer un certain nombre

Dans le cadre d'une boutique, je scripterai par exemple une fonction pour ajouter un produit, modifier sa quantité ou tout simplement le supprimer de mon panier

De manière aussi très fréquente, je coderai une fonction pour que l'utilisateur se connecte sur mon site. Et dans la foulée, une autre qui distinguera les droits de cet utilisateur, pour pouvoir lui laisser l'accès à certaines pages mais pas à d'autres (s'il n'est pas administrateur du site, par exemple)

Ci-dessous nous allons coder une fonction qui nous permettra de calculer la TVA pour chaque produit vendu. Nous allons la rendre plus complexe et intéressante progressivement, sans aller trop loin.

Dans un premier temps, je la scripte de manière à ce qu'elle me calcule le prix, TVA incluse, d'un produit vendu 100€ HT.

```
function calculTva(){  
    return 100*1.2;  
}  
  
echo calculTva();
```

Ici, pas d'argument, par contre le mode de calcul dans le bloc d'instruction (avec `return` pour pouvoir ensuite l'afficher...il me le retournera lors de l'exécution de la fonction)

En tout dernier, hors du bloc fonction, j'appelle/exécute cette dernière (sans argument)

Je vais à présent la rendre plus intéressante en permettant de calculer différents prix, et non plus seulement 100€

Je vais devoir la déclarer avec un argument obligatoire lors de son exécution (et je dois aussi lui changer son nom en `calculTva2`, si je veux conserver la précédente)

```
function calculTva2($prix){  
    return $prix*1.2;  
}  
  
echo calculTva2(50);  
echo calculTva2(150);
```

Ainsi, au moment où je l'exécute, je peux lui passer en argument le prix que je veux être calculé.

Autre amélioration, nous allons permettre de calculer les prix, avec des taux de TVA différents (20% et 5,5%). Pour cela, je vais avoir besoin d'un second argument (le taux de TVA qui lui aussi pourra désormais varier)

Essayez de le faire seuls

```
function calculTva3($prix, $taux){  
    return $prix*$taux;  
}  
  
echo "50€ HT auxquels j'applique un taux de TVA de 5.5% donnera un prix de " .  
calculTva3(50, 1.055) . " TTC";  
echo "150€ HT auxquels j'applique un taux de TVA de 20% donnera un prix de " .  
calculTva3(150, 1.2) . " TTC";
```

Je lui donne un nouveau nom, pour conserver les deux autres versions. Je donne un second argument dans la parenthèse au moment de la déclaration de la fonction.

J'utilise les deux arguments dans le nouveau mode de calcul. Puis, au moment de l'exécution, je donne a chaque fois deux nombres. Le premier, pour alimenter `$prix`. Le second pour renseigner `$taux`

Dernière modification (pour que cela ne devienne pas trop compliqué pour un premier aperçu). Je vais faire en sorte qu'un taux de TVA soit appliqué par défaut si je ne renseigne pas cet argument au moment de l'exécution de ma fonction.

Voici sa syntaxe

```
function calculTva4($prix, $taux = 1.2){  
    return $prix*$taux;  
}  
  
echo calculTva4(50, 1.055) . "<br>";  
echo calculTva4(250) . "<br>";
```

Dans la parenthèse, j'indique désormais `$taux = 1.2`. Cela signifie que ça sera le taux par défaut. Et typiquement, comme je n'ai rien précisé lors de la seconde exécution, c'est ce taux par défaut qui sera appliqué.

Et il ne concernera pas la première exécution, qui a son propre taux de renseigné, et c'est donc celui qui sera appliqué et non celui par défaut

## 9 La portée des variables (espace global et espace local)

[Documentation officielle](#)

Ceci est un chapitre plutôt abstrait mais néanmoins important en programmation

Fait partie d'un espace local, tout ce qui est déclaré dans une fonction. Mon bloc d'instructions = espace local.

Devient espace global tout ce qui est extérieur à ce bloc d'instructions

Les valeurs déclarées dans un espace local ne seront accessibles dans l'espace global (et inversement) **qu'en utilisant une syntaxe spécifique**. C'est pour cela que l'on parle de **portée des variables**

```
-----espace global-----  
code  
  
function monScript{  
-----espace local -----  
    bloc d'instructions  
-----espace local -----  
}  
  
code  
  
-----espace global-----
```

## 9-1 Du global vers le local

Pour pouvoir récupérer la valeur de ma variable `$pays` (déclarée dans l'espace global ) à l'intérieur de ma fonction (espace local), je devrai utiliser le mot clé `global`

```
$pays = 'France';  
  
function affichePays(){  
    global $pays;  
    echo $pays;  
}  
  
affichePays();
```

Sans le mot clé `global` (retirez le pour faire un test ), ma variable ainsi que sa valeur deviennent inconnues dans mon bloc d'instructions → **Undefined variable \$pays**.

`global` me permet en quelque sorte de les importer

## 9-2 Du local vers le global

Inversement, pour récupérer la valeur de ma variable `$jour` déclarée dans l'espace local , je devrai utiliser le mot clé `return` . Je pourrai ainsi l'afficher avec `echo` dans mon espace global

```
function afficheJour(){  
    $jour = "mercredi";  
    return $jour;  
}
```

```
}  
echo afficheJour();
```

Sans, j'aurai droit comme tout à l'heure à **Undefined variable \$jour**

Le mot clé **return** sert d'exportateur

**Remarque:** dans le chapitre précédent sur les fonctions utilisateur, je n'avais pas ce problème d'espace local vs espace global. Tout simplement car aucune variable n'avait été déclarée dans l'un ou l'autre.

Je ne faisais que passer des arguments lors de l'exécution des fonctions

### Cas intéressant à étudier

```
function afficheJour2(){  
    $jour = "jeudi";  
    return $jour;  
    echo "demain c'est vendredi";  
}  
  
echo afficheJour2() . "<br>";
```

Pourquoi la suite du bloc d'instructions (echo «demain c'est vendredi») ne s'exécute pas, rien ne s'affiche hormis la valeur de la variable ?

C'est un exemple pour montrer qu'après un **return**, tout le code qui suit est comme désactivé. En fait, l'instruction **return** fait quitter la fonction !

## 10 Les boucles

Dans ce chapitre consacré aux **boucles itératives**, nous allons en voir 3. La **while**, la **do while**, la **for**. Il en existe une quatrième; la **foreach**. Très usitée et pratique. Nous l'étudierons dans un futur chapitre consacré aux **tableaux/arrays**

A l'instar des conditions, l'usage des différentes **boucles** sera incontournable pour un site dynamique

Pour reprendre l'exemple de l'affichage de mes différentes catégories de vêtements en BDD, pour par exemple les afficher dans une barre de navigation en tant qu'onglets, j'utiliserai la **boucle while**

Pour parcourir mon panier \$\_SESSION['panier'], je ferai appel à la **boucle for** etc...les boucles font partie intégrante de la construction d'un site web dynamique

## 10-1 La boucle while

[Documentation officielle](#)

Voici sa syntaxe, puis les explications

```
$i = 0;

while($i <= 5){
    echo "Tour " . $i . " --- ";
    $i++;
}
```

\* A la première ligne, j’initialise ma variable `$i` (i pour integer), mais je pourrai lui donner un autre nom, et je lui donne la valeur 0.

Je lui donne cette valeur car pour parcourir un tableau avec une boucle, il faut savoir que le premier indice aura pour valeur 0, le second aura la valeur de 1 etc...

### Contre exemple

Lorsque je fais de la pagination, j’appelle cette même variable `$page`, pour plus de clarté dans mon code. Et je l’initialiserai à 1 (avoir une page 0 n’aurait pas de sens)

Le nom de i et la valeur 0 ne sont pas obligatoires, mais vous verrez très fréquentes

\* A la seconde ligne je donne une condition à ma boucle while (**tant que**) entre ses parenthèses.

Je déclare que ma boucle doit exécuter son bloc d’instructions **tant que \$i est inférieur ou égal à 5**

\* Troisième ligne, dans les accolades, l’instruction echo (pour affiche moi) Tour suivi de la valeur de `$i` (au départ de 0) suivi de `---` (qui ne sont ici que des séparateurs, j’aurais pu écrire autre chose, comme `/` ou autre)

\* Quatrième ligne, l’instruction suivante est `$i++`, qui permet d’incrémenter sa valeur de +1 (j’aurai pu écrire `$i = $i + 1` ou `$i+=1`)

Une fois `$i` incrémenté, je retourne dans ma condition (ligne deux) pour vérifier qu’elle est toujours respectée, que ma boucle peut continuer à s’exécuter....c’est le cas, `$i` ayant pour valeur désormais 1.

Et ainsi de suite, avec Tour 2, puis Tour 3, en vérifiant à chaque tour de boucle que ma condition est respectée. Une fois `$i` ayant pris la valeur de 5, il s’incrémente et prend la valeur de 6.

Par contre, comme il ne vérifie plus ma condition (inférieur ou égal à 5), je n’entre plus dans son bloc d’instructions et je n’afficherai pas Tour 6 !

**Question:** comment faire pour que le dernier tour de boucle (qui va afficher Tour 5) ne soit pas suivi des tirets `---` ?

Variation/combinaison de la while avec un if/else

```
$i = 0;

while($i <= 5){
    if($i == 5){
```

```

    echo "Tour " . $i;
}else{
    echo "Tour " . $i . " --- ";
}
$i++;
}

```

Noter que l'incrémentation doit se situer à l'extérieur de la condition

## 10-2 La boucle do while

### Documentation officielle

Sa syntaxe ressemblera beaucoup à la while. Néanmoins elle est beaucoup moins utilisée

Nous allons l'étudier, mais plus à titre informatif, car vous ne la rencontrerez pas très souvent

```

$i = 0;

do{
    echo "Tour " . $i . " --- ";
    $i+=2;
}while($i <= 10);

```

Elle débute de la même manière, en l'initialisant. La différence se situe juste après.

Le bloc d'instructions débute avec **do** puis les accolades, puis les instructions d'affichage et d'incrémentation. Et ce n'est qu'à la fin qu'arrive la condition à respecter (avec **while**)

Noter l'incrémentation 2 par 2, aussi à titre indicatif

## 10-3 La boucle for

### Documentation officielle

Celle ci est en revanche très fréquente, avec le même objectif que la while; répéter une instruction tant que la condition pour stopper la boucle est respectée.

Sa syntaxe différera dans le sens où initialisation, condition et incrémentation se feront entre les parenthèses

```

for($i = 0; $i <= 5; $i++){
    echo " Tour " . $i;
}

```

La logique et le résultat étant par contre similaires

Voici un cas particulier intéressant.

Nous allons mettre à la disposition des utilisateurs un sélecteur pour choisir leur année de naissance. Et pour qu'il soit le plus pratique pour la grande majorité, il faudra que ce soit les années les plus récentes qui s'affichent en premier, et les plus anciennes en fin de sélecteur.

Pour cela, je vais devoir décrémenter

Je mets en place bien sur la structure du formulaire en premier, avec <form>, <select> et <option>

```
echo "<form>";
echo "<select>";
echo "<option selected>Sélectionnez l'année</option>";
for($annee = date('Y'); $annee >= date('Y') - 100; $annee--){
    echo "<option>$annee</option>";
}
echo "</select>";
echo "</form>";
```

Puis, dans ma boucle for j'initialise ma variable (volontairement nommée \$annee au lieu de \$i, pour bien illustrer que ce n'est pas une obligation). Je lui donne comme valeur initiale l'année 2022 grâce à la fonction prédéfinie date() vue il y a peu.

La condition sera que \$annee devra être supérieure ou égale à 2022 – 100, c'est à dire 1922 (je remonte 100 ans en arrière).

Puis je décrémante à chaque tour de boucle 2022, 2021, 2020 etc.

Une fois arrivé 1921, cette valeur ne respectant pas ma condition, l'instruction pour l'affichage ne s'exécutera pas, stoppant ainsi ma boucle.

## **10-4 Boucle imbriquée (double boucle for)**

Voyons en premier la boucle qui me permettrait d'afficher les valeurs de 1 à 10 dans un tableau

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

```
echo "<table border='1' style='border-collapse: collapse'>";
echo "<tr>";
for($i = 0; $i <= 9; $i++){
    echo "<td>" . $i + 1 . "</td>";
}
echo "</tr>";
echo "</table>";
```

Nous devons maintenant construire un tableau qui m'afficherait les dix premières valeurs dans une première ligne. Les dix suivantes (11 à 20) dans une seconde ligne. Les dix suivantes (21 à 30) dans une troisième ligne etc...

Jusqu'à une dixième et dernière ligne pour afficher les valeurs de 91 à 100

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Pour cela je vais devoir imbriquer deux for.

La première pour générer les lignes de 1 à 10. Et la seconde qui générera à l'intérieur de chaque ligne les cellules de 1 à 10

Voici sa syntaxe

```
echo "<table border='1' style='border-collapse: collapse'>";
for($ligne = 0; $ligne <=9; $ligne++){
    echo "<tr>";
    for($cellule = 0; $cellule <= 9; $cellule++){
        echo "<td style='padding: .5rem'>" . (10*$ligne+$cellule) + 1 . "</td>";
    }
    echo "</tr>";
}
echo "</table>";
```

## 11 Inclusions de fichiers

PHP permet l'inclusion de fichiers. C'est à dire que nous pouvons importer dans un fichier le contenu d'un autre fichier (et l'exécuter)

Dans un premier temps, à la racine de mon fichier actuel (entrainement.php) nous allons créer un dossier que nous nommerons **inc** (pour include/inclusion)

Dans ce dossier, créons un fichier que nous allons nommer **fichier.inc.php**

Le **.inc** n'est pas obligatoire. C'est une convention qui permet de repérer très vite que ce fichier n'est pas destiné à s'afficher en tant que page, mais à être inclus dans un autre fichier



Voici son contenu

```
<p>
<strong>Voici mon contenu</strong><br>
<em>Rappel: je suis du code HTML exclusivement</em>
</p>
```

Ceci pour rappeler que bien que mon fichier soit une extension `.php`, je peux tout à fait ne mettre que du code HTML, sans aucun traitement PHP à l'intérieur

Voici à présent une des syntaxes pour inclure/importer son contenu dans mon fichier principal

```
echo "Je suis le fichier <mark>entrainement.php</mark>";
include('inc/fichier.inc.php');
echo "Je suis la suite du fichier <mark>entrainement.php</mark>";
```

`include()` permet d'inclure cette inclusion. Il en sera de même avec `require()`. Ces deux fonctions prédéfinies font partie des [structures de contrôle](#)

La différence entre les deux réside dans le fait qu'avec `include()`, l'erreur sera notifiée, mais n'empêchera pas la suite du code de s'exécuter

Avec `require()`, s'il y a une erreur dans l'exécution du code, toute la suite du script sera bloquée.

Deux autres expressions permettent de s'assurer que le fichier appelé ne pourra être inclus une seconde fois, il s'agit de `include_once()` et `require_once()`

Logiquement, nous utiliserons une de ces deux dernières pour appeler le header ou le footer

Ainsi, si le fichier n'a jamais été appelé, il sera inclus. En revanche, s'il a déjà été appelé, alors `require_once()` ou `include_once()` ignorera cette (seconde) inclusion.

Inclure ces deux éléments est très pratique. Si, ultérieurement, nous devons procéder à une modification (ajout, suppression d'onglets dans la barre de navigation, par exemple) ou tout autre correction dans le code, nous n'aurons désormais à le faire qu'une seule fois, dans le bon fichier

A charge pour le `require_once` de répercuter cette mise à jour sur toutes les pages du site, plutôt que de devoir procéder à cette même modification sur autant de fichiers que compte le site de pages

Testons cette solution ensemble, en créant tout d'abord un `indexSite.php`

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Inclusion</title>

  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
1BmE4kWbQ78iYhF1dvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
```

```
</head>
<body>
  <header>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
      <div class="container-fluid">
        <a class="navbar-brand" href="#">Navbar</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link active" aria-current="page" href="#">Home</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">Link</a>
            </li>
            <li class="nav-item dropdown">
              <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown"
role="button" data-bs-toggle="dropdown" aria-expanded="false">
                Dropdown
              </a>
              <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
                <li><a class="dropdown-item" href="#">Action</a></li>
                <li><a class="dropdown-item" href="#">Another action</a></li>
                <li><hr class="dropdown-divider"></li>
                <li><a class="dropdown-item" href="#">Something else here</a></li>
              </ul>
            </li>
            <li class="nav-item">
              <a class="nav-link disabled">Disabled</a>
            </li>
          </ul>
          <form class="d-flex">
            <input class="form-control me-2" type="search" placeholder="Search"
aria-label="Search">
            <button class="btn btn-outline-success" type="submit">Search</button>
          </form>
        </div>
      </div>
    </nav>
  </header>

  <div class="container">
    <h1>Mon site</h1>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
```

```

molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>
</div>

<footer>
    <p>&copy; 2022 Aziz</p>
</footer>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>

</body>
</html>

```

Je vais a présent découper cette page en trois. `header.inc.php`, `indexSite.php` et `footer.inc.php`

Les premiers et derniers étant créés dans mon dossier `inc`

Dans `header.inc.php` je copie colle le contenu de mon index, de la première ligne jusqu'à la balise ouvrante `<div=«container»>` (et je le supprime de index)

dans `footer.inc.php`, je copie colle le contenu de index a partir de la balise fermante `</div>` (celle de container jusqu'à la dernière ligne (et je le supprime de index)

Désormais, voici le contenu de mes trois fichiers

`header.inc.php`

```

<!DOCTYPE html>
<html lang="fr">

```

```
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Inclusion</title>

  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">

</head>
<body>
  <header>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
      <div class="container-fluid">
        <a class="navbar-brand" href="#">Navbar</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
              <a class="nav-link active" aria-current="page" href="#">Home</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">Link</a>
            </li>
            <li class="nav-item dropdown">
              <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown"
role="button" data-bs-toggle="dropdown" aria-expanded="false">
                Dropdown
              </a>
              <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
                <li><a class="dropdown-item" href="#">Action</a></li>
                <li><a class="dropdown-item" href="#">Another action</a></li>
                <li><hr class="dropdown-divider"></li>
                <li><a class="dropdown-item" href="#">Something else here</a></li>
              </ul>
            </li>
            <li class="nav-item">
              <a class="nav-link disabled" href="#">Disabled</a>
            </li>
          </ul>
          <form class="d-flex">
            <input class="form-control me-2" type="search" placeholder="Search"
aria-label="Search">
            <button class="btn btn-outline-success" type="submit">Search</button>
          </form>
        </div>
      </div>
    </nav>
  </header>
</body>
```

```
        </form>
    </div>
</div>
</nav>
</header>

<div class="container">
```

indexSite.php

```
<h1>Mon site</h1>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>
```

Et footer.inc.php

```
</div>

<footer>
    <p>&copy; 2022 Aziz</p>
</footer>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYs0g+0MhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>
```

```
</body>
</html>
```

A présent, si je rafraîchis ma page index, je perds bien évidemment tout le contenu retiré

Pour le récupérer le vais utiliser require\_once()

```
<?php require_once('inc/header.inc.php') ?>

<h1>Mon site</h1>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Odit eaque pariatur
alias suscipit, consequatur minima ad. Sapiente, incidunt! Itaque mollitia,
molestias repudiandae id voluptate animi quia. Vel quisquam consequatur
dolores.</p>

<?php require_once('inc/footer.inc.php') ?>
```

Continuons notre test/démonstration en créant a présent deux nouveaux fichiers, page1.php et connexion.php

```
<h1>Ma page 1</h1>

<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,
corrupti.</p>

<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,
corrupti.</p>
```

```
<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat  
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel  
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,  
corrupti.</p>  
<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat  
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel  
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,  
corrupti.</p>  
<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat  
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel  
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,  
corrupti.</p>  
<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat  
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel  
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,  
corrupti.</p>  
<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat  
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel  
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,  
corrupti.</p>  
<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat  
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel  
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,  
corrupti.</p>  
<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat  
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel  
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,  
corrupti.</p>  
<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat  
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel  
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,  
corrupti.</p>  
<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat  
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel  
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,  
corrupti.</p>
```

puis connexion.php (formulaire pris sur Bootstrap)

```
<form>  
  <div class="mb-3">  
    <label for="exampleInputEmail1" class="form-label">Email address</label>  
    <input type="email" class="form-control" id="exampleInputEmail1" aria-  
describedby="emailHelp">  
    <div id="emailHelp" class="form-text">We'll never share your email with anyone  
else.</div>  
  </div>
```

```

<div class="mb-3">
  <label for="exampleInputPassword1" class="form-label">Password</label>
  <input type="password" class="form-control" id="exampleInputPassword1">
</div>
<div class="mb-3 form-check">
  <input type="checkbox" class="form-check-input" id="exampleCheck1">
  <label class="form-check-label" for="exampleCheck1">Check me out</label>
</div>
<button type="submit" class="btn btn-primary">Submit</button>
</form>

```

Nous allons maintenant appeler les header et footer pour chacun d'eux

```

<?php require_once('inc/header.inc.php') ?>

<h1>Ma page 1</h1>

<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,
corrupti.</p>
...
<p>Lorem ipsum dolor, sit amet consectetur adipisicing elit. Magni repellat
perferendis cum qui deserunt minus suscipit blanditiis, possimus iusto vel
pariatur, excepturi commodi itaque iste expedita tenetur cumque. Explicabo,
corrupti.</p>

<?php require_once('inc/footer.inc.php') ?>

```

Cette manipulation me permet non seulement de ne pas avoir à recoder tout le contenu de header et footer pour chaque page nouvellement générée, mais en plus, si je dois procéder à une modifications dans mon footer, ma nav, je n'aurai à le faire une seule fois (changer de logo, relier une nouvelle page dans ma nav etc.)

Dans le même ordre d'idées, pour optimiser la maintenance du site, je pourrai inclure d'autres fichiers encore. Par exemple un fichier appelé init.inc.php dans lequel j'aurai initialisé la connexion à la base de données, déclaré mon session\_start() etc... Sinon, je devrai le faire au début de chaque fichier !

Je pourrai également inclure un fichier fonctions.inc.php (dans lequel seront scriptées toutes les fonctions dont j'aurai besoin)

## 12 Arrays

Un tableau array, autrement appelé tableau de données, est un **type** de variable comme un int, string etc...



Ce type particulier permet de stocker plusieurs valeurs alors que les autres (int, string etc.) n'en contiennent qu'une seule

Si je veux stocker une liste de prénoms dans une variable «classique», cela n'aurait pas de sens (ou d'utilité)

```
$listePrenoms = "Inès, Yann, Aziz, Julien, Mathilde";  
echo $listePrenoms;
```

Nous avons vu dans un cas précédent, que je ne pouvais les dissocier. Ils forment un tout, ne sont pas exploitables individuellement.

Un array va permettre cela en revanche

Voici une des syntaxes, la plus basique, pour déclarer un array.

Il faut pour cela introduire notre tableau par le mot clé **array**, puis introduire nos valeurs entre parenthèses, chacune séparées par une virgule

```
$tableauPrenoms = array('Inès', 'Yann', 'Aziz', 'Julien', 'Mathilde');
```

Pour une meilleure lisibilité, nous pourrions rencontrer cette syntaxe aussi, si la liste est longue

```
$tableauPrenoms = array(  
    'Inès',  
    'Yann',  
    'Aziz',  
    'Julien',  
    'Mathilde'  
);
```

Désormais, je peux appeler chaque valeur individuellement.

## **12-1 Print\_r()**

Mais dans un premier temps, voici une première façon d'afficher toutes les valeurs de mon tableau

Je vais utiliser l'instruction **print\_r()**, pour un affichage de toutes les valeurs. **Affichage non conventionnel**, non destiné aux grand public, aux utilisateurs de mon site.

Mais pour mon usage personnel, durant la phase de développement de mon site (nous verrons dans un second temps comment afficher ces données au grand public)

Voici la syntaxe pour **print\_r()**

```
echo "<pre>"; print_r($tableauPrenoms); echo "<pre>";
```

J'aurai pu écrire ceci à la place

```
echo print_r($tableauPrenoms);
```

Mais il est préférable d'entourer le `print_r()` de la balise `<pre>` pour une meilleure lisibilité du résultat d'affichage (faites le test en la supprimant)

`print_r()` me confirme donc que c'est un type array (et donc un tableau) qui contient mes valeurs

Et devant chacune, figure un chiffre entre crochets. C'est leur `indice (ou clé)`.

La valeur première, Inès, possède l'`indice 0`. La seconde valeur, Yann, possède l'`indice 1` etc ...

## **12-2 var\_dump()**

Une seconde instruction, `var_dump()` permettra d'afficher les valeurs contenues dans mon tableau, avec encore plus d'informations.

```
echo "<pre>"; var_dump($tableauPrenoms); echo "<pre>";
```

Une première information supplémentaire m'est proposée en précisant que mon array contient 5 valeurs.

Devant chacune d'entre elles, j'ai aussi son type (toutes des strings), ainsi que la longueur de la chaîne de caractères

**Remarquons** que pour Inès, elle n'est pas de 4...mais 5, à cause de l'accent, déjà vu avec la fonction prédéfinie `strlen()`

**Important:** `print_r()` et `var_dump()`, font partie des [fonctions de gestion de variables](#)

## **12-3 Affichage d'une seule valeur**

Nous venons de voir comment les afficher toutes, voici la syntaxe pour en récupérer une seule

Je vais pour cela utiliser l'`indice (ou clé)` de la valeur qui m'intéresse

```
echo $tableauPrenoms[1];
```

Ci-dessus, j'appelle mon tableau, et entre crochets, j'indique son indice (ici pour récupérer la valeur Yann).

On dit que je crochète à son indice

## **12-3 Autre syntaxe pour un tableau**

Voici une autre syntaxe pour déclarer un tableau, intéressante si je ne connais pas à l'avance les valeurs, je pourrais les ajouter ainsi au fur et à mesure de mon code (séparées les unes des autres par d'autres lignes de code, sans remonter à la déclaration d'origine et ajouter une valeur)

```
$listePays[] = "France";  
$listePays[] = "Algérie";  
$listePays[] = "Italie";  
$listePays[] = "Espagne";  
$listePays[] = "Russie";
```

Son print\_r() pour vérifier

```
echo "<pre>"; print_r($listePays); echo "</pre>";
```

Quelle syntaxe dois-je utiliser pour récupérer la valeur Espagne ?

```
echo $listePays[3];
```

C'est effectivement son indice (3), en tant que 4ème valeur, en gardant en mémoire qu'en permanence, la première valeur aura pour clé ou indice 0

## 12-4 Parcourir un tableau avec la boucle foreach

L'occasion nous est donnée ici de découvrir une nouvelle boucle, la **foreach**, très adaptée pour parcourir et afficher un tableau de manière conventionnelle.

Sa syntaxe est très simple à mettre en place, mais un peu abstraite à comprendre.

Nous allons détailler cela. Voici tout d'abord sa syntaxe, puis nous l'expliquerons

```
foreach($listePays as $indice => $valeur){  
    echo "<p>L'indice " . $indice . " à pour valeur le pays " . $valeur . "</p>";  
}
```

Je récupère chaque valeur avec la balise **<p>**, mais j'aurai aussi pu utiliser un système de liste avec **<ul>** et **<li>**

**Les explications:**

Je déclare bien sur au préalable ma boucle **foreach**, et la syntaxe entre parenthèses mérite toute notre attention

J'appelle mon tableau, **\$listePays** auquel je donne un **alias \$indice** (avec **as**). Et je demande à faire correspondre chaque **\$indice** rencontré dans le tableau, sa valeur (**\$valeur**) avec le signe **=>**

Cette syntaxe est primordiale dans le sens où ce sont les 5 informations (**\$listePays**, l'**alias**, le **as**, la **valeur** que je fais correspondre avec **=>**) attendues par la **foreach** pour pouvoir travailler et afficher ce que je lui demande...

**Cependant**, les trois éléments les plus importants sont **\$listePays**, **as** et **=>**. **\$indice** et **\$valeur** ne sont que des noms de variables que je peux modifier, même de manière loufoques, cela n'aura aucune incidence vis à vis de la **foreach** qui continuera à effectuer son travail de la même manière (voir ci-dessous)

```
foreach($listePays as $voiture => $moto){
    echo "<p>L'indice " . $voiture . " à pour valeur le pays " . $moto . "</p>";
}
```

Le résultat sera strictement identique. Testez !

C'est la preuve que la foreach a besoin des 5 éléments, mais de manière immuable, ce sont les éléments 1, 2 et 5 que je ne pourrai pas modifier dans cette syntaxe.

Par contre, il est entendu que je dois donner des noms cohérents, pertinents à mes variables. `$indice` et `$valeur` sont des nommages pertinents.

Nous rencontrerons aussi très souvent le nommage `$key` (pour `$indice`) et `$value` (pour `$valeur`). Et nous les utiliserons souvent dans nos futurs scripts. Par convention.

## 12-5 Fonctions prédéfinies utiles

\* implode() ([Documentation officielle](#))

Si `implode()` n'est pas listée en tant que fonction sur les tableaux (mais sur les [chaines de caractères](#)), elle permet néanmoins l'affichage de tous les éléments d'un tableau sous forme de chaîne

**Notez** le séparateur qu'elle prend en argument pour "aérer" les éléments entre eux (qui n'est pas obligatoire en tant qu'argument, mais tellement utile qu'il le devient)

```
echo "<p>" . implode(' - ', $listePays) . "</p>";
```

\* count() et sizeof()

`count()` et `sizeof()` en revanche sont deux [fonctions sur les tableaux](#) (il en existe beaucoup d'autres)

Chacune permet de compter les éléments d'un table, et c'est `count()` que nous utiliserons le plus souvent

```
echo count($listePays);
echo sizeof($listePays);
```

## 12-6 Boucle for pour un tableau

Nous allons à présent utiliser `count()` pour parcourir et afficher notre tableau avec une boucle `for`

Voici la syntaxe, avec affichage du résultat dans une liste

```
for($i = 0; $i < count($listePays); $i++){
    echo "<ul>";
    echo "<li>" . $listePays[$i] . "</li>";
    echo "</ul>";
}
```

`count()` servant donc à calculer le nombre d'indices pour donner une limite à ma boucle

## 12-7 Tableau associatif

Si je veux, si j'en ai besoin, je peux décider d'associer à mes valeurs, un indice choisi par mes soins, plutôt que l'indice numérique par défaut qui leur sera affilié

Mon array devient un tableau associatif.

Voici la syntaxe pour faire cela, ainsi que le `print_r()` pour vérifier le résultat

```
$listeCouleurs = array("j" => "jaune", "b" => "bleu", "v" => "vert");  
  
echo '<pre>'; print_r($listeCouleurs); echo '</pre>';
```

Et par conséquent, désormais, si nous voulons récupérer une valeur en particulier, nous ne crocheterons plus à un indice numérique, mais à l'indice lettre que nous lui avons affilié au préalable.

```
echo $listeCouleurs['b'];
```

Cela peut s'avérer utile dans certains cas, mais ce n'est pas une pratique généralisée

**Cas particulier** concernant la syntaxe ci-dessous

```
$listeFruit = array("fruit1" => "orange", "fruit2" => "pomme", "fruit3" =>  
"fraise"); // déclaration array  
echo 'fruit1 : ' . $listeFruit['fruit1'] . '<br>'; // quotes dans les crochets  
echo "fruit1 : $listeFruit[fruit1] <br>"; // pas de quotes dans les crochets
```

Dans le premier `echo`, nous utilisons de **simples quotes** qui entraînent une **concaténation** (sinon ma variable ne sera pas interprétée). Dans ce cas précis, nous devons ajouter des **quotes entre crochets**

Dans le second `echo`, nous avons fait appel à des  **doubles quotes**, qui évitent de devoir concaténer. Et dans ce cas là, pas besoin de quotes entre les crochets.

## 12-8 Tableau multi-dimensionnel

Première syntaxe pour déclarer un **tableau multi-dimensionnel**

```
$marvel_hero = array(  
    array('personnage' => 'Spiderman', 'prénom' => 'Peter', 'nom' => 'Parker'),  
    array('personnage' => 'Iron Man', 'prénom' => 'Tony', 'nom' => 'Stark'),  
    array('personnage' => 'Veuve Noire', 'prénom' => 'Natasha', 'nom' => 'Romanoff'),  
);  
  
echo "<p>" . $marvel_hero[2]['prénom'] . '</p>';  
  
echo '<pre>'; var_dump($marvel_hero); echo '</pre>';
```

Seconde façon, avec un tableau associatif, c'est à dire que je vais choisir le nom de l'indice plutôt que de laisser l'indice numérique par défaut.

Cela va générer un `var_dump` différent, ainsi qu'une autre syntaxe pour crocheter à l'indice désiré

```
$marvel_hero = array(
    "Spiderman" => array("prénom" => "Peter", "nom" => "Parker"),
    "Iron Man" => array("prénom" => "Tony", "nom" => "Stark"),
    "Veuve Noire" => array("prénom" => "Natasha", "nom" => "Romanoff"),
);

echo "<p>" . $marvel_hero['Iron Man']['prénom'] . "</p>";

echo "<pre>"; var_dump($marvel_hero); echo "</pre>";
```

Je peux exploiter la `foreach` (en fait une double imbriquée), pour parcourir les éléments de mon tableau et les afficher dans une liste

Voici en premier la syntaxe, puis les explications

```
foreach($marvel_hero as $key => $value){
    echo '<ul>';
    if(is_array($value)){
        echo '<li>';
        echo "<h4> $key </h4>";
        foreach($value as $key2 => $value2){
            echo "<p>" . $key2 . ": " . $value2 . "</p>";
        }
        echo '</li>';
    }
    echo '</ul>';
}
```

De manière basique pour ma première `foreach`, je donne un `alias` à mon tableau (`$marvel_hero`) et je lui fais correspondre sa `valeur` avec `=>`

Je fais ensuite appel à une condition `if` pour savoir si la valeur du tableau `$marvel_hero` est ou pas un autre tableau ( `is_array($value)` ) ?

C'est effectivement le cas (tableau imbriqué / multi-dimensionnel)

Alors je récupère l'indice (nominatif) de chaque tableau pour mon H4 (ça sera Spiderman, Iron Man et Veuve Noire ) avec ma première boucle.

Et je scripte ma seconde boucle `foreach` pour récupérer les indices et valeurs à l'intérieur du second tableau

Et toujours de manière classique, je donne un `alias` `$key2` à mon tableau (imbriqué `$value` désormais) auquel je fais correspondre sa valeur `$value2` avec `=>`

**Noter** que les nommages `key2` et `value2` sont totalement arbitraires de ma part.

J'aurais pu les nommer key et value une nouvelle fois (comme pour le premier tableau), cela aurait fonctionné. Mais cela aurait pu laisser penser que c'est une syntaxe obligatoire les concernant.

C'est pour cela que j'ai légèrement changer leur nom, pour qu'il n'y ai pas de confusion.

Et une nouvelle fois, ils auraient pu être nommés voiture et moto

Je laisse key2 et value2, mais cela aurait très bien fonctionné key et value, pas de conflit avec les noms du premier tableau marvel\_hero

## 13 Classes et objets

[Documentation officielle](#)

Ce chapitre est une introduction au **PHP Orienté Objet**

La **Programmation Orientée Objet** s'est progressivement imposée, prenant le dessus sur la logique **procédurale**, pour des raisons de maintenance de code.

Même si cette dernière reste valide, elle est désormais "désuète" pour du développement moderne

Elle est basée sur **4 principes** : Abstraction, Encapsulation, Héritage et Polymorphisme

Je ne vais, pour cette introduction à la **POO**, qu'évoquer le principe d'**encapsulation** en créant une **classe** (ici, Habitant), qui va regrouper toutes les spécificités d'un même **objet**

Désormais, en **POO** je ne parlerai plus de variables, mais de **propriétés** (ou **attributs**). Le terme **fonction** sera remplacé par **méthode**

Leur utilité sera la même qu'en procédural, c'est simplement un changement de vocabulaire

Pour résumer, une classe **encapsule** (regroupe) toutes les caractéristiques (propriétés comme méthodes) d'un sujet particulier (dans un souci de maintenance du code)

### Important

Je vais continuer à travailler dans ce même fichier entrainement.php, mais il faut savoir que pour chaque classe il faut créer son propre fichier (du même nom que la classe...ici je vais coder la **classe Habitant**, il aurait fallu le faire dans un fichier nommé **habitant.php**)

```
class Habitant{
    public $prenom = 'Aziz';
    public $age = 53;

    public function ville(){
        return 'Créteil';
    }
}
```

Une classe s'introduit avec le mot clé class et son nom devra obligatoirement débuter par une majuscule (par convention)

Dans cette classe, je déclare deux attributs/propriétés `$prenom` et `$age`. Et plus en-dessous une méthode `ville()`

Un attribut permet globalement de décrire le sujet et la méthode pour lui donner un comportement

A mes attributs et méthode, j'ai affecté le mot-clé `public`

C'est le niveau de visibilité le plus "permissif" (visible par tous)

Il en existe deux autres, plus fréquemment utilisés: `protected` et `private` (de plus en plus restrictifs, pour protéger le code)

`public` va me permettre de faire les manipulations que je veux, sans trop rendre complexe cette initiation

Pour exploiter les caractéristiques de ma classe, je vais devoir l'`instancier`. Cela veut dire que je vais devoir `créer un objet` à partir de cette classe. Voici la syntaxe pour le faire

```
$habitant = new Habitant;
```

je le fais en utilisant le mot clé `new`. `$habitant` est un objet de ma classe. C'est une `instance` de ma classe `Habitant`

Je vais à présent faire un `var_dump` de cet objet

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

Première constatation, le `var_dump` m'indique que je suis face à un objet de ma classe `Habitant`

Voici la syntaxe pour récupérer les données si je veux les afficher

```
echo "Je me prénomme " . $habitant->prenom . ", j'ai " . $habitant->age . " ans et  
je réside à " . $habitant->ville();
```

L'élément `→` (flèche) permet à l'`objet instancié` (`$habitant`) d'atteindre tous les `attributs` (ou `propriétés`) ainsi que les méthodes encapsulées dans sa classe

Notez que si pour atteindre un attribut (prenom ou age) je ne fais pas appel à `$`, en revanche pour atteindre une méthode (ville), je ne dois pas oublier les parenthèses ouvrantes et fermantes (sinon, cela reviendrait à vouloir atteindre un attribut ... qui n'existe pas !)

Pour résumer:

Je peux considérer que ma classe est un moule, un modèle, qui servira à créer différents objets, que je pourrai distinguer les uns des autres (prénoms différents(nous verrons comment plus tard), ages, couleurs, vitesses, poids, etc...selon les attributs que j'aurai déclaré dans ma classe)

## 14 Les Superglobales

[Documentation officielle](#)



Les **Superglobales** sont des variables (prédéfinies par PHP) de **type array** (elles permettent de conserver différentes valeurs)

Du fait aussi que ce soit des arrays, je pourrai utiliser un **print\_r** ou **var\_dump** pour lire les infos qu'elles contiennent

Leur particularité étant d'être aussi disponibles partout, c'est à dire **disponibles dans l'espace global comme local**, sans avoir à faire un passage d'arguments ou utiliser le mot clé global

Au niveau de la syntaxe, toutes commencent par un **\_** (underscore), hormis **\$GLOBALS**, et sont écrites en majuscules

Superglobale	Description	Exemple d'utilisation
<b>\$GLOBALS</b>	Contient toutes les variables disponibles dans un contexte global	-
<b>\$_SERVER</b>	Contient toutes les informations fournies par le serveur web	Pratique pour connaître le chemin du site, d'un dossier, etc.
<b>\$_GET</b>	Contient les informations fournies en paramètre au script via la méthode GET par l'URL et le protocole HTTP.	Utile pour véhiculer des informations d'une page à l'autre.
<b>\$_POST</b>	Contient les informations fournies par un formulaire via la méthode POST du protocole HTTP.	Utile pour récupérer les saisies postées dans un formulaire par un internaute.
<b>\$_FILES</b>	Contient les informations liées à l'upload d'un (ou plusieurs) fichier(s) par un formulaire (fonctionne en complément de la superglobale \$_POST).	Utile pour récupérer le(s) fichier(s) uploadé(s) dans un formulaire par un internaute.
<b>\$_COOKIE</b>	Contient les informations fournies par les cookies via le protocole HTTP.	Utile pour conserver des informations sur un internaute.
<b>\$_SESSION</b>	Contient les informations de la session en cours.	Utile pour maintenir une connexion avec un internaute sur un site web
<b>\$_REQUEST</b>	Contient les variables fournies au script (peu importe la méthode utilisée).	Utile pour récupérer des informations sans savoir précisément d'où elles proviennent
<b>\$_ENV</b>	Contient les variables fournies par l'environnement.	-

## 14-1 Aperçu du contenu de quelques unes

### \* \$GLOBALS

C'est donc la seule sans le underscore et elle a aussi la particularité de regrouper toutes les autres, dont elle même (signification du RECURSION lors de son affichage)

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

Si je teste d'autres tels que **\$\_ENV** ou **\$\_REQUEST**, ces tableaux/arrays ne contiennent rien pour l'instant

### \* \$\_SERVER

Celle-ci contient toutes les informations concernant le serveur

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

Il en est une particulièrement intéressante pour nous, c'est celle qui possède l'**indice DOCUMENT\_ROOT** car elle contient le chemin physique vers httdocs

```
[DOCUMENT_ROOT] => C:/xampp/htdocs
```

Nous pourrions par exemple nous en servir dans une constante pour définir le chemin vers notre projet.

Il nous suffira de crocheter dans notre array `$_SERVER` au bon indice, d'y ajouter le nom de notre dossier `/boutique/`

```
define('RACINE_SITE', $_SERVER['DOCUMENT_ROOT'] . '/boutique/');
```

Pour ensuite disposer du contenu de notre constante `RACINE_SITE` sur toutes les pages de notre site sans avoir à rappeler in-extenso ce chemin

Les autres **Superglobales** vont être détaillées chacune dans leur chapitre

## 15 Les méthodes GET et POST

### 15-1 Méthode GET

[Documentation Officielle](#)

Montrer le one-page **Au Pois Gourmand** + **La Boutique** pour expliquer l'utilité

La Superglobale `$_GET` permet de faire transiter des informations d'une page web vers l'autre par le biais de l'URL. La syntaxe pour les acheminer est relativement simple

Pour faire notre démonstration nous allons devoir créer un autre fichier pour accueillir ces informations (appelons le `page_get.php`)

Maintenant, dans mon fichier `entrainement.php`, dans l'attribut `href` de ma balise `<a>`, j'indique tout d'abord le fichier de destination (en l'occurrence `page_get.php`). Il devra être suivi obligatoirement de `?` puis des informations à "transporter"

Ces informations sont en fait, avant le `=`, l'**indice**, et après le signe égal, sa **valeur**

**N'oublions pas**, je suis dans un tableau

Chaque nouvelle information devant être séparée par un `&` de la précédente.

```
echo '<button><a href="page_get.php?produit=Gateau&variete=chocolat&prix=12">Tester</a></button>';
```

Voici maintenant le script concernant ma page d'accueil

```
<?php
if($_GET){
    echo $_GET['produit'] . " au " . $_GET['variete'] . "<br>";
    echo "Vendu " . $_GET['prix'] . " €, aujourd'hui seulement !";
}
```

Tout d'abord, et **obligatoirement** (même si cela va fonctionner sans dans ce cas précis) il faudra débiter notre script par `if($_GET){}`

Cela permet de se prémunir d'une erreur PHP dans le cas où la page a été chargée sans qu'aucune information n'ait transité via l'URL

Une fois cette précaution prise, je pourrai récupérer les informations désirées dans ce nouveau fichier grâce à la Superglobale `$_GET` en crochétant au bon indice

Cette **méthode GET** sera très utile dans le cadre d'un E Commerce.

Elle permettra par exemple la transition d'une page où sont affichés tous les articles à vendre, vers la fiche du produit qui m'intéresse (en récupérant toutes les infos qui le concerne, photo comprise)

Je ne ferai par contre transiter qu'une seule information le concernant dans l'URL, son ID.

J'utiliserai ensuite ce dernier pour récupérer les informations en BDD liées à lui

## **15-2 La méthode POST**

### [Documentation officielle](#)

La méthode `$_POST` permet de transmettre les données récoltées dans un formulaire, vers ma base de données ou vers une autre page

Sans PHP et cette méthode, un formulaire codé en HTML/CSS n'a pas grande utilité autre que formelle.

Voici le script d'un premier formulaire (dans du code HTML, avec à l'intérieur son propre passage PHP)

```
<form method="POST" action="">

  <label for="prenom">Prénom</label>
  <input type="text" name="prenom" id="prenom" placeholder="Votre prénom">

  <label for="description">Description</label>
  <textarea name="description" id="description" rows="3"
placeholder="Description">
</textarea>

  <label for="annee">Année de naissance</label>
  <select name="annee" id="annee">
    <?php
      for($i = date('Y'); $i >= date('Y') - 100; $i--){
        echo '<option>' . $i . '</option>';
      }
    ?>
  </select>
  <br>
  <input type="submit" value="Soumettre">
</form>
```

### Plusieurs choses à noter:

\* En tout premier, le `champs method` à renseigner. Je pourrai indiquer POST ou GET. Dans notre cas, ça sera en permanence POST, sachant par ailleurs que faire transiter des informations d'un formulaire via l'URL pourrait s'avérer sensible

\* Concernant le `champs action`; je le renseignerai si je décidais de récupérer les informations récoltées dans le formulaire sur une autre page. J'y indiquerai par exemple:

`action="monAutrePage.php"`

Mais comme je vais tout récupérer cela sur `entrainement.php`, je laisse ce champ vide.

\* Indifféremment pour les balises `<label>`, `<input>`, `<textarea>`, `<select>` et autres encore, c'est l'attribut `name` qui va être le plus important concernant l'envoi et la récupération des données à partir d'un formulaire.

Les deux autres attributs `for` et `id` ont leur importance (ils sont d'ailleurs liés l'un à l'autre) mais par pour PHP.

C'est `name` qui va jouer un rôle décisif. Sans lui, pas de possibilité de relier notre formulaire à la page qui va réceptionner les données ou la BDD qui va les stocker

Voici le code pour récupérer les données du formulaire

```
<ul>
<li>Prénom: <?= $_POST['prenom'] ?></li>
<li>Description: <?= $_POST['description'] ?></li>
<li>Année de naissance: <?= $_POST['annee'] ?></li>
</ul>
```

Une fois de plus, je crochète à l'indice désiré (le `name` en fait que j'ai donné) de ma Superglobale/array `$_POST`

Si je rafraîchis ma page pour afficher cette nouvelle zone, je vais avoir un warning erreur de PHP

`Undefined array key "prenom".`

C'est normal. Au chargement de ma page, je n'ai pas encore rempli le formulaire et donc l'indice `prenom` et les autres sont encore inconnus. Ils ne deviendront connus qu'à partir du moment où j'aurai rempli et validé le formulaire.

Pour ne plus avoir ce problème, comme pour la méthode GET, je vais devoir mettre le code pour récupérer les données dans une condition

```
<?php if($_POST): ?>
<ul>
<li>Prénom: <?= $_POST['prenom'] ?></li>
<li>Description: <?= $_POST['description'] ?></li>
<li>Année de naissance: <?= $_POST['annee'] ?></li>
</ul>
<?php endif; ?>
```

### Remarque

Si je dois scripter plusieurs formulaires sur une même page, alors il sera nécessaire de donner un `name` à l'input submit

```
<input type="submit" name="nomDuBouton" value="Soumettre">
```

Ainsi, je pourrais crocheter ce nom dans le `if($_POST['nomDuBouton'])`. Sans cela, tous mes formulaires seront activés en même temps, sans discernement

#### \* Envoi des informations sur une autre page

De manière généralisée, les données collectées dans un formulaire sont envoyées en BDD.

Néanmoins nous allons voir ici comment les transmettre sur une autre page.

Je nomme ce fichier `recupForm.php`

Je reproduis le précédent formulaire à l'identique, en renseignant le action `action=recupForm.php` et en donnant un name au bouton `bouton2`

```
<form method="POST" action="recupForm.php">

  <label for="prenom">Prénom</label>
  <input type="text" name="prenom" id="prenom" placeholder="Votre prénom">

  <label for="description">Description</label>
  <textarea name="description" id="description" rows="3"
placeholder="Description">
  </textarea>

  <label for="annee">Année de naissance</label>
  <select name="annee" id="annee">
    <?php
      for($i = date('Y'); $i >= date('Y') - 100; $i--){
        echo '<option>' . $i . '</option>';
      }
    <?>
  </select>
  <br>
  <input type="submit" name="bonton2" value="Soumettre">
</form>
```

Voici le code pour la page qui récupère les informations

```
<?php

if($_POST){
  echo "<ul>";
  echo "<li>Prénom: $_POST[prenom]</li>";
  echo "<li>Description: $_POST[description]</li>";
  echo "<li>Année de naissance: $_POST[annee]</li>";
  echo "</ul>";
}
```

Noter que je n'ai pas besoin de concaténer, ni de simple quotes dans mes crochets autour de l'indice/name. C'est du au double quotes qui ouvrent mes chaînes de caractères.

#### \* Sauvegarder les données dans un fichier

Cela pourrait être intéressant si je n'ai pas de BDD mais que je souhaite néanmoins conserver les données pour une future newsletter destinée à mes utilisateurs

Je nomme ce fichier de conservation des données `recupDonnees.txt`

J'ajoute ce code à la suite de `recupForm.php`, toujours dans ma condition `if($_POST)`.

```
$fichier = fopen("recupDonnees.txt", "a");
fwrite($fichier, "Prénom: " . $_POST['prenom'] . "\n");
fwrite($fichier, "Description: " . $_POST['description'] . "\n");
fwrite($fichier, "Année de naissance: " . $_POST['annee'] . "\n");
fwrite($fichier, "-----\n");
fclose($fichier);
```

**Explications concernant le code:** la fonction prédéfinie `fopen` (pour folder open) permet d'ouvrir un fichier

Elle prend deux arguments. Le premier est le nom du fichier, ici `recupDonnees.txt`. Le second argument est le mode d'ouverture de ce fichier. Ici `a`

Le mode `a` va me permettre d'ouvrir le fichier s'il existe déjà, ou de le créer s'il n'existe pas encore.

Voici la documentation concernant `fopen()` et les autres modes existants

La variable `$fichier` que je crée représente le fichier ouvert (j'aurai pu la nommer `$f`, mais pour une meilleure clarté dans les explications ...). C'est la valeur que je lui affecte pour la rappeler ensuite et travailler dans ce fichier.

Pour les lignes 2, 3, 4 et 5, j'utilise la fonction `fwrite` pour écrire à l'intérieur en lui donnant comme premier argument le fichier ouvert ( `$fichier` ), puis le contenu à écrire.

La ligne 5 me servant de séparateur, utile pour la suite.

Enfin, en ligne 6, je ferme mon fichier avec `fclose` avec ma variable `$fichier` en argument encore une fois, car elle représente le fameux fichier ouvert

Ce code terminé, le formulaire mis en place est prêt à l'emploi. Non seulement les données s'afficheront sur la page d'accueil prévue à cet effet, mais elles seront stockées dans le fichier qui va apparaître à côté de `entrainement.php` et `recupForm.php`.

Désormais, si un nouvel utilisateur remplit et valide le formulaire. Le fichier `txt` sera simplement ouvert (car déjà créé) et les données du formulaire iront s'ajouter à celles déjà conservées au préalable, séparées les unes des autres par ma ligne 5 (le séparateur)

#### \* Récupérer et lire les données contenues dans un fichier.txt

Une fois les données stockées, je vais faire l'opération inverse, je vais les récupérer dans un fichier que je nomme `lecture.php`.

Voici le code ( dans `lecture.php` ), puis les explications

```
<?php

$nomFichier = "recupDonnees.txt";
$folder = file($nomFichier);

print "<pre>"; print_r($folder); print "</pre>";

foreach($folder as $ligne)
{
    echo $ligne."<br>";
}

echo implode( ' / ', $folder);
```

Je déclare une variable nommée `$nomFichier` à laquelle j'affecte le fichier que je veux explorer

Je déclare une nouvelle variable `$folder` (j'aurai pu l'appeler `$fichier` comme précédemment, mais je donne un nom différent pour éviter la confusion et la différencier de l'autre)

La fonction prédéfinie `file()` permet d'ouvrir un fichier et retourner son contenu sous forme de tableau. Je lui donne donc en argument `$nomFichier` (qui contient le fichier à explorer)

Noter que j'aurais pu ne pas coder la première ligne et écrire directement

```
$folder = file("recupDonnees.txt");
```

Mais, vous verrez fréquemment l'autre syntaxe, donc pour vous habituer....

En ligne 3 je fais un `print_r` pour vérifier ce que contient mon tableau

En ligne 4, ma boucle `foreach` pour récupérer ces données.

Cette fois, je veux afficher directement les valeurs contenues, sans afficher l'indice.

Je donne simplement l'`alias`, sans lui adjoindre la `=>` qui relie une valeur

PHP interprète cette syntaxe de manière à afficher directement chaque valeur qu'il rencontre

En ligne 8, si je veux, je peux afficher ces mêmes valeurs sur une même ligne, avec `implode()` avec `/` pour les séparer entre elles

Pour terminer ce sous-chapitre, après mon formulaire je mets un lien pour aller vérifier que mon fichier `lecture.php` récupère bien les données envoyées dans le `.txt` (avec un `target= _blank` pour l'ouvrir dans un nouvel onglet )

```
<button><a href="lecture.php" target="_blank">Vers fichier lecture</a></button>
```

# 16 Les Cookies

[Documentation officielle](#)

Un **cookie** est un fichier sauvegardé sur l'ordinateur de l'internaute avec des informations à l'intérieur. Ce seront des **informations non sensibles** (pas de mot de passe)

Il permet de stocker des informations sur les visiteurs, par exemple leur langue préférée (pour la navigation sur le site). Les sites les utilisent fréquemment à des fins publicitaires, pour engranger des informations sur les habitudes, les centres d'intérêt etc...

Pour cet exemple, nous allons créer un **cookie** qui recueille la langue usuelle de l'utilisateur

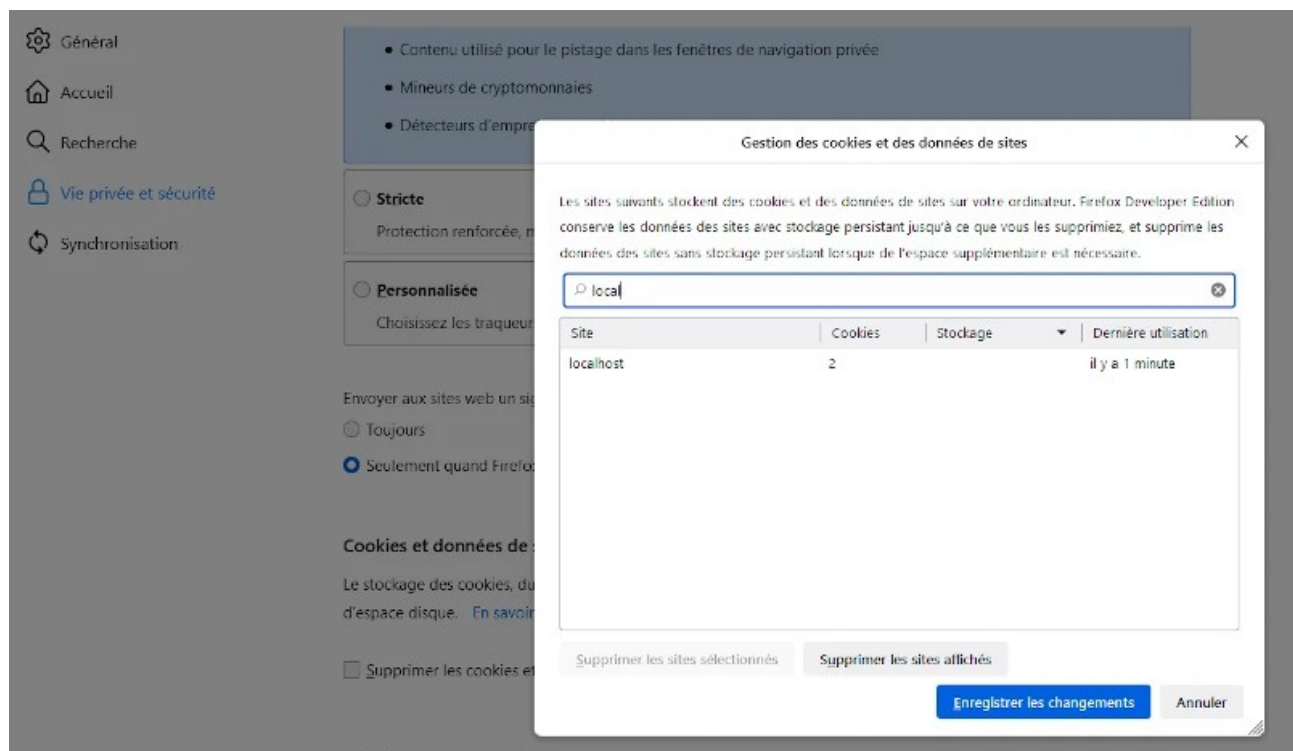
Une fois ce dernier créé, lorsque cet utilisateur reviendra sur le site, ces informations seront conservées (selon une durée fixée dans un paramètre du **cookie**)

Désormais, l'utilisateur retrouvera toutes ses préférences, visite après visite

## 16-1 Stockage des cookies

Pour visualiser les différents cookies stockés sur votre ordinateur, si vous utilisez **Firefox**, allez dans l'onglet **outil** de la barre des menus de FF, puis cliquez sur **paramètres**. Dans la barre de gauche, cliquez sur **Vie privée et sécurité**

Dans **Cookies et données de sites**, allez dans **Gérer les données**, puis tapez dans la barre de recherche localhost. Et ainsi apparaîtront les différents cookies créés pour le local





## 16-2 création d'un cookie

Pour donner le choix de la langue au visiteur, je vais utiliser de petits fanions, cliquables (test disponible à la fin de ce sous-chapitre)

Je mets en place tout d'abord ce bouton dans mon fichier d'entraînement

```
<button><a href="pageCookie.php" target="_blank">Vers fichier cookies</a></button>
```

Il va le relier au fichier `pageCookie.php` où je vais tout coder

Voici le code HTML

```
<div>
  <a href="?pays=en"></a>
  <a href="?pays=es"></a>
  <a href="?pays=dz"></a>
</div>
```

Je vais entourer ma balise `<img>` d'une balise `<a>` qui va servir à faire transiter des informations grâce à la méthode `GET`

Dans son attribut href, je vais faire transiter l'information dans l'URL. Ainsi, pour le drapeau de l'Angleterre, j'indique dans l'href que son `pays=en`, pour l'Espagne que `pays=es` etc...

Ainsi, si on clique sur le premier drapeau, pays sera égal à 'en', sur le second, pays sera égal à 'es' etc...

Pour récupérer cette information, nous allons utiliser la `condition switch/case` (mais nous aurions tout aussi bien pu le faire avec un `if/elseif`)

```
<?php
switch($pays){
  case 'fr' :
    echo "<h1>Bonjour</h1>";
    break;
  case 'en' :
    echo "<h1>Hello</h1>";
    break;
  case 'es' :
    echo "<h1>Hola</h1>";
    break;
  case 'dz' :
    echo "<h1>Salaam</h1>";
    break;
  default :
    echo "<h1>Vous devez choisir une langue</h1>";
    break;
}
```

Attention:

Mais au préalable, en début de script donc (au-dessus de la switch), nous allons devoir vérifier si nous avons bien reçu une information via l'URL

Cette vérification est obligatoire. Car lorsque l'utilisateur arrivera pour la première fois sur la page, il n'aura encore rien cliqué/choisi. Le `if(isset($_GET['pays']))` sera donc indispensable, sinon nous aurons droit à une erreur PHP ( `Undefined array key "pays"` )

Au lieu de `isset`, nous pourrions utiliser de manière tout aussi sûre et efficace `array_key_exists`

Voici le code pour cette vérification/condition ( à placer au-dessus de la `switch` ), puis les explications

```
if(isset($_GET['pays'])){
    $pays = $_GET['pays'];
}elseif(isset($_COOKIE['pays'])){
    $pays = $_COOKIE['pays'];
}else{
    $pays = 'fr';
}
```

Après avoir vérifié en ligne 1 que j'ai bien reçu une information dans mon URL, je déclare la variable `$pays` à laquelle j'affecte la valeur de l'information reçue dans l'URL (ligne 2)

En ligne 3, je soumetts une autre hypothèse ( `elseif` ). La [Superglobale \\$\\_COOKIE](#) me permet de vérifier si un `cookie` du même nom n'existe pas déjà. Auquel cas, pas besoin d'en créer un nouveau, `$_COOKIE` me permet de récupérer le contenu de celui existant

`$_COOKIE` vérifie si un cookie existe, et si oui, elle récupère son contenu

C'est la signification de la ligne 4. `elseif` le cookie `['pays']` existe déjà ? Alors ma variable `$pays` prendra sa valeur

Enfin, en lignes 5 et 6, le cas de figure où aucune information n'a été passée dans l'URL, si le cookie n'existe pas encore, alors `$pays` prendra la valeur `'fr'`, ce qui affichera le message 'Bonjour', par défaut

**Faire le test.**

Si tout fonctionne, le message sera Bienvenue, et si nous cliquons sur un drapeau, Bienvenue sera traduit dans la langue du pays du drapeau.

Par contre, si nous fermons cette fenêtre et que nous y retournons, l'information ne sera pas conservée. Nous n'avons pas encore créé le cookie

Il nous reste donc à le code, entre ma `vérification/condition` et la `switch`

Pour cela, nous allons avoir besoin de la fonction `setcookie()`. Elle fait partie des [fonctions réseaux](#), et prend 3 arguments

```
setcookie('pays', $pays, time()+365*24*3600);
```

Le premier, son nom (`'pays'`)

Le second, sa valeur, et cela pourra donc être la valeur par défaut, la valeur passée dans l'URL, ou la valeur récupérée dans le `cookie` existant

Enfin, la date à laquelle il doit expirer. Ici, je fais appel à `time()` qui me permet de récupérer le `timestamp`, exprimé en secondes

Cela représente le nombre de secondes écoulées entre le 01 janvier 1970 (date clé en informatique) et maintenant (le moment présent). Retenez que c'est un moyen de récupérer la date d'aujourd'hui, mais en secondes

Faire plusieurs tests avec `echo time()` dans ce fichier. Le nombre va évoluer.

Aussi, pour avoir un cookie qui expire dans un an, je vais additionner à `time()` : 365(pour les jours) multiplié à 24 (les heures), multiplié à 3600 (secondes dans une heure). Car, comme `time()` est indiqué en secondes, je dois lui ajouter une année, mais exprimée en secondes aussi.

Il faut savoir que `setcookie` permet non seulement de créer le cookie, mais aussi de le régénérer ! Le compteur repart à zéro à chaque fois que je l'appelle avec `$_COOKIE` (+ un an à chaque visite sur le site)

**Tester à nouveau.** Désormais les informations sont conservées.

Voici l'intégralité du code de ce fichier, car il y a un ordre à respecter pour le scripter

```
<div>
<a href="?pays=en"></a>
<a href="?pays=es"></a>
<a href="?pays=dz"></a>
</div>

<?php
if(isset($_GET['pays'])){
    $pays = $_GET['pays'];
}elseif(isset($_COOKIE['pays'])){
    $pays = $_COOKIE['pays'];
}else{
    $pays = 'fr';
}

setcookie('pays', $pays, time()+365*24*3600);

switch($pays){
    case 'fr' :
        echo "<h1>Bonjour</h1>";
        break;
    case 'en' :
        echo "<h1>Hello</h1>";
        break;
    case 'es' :
        echo "<h1>Hola</h1>";
        break;
```

```

case 'dz' :
    echo "<h1>Salaam</h1>";
    break;
default :
    echo "<h1>Vous devez choisir une langue</h1>";
    break;
}
?>

```

## 17 Les Sessions

### [Documentation officielle](#)

Les **sessions** servent sur un site à collecter des données, et les faire transiter de page en page, mémorisées. Si un *cookie* les conserve coté client (ordinateur), dans le cas de la **session** ça sera coté serveur (et en local, ce fichier sera visible dans **C:\xampp\tmp** ).

Ainsi, un utilisateur connecté sera reconnu sur toutes les pages du site, naviguant de l'une vers l'autre, sans perdre ses données ou ses droits (certaines pages étant visibles aux seuls connectés)

Dans le cadre d'un E Commerce, je créerai une autre **session** pour le panier. Cela permettra d'alimenter ce dernier, de le modifier (ajouter un article, modifier sa quantité, ou le supprimer), jusqu'à la procédure de paiement

#### \* Exemple avec la connexion d'un internaute (membre) :

Lorsqu'un internaute s'inscrit à 1 site, nous l'enregistrons dans une base de données.

Lorsqu'il se connecte (avec le bon pseudo et le bon mot de passe), nous gardons l'information (comme quoi l'internaute est actuellement connecté) en mémoire dans 1 fichier de session.

En effet, nous n'allons pas solliciter une éventuelle table de la base

"internaute\_actuellement\_connecte" car cela change souvent et les requêtes sql pour cette tâche alourdirait le site web

#### \* Exemple pour un panier :

Nous ferons le panier d'un site e commerce avec un système de session.

Nous n'enregistrerons pas les produits ajoutés au panier dans une base de données puisque la plupart du temps les paniers ne sont pas payés (les internautes ne finalisent pas toujours la commande).

Cela évitera de "polluer" la base de données pour rien. En revanche, (si le panier et le paiement sont validés) la commande sera enregistrée dans une base de données

#### \* Plus généralement, voici des cas d'utilisation :

- La connexion d'un membre à un site web.
- Le panier d'un site e commerce (vente en ligne).
- Les formulaires à plusieurs étapes.
- D'autres informations générales. Cela peut être marketing (derniers produits vus, suggestions de produits adaptés aux préférences de l'internaute selon sa navigation), etc.

Pour conclure, nous utiliserons les sessions pour sauvegarder des informations temporaires et garderons l'utilisation de base de données pour sauvegarder des informations durables.

## 17-1 Création de session

C'est avec la *fonction prédéfinie* `session_start()`, spécifique aux sessions, que j'en créerai ou rouvrirai une

Il en existe d'autres ( [Documentation officielle](#)) dont `session_unset` qui permettra de vider une session de toutes ou une partie de ses variables, sans détruire le fichier (ce que fera par contre `session_destroy`)

Pour que `session_start()` fonctionne, il faudra la déclarer en haut de page, avant le `DOCTYPE`. Si le `header` est positionné plus en haut, cela générera une erreur **PHP**

Le fichier, créé avec `session_start`, se localisera sur mon PC dans le dossier `C:\xampp\tmp` (comme déjà précisé en introduction)

C'est avec la **Superglobale** `$_SESSION` que je vais créer mes variables de session, stockées dans un tableau associatif

En premier lieu, nous allons à nouveau créer un lien/bouton dans notre fichier `entrainement.php`

```
<button><a href="pageSession.php" target="_blank">Vers fichier session</a></button>
```

Voici le script dans ce nouveau fichier `pageSession.php` pour "collecter" les diverses données d'un utilisateur

```
<?php
session_start();

$_SESSION['prénom'] = "Aziz";
$_SESSION['statut'] = "utilisateur" ;

echo '<button><a href="profil.php" target="_blank">Vers le Profil</a></button>';
```

Puis le code vers le fichier `profil.php` qui va récupérer les données

```
<?php
session_start();

if(isset($_SESSION)){
    echo "Bienvenue " . $_SESSION['prénom'] . " vous avez le statut d' " .
    $_SESSION['statut'] . " sur ce site.";
}
```

Cette nouvelle page devra aussi débuter par `session_start`, sinon, elle ne pourra récupérer les données

Notre script, pour fonctionner, devra vérifier si la session existe. Puis, nous récupérerons nos données avec la Superglobale `$_SESSION`, suivie de crochets [ ], les données étant stockées dans un tableau.

Il faut savoir qu'en même temps qu'un fichier session est créé dans le dossier `temp` de xampp, un cookie sera créé et stocké dans le navigateur (vérifier comme tout à l'heure).

Cela nous sera utile par la suite

Durant la construction de la simulation de E Boutique, nous verrons aussi ensemble comment vider une session de ses données ou totalement la supprimer.

## 18 PDO et requêtes SQL

[Documentation officielle](#)

PDO ( `PHP Data Object` ) est une couche d'abstraction intégrée à PHP depuis sa version 5.1

Elle permet de relier le langage PHP à la BDD ( `base de données` ), et ainsi l'exploiter, l'alimenter, pour les besoins du site

Dans un site web, il est utile de pouvoir manipuler des données.

Quelques exemples rapides :

Insérer des données :

côté FRONT : Lorsqu'un internaute s'inscrit sur votre site web,

côté BACK : Lorsque vous ajoutez un nouveau produit dans votre boutique.

Modifier des données :

côté FRONT : Lorsqu'un internaute modifie son profil sur votre site web,

côté BACK : Lorsque vous modifiez le prix d'un produit de votre boutique.

Supprimer des données :

côté FRONT : Lorsqu'un internaute supprime son compte de votre site web,

côté BACK : Lorsque vous supprimez un produit de votre boutique.

Consulter des données :

côté FRONT : Lorsqu'un internaute souhaite se connecter à votre site web,

côté BACK : Lorsque vous affichez les produits de votre boutique.

### 18-1 Connexion à la Base de données

Pour les besoins de ce chapitre, nous allons nous connecter à une base de données

Voici la syntaxe complète, puis les explications

```
$pdo = new PDO('mysql:host=localhost;dbname=boutique', 'root', '', array(
    PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING,
    PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8'
));
```

Je crée un objet `$pdo` de ma classe **PDO**, mais j'aurais pu tout aussi bien le nommer `$bdd`, par exemple. Son nom n'a pas d'importance, comme souvent pour une variable, tant qu'il est évocateur.

Je travaille pour l'instant en local, donc `host` (qui renseigne sur le serveur) sera bien égal à `localhost`.

Dans le `dbname`, je dois renseigner le nom de la bdd à laquelle je souhaite me connecter. Ici, c'est celle nommée `boutique`. Le champ suivant `'root'` renseigne le `dbuser`. En local il aura toujours cette valeur.

Et pour terminer, mon `mot de passe` restera vide, d'où `"` (quote sans aucune donnée, même pas un espace), tant que je suis en local.

`PDO::ATTR_ERRMODE` (pour attribut mode erreur) me sert à définir sous quel attribut **PDO** va m'envoyer ses messages d'erreurs. Avec cette syntaxe, je choisis une alerte `E_WARNING`.

Concernant `PDO::MYSQL_ATTR_INIT_COMMAND`, cela permet de sélectionner le type de codage de caractères qui sera automatiquement interprété à chaque connexion. Ici en `utf8`, comme dans mon DOCTYPE.

## 18-2 Les méthodes de PDO

PDO est une classe prédéfinie de PHP, qui possède ses propres `méthodes`/fonctions qui permettent de faire des requêtes SQL.

Voici, à titre indicatif, leur liste (la variable `$pdo` dans le code ci-dessous étant l'objet créé de la classe PDO, dans le sous-chapitre précédent):

```
echo "<pre>"; print_r(get_class_methods($pdo)); echo "</pre>";
```

## 18-3 La méthode Query()

La [méthode query\(\)](#) de **PDO** permet de faire différentes requêtes.

Ci-dessous je vais sélectionner tout ce qui concerne le membre prénommé Livia dans ma table `membre` de ma bdd `boutique` (et je fais un `var_dump` pour lire les informations collectées).

```
$afficheMembre = $pdo->query("SELECT * FROM membre WHERE prenom = 'Livia' ");
echo "<pre>"; var_dump($afficheMembre); echo "</pre>";
```

Cependant, il vaut mieux la réserver seulement à la requête `SELECT` et faire des `requêtes préparées` si je veux ajouter (`INSERT INTO`), modifier (requête de type `UPDATE`) ou supprimer (`DELETE`) des données dans ma BDD.

Cela permettra de sécuriser l'envoi de données (voir un prochain chapitre).

### Remarque:

Je fais un aparté sur la méthode `exec()` (listée dans le `print_r` du sous-chapitre 18.2), pour dire qu'elle permet de faire les `INSERT`, `UPDATE` et `DELETE` (comme `query()`). Par contre, le `SELECT` est réservé à `query()`. De plus, `exec()` ne retournera pas un résultat de données, mais un `integer`, qui indiquera le nombre des modifications impactées par ma requête

## 18-4 La méthode `fetch()`

Une fois exécuté la requête `query()`, je vais lui adjoindre la [méthode `fetch\(\)`](#), pour pouvoir parcourir et récupérer les données ciblées par mon `SELECT`

Je vais affecter toutes les valeurs récupérées par `fetch()` dans une variable nommée `$membre`

```
$membre = $afficheMembre->fetch(PDO::FETCH_ASSOC);
```

`fetch()` permet de faire une recherche sur le nom de la colonne (c'est souvent suffisant) et pour cela j'utiliserai `PDO::FETCH_ASSOC`. Je pourrai aussi faire une recherche combinée sur l'indice de la colonne + son nom, et dans ce cas, j'opterai pour `PDO::FETCH_BOTH`. Il existe aussi l'option pour cibler uniquement l'indice, c'est `FETCH_NUM`

Je fais un `print_r` pour visualiser les informations recueillies par `$membre`

```
echo "<pre>"; print_r($membre); echo "</pre>";
```

### Remarque

Dans `PDO::FETCH_ASSOC`, les `::` permettent de faire appel à une constante de la classe `PDO` (ici `FETCH_ASSOC`) sans devoir l'instancier (créer un objet)

Voici à présent la syntaxe pour récupérer les valeurs que je veux afficher concernant le membre prénommé Livia

```
echo "<p>Le membre qui à le pseudo $membre[pseudo] habite dans la ville de $membre[ville] dont le code postal est " . $membre['code_postal'] . "</p>";
```

Ci-dessus, j'ai volontairement concaténer la dernière valeur en la séparant de la précédente chaîne de caractères.

Pour mettre l'accent sur la syntaxe différente pour crocheter `code_postal`, qui a besoin de simple quotes. Alors qu'à l'intérieur des double quotes, je n'ai pas besoin des quotes dans les crochets (simple rappel)

Je pourrai aussi afficher tous les clients en base de données, sans restriction de prénom, sous forme de liste (je limite volontairement à 10 l'affichage des membres trouvés en BDD)

Et pour cela j'utiliserai la boucle `while`

```
$afficheMembre = $pdo->query("SELECT * FROM membre LIMIT 10");  
  
echo "<ul>";
```



```

while($membre = $afficheMembre->fetch(PDO::FETCH_ASSOC)){
    echo "<li>Le membre qui à le pseudo $membre[pseudo], habite dans la ville de
$membre[ville], dont le code postal est " . $membre['code_postal'] . "</li>";
}
echo "</ul>";

```

Pour un affichage dans un tableau, pour mon Back-Office par exemple, j'aurai alors besoin de combiner For, While et Foreach

Si ma requete Select reste similaire, ma syntaxe à l'intérieur des boucles va etre légèrement plus complexe

Je vais la détailler après l'affichage du code et du tableau (j'utilise volontairement des passages PHP contractés, car le code HTML va être conséquent)

```

<table>
<?php $afficheMembre = $pdo->query("SELECT * FROM membre LIMIT 10"); ?>
<thead>
<tr>
<?php for($i = 0; $i < $afficheMembre->columnCount(); $i++):
    $colonne = $afficheMembre->getColumnMeta($i) ?>
    <th><?= $colonne['name'] ?></th>
<?php endfor; ?>
</tr>
</thead>
<tbody>
<?php while($membre = $afficheMembre->fetch(PDO::FETCH_ASSOC)): ?>
<tr>
<?php foreach($membre as $key => $value): ?>
    <td><?= $value ?></td>
<?php endforeach; ?>
</tr>
<?php endwhile; ?>
</tbody>
</table>

```

Dans ma boucle For, rien de particulier concernant l'initialisation et l'incréméntation. Je vais par contre détailler la condition `$i < columnCount`

J'utilise la fonction prédéfinie `columnCount` (de la classe **PDOStatement**), qui va me permettre de stopper ma boucle For

Je vais ainsi générer autant de `<th>` que j'ai de colonnes dans ma Table en BDD

Ensuite, avec `getColumnMeta`, une autre fonction prédéfinie de la classe **PDOStatement**, je vais récupérer les noms de mes colonnes

-----

Le `print_r` de `$colonne` (volontairement limité à 2), ci dessous, montre que si je crochète à l'indice `$colonne['name']`, ce n'est pas de manière arbitraire

```
for($i = 0; $i < 2; $i++){
    $colonne = $afficheMembre->getColumnMeta($i);
    echo "<pre>"; print_r($colonne); echo "</pre>";
}
```

L'indice **name** est l'intitulé qui permet de récupérer le nom de chaque colonne

-----

Concernant la récupération des valeurs pour chaque client, dans les <td>, je vais d'abord faire appel à la **méthode fetch** (avec **PDO::FETCH\_ASSOC**), dans une boucle **While**

Suivie, après les <tr>, d'une boucle **Foreach** pour récupérer chaque **\$value** dans son <td>

### Cas particulier

Dans le cas où je dois récupérer une photo (pour un portrait, un article de presse, un produit etc...) ou un prix, j'utiliserai la **condition if()** dans ma boucle **Foreach**

Mon premier **if** concernera le cas où l'**indice/\$key** (nom de la colonne) sera égal a photo. Si cette condition est remplie, la **valeur/\$value** servira de nom de fichier (dans l'attribut src de ma balise img, précédée du chemin physique vers le dossier img)

Le **elseif** qui le suit ciblera l'**indice/\$key** intitulé prix. Pour pouvoir le faire suivre du sigle €

Enfin, le **else** servira pour tous les autres cas de figure, pour tous les autres indices/valeurs.

Voici un exemple de syntaxe pour ce cas de figurera

```
<?php while($produit = $afficheProduit->fetch(PDO::FETCH_ASSOC)): ?>
<?php foreach($produit as $key => $value): ?>
    <?php if($key == "photo"): ?>
        <td></td>
    <?php elseif($key == "prix"): ?>
        <td> <?= $value ?> €</td>
    <?php else: ?>
        <td> <?= $value ?> </td>
    <?php endif; ?>
<?php endforeach; ?>
<?php endwhile; ?> </tbody>
```

### Remarque

Il existe une autre méthode; **fetchAll**, qui m'aurait permis de faire l'économie de la boucle **while** dans le même code ci-dessus

Mais cela ne fonctionne qu'avec une base de données comportant un faible volume de données. Si ce n'est pas le cas, **fetch** sera plus approprié car elle récupère les données ligne par ligne, au lieu de renvoyer l'intégralité des données dans un tableau qui consommerait beaucoup de mémoire

## 19 Réalisation d'un espace de dialogue (cas concret)

Nous pouvons retrouver ce type de fonctionnalité sur Facebook, YouTube et de nombreux autres sites permettant à leurs internautes de dialoguer en direct

Quelles sont les différentes étapes afin de pouvoir créer cela ?

Tout d'abord, nous aurons besoin d'une base de données afin que les commentaires puissent être enregistrés dans une table.













Nous aurons également besoin d'une page web avec un formulaire afin de déposer des commentaires.

## 19-1 création d'une base de données

Voici les caractéristiques de la BDD que nous allons créer dans l'interface de PHPMyAdmin

Champ	Type	Taille	Spécificité
id_commentaire	INT	3	Clé primaire (PK - Primary Key), AUTO_INCREMENT (AI)
pseudo	VARCHAR	20	-
message	TEXT	-	-
date_enregistrement	DATETIME	-	-

Une fois entré ces paramètres, voici à quoi elle devra ressembler

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1 id_commentaire	int(3)			Non	Aucun(e)		AUTO_INCREMENT	 Modifier  Supprimer  Plus
<input type="checkbox"/>	2 pseudo	varchar(20)	utf8mb4_general_ci		Non	Aucun(e)			 Modifier  Supprimer  Plus
<input type="checkbox"/>	3 message	text	utf8mb4_general_ci		Non	Aucun(e)			 Modifier  Supprimer  Plus
<input type="checkbox"/>	4 date_enregistrement	datetime			Non	Aucun(e)			 Modifier  Supprimer  Plus

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

## 19-2 Le code

Nous allons mettre notre projet dans un dossier appelé /dialogue/ avec à l'intérieur un fichier nommé dialogue.php

En premier lieu, je me connecte à ma base de données

```
<?php

$pdo = new PDO('mysql:host=localhost; dbname=dialogue', 'root', '',
array(PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING, PDO::MYSQL_ATTR_INIT_COMMAND =>
'SET NAMES utf8'));
```

Je vais à présent mettre en place mon formulaire qui va permettre aux utilisateurs d'envoyer leurs messages

```

<form class="py-5" method="post">
  <div class="mb-3 col-3">
    <label for="pseudo" class="form-label">Votre pseudo</label>
    <input type="text" class="form-control" id="pseudo" name="pseudo"
placeholder="Pseudo">
  </div>
  <div class="mb-3 col-3">
    <label for="message" class="form-label">Votre message</label>
    <textarea class="form-control" id="message" name="message" rows="3"
placeholder="message"></textarea>
  </div>
  <button type="submit" class="btn btn-primary">Envoyer</button>
</form>

```

Dans ce formulaire, je ne prévois que deux champs. Celui pour écrire son pseudo et celui dédié au message.

Je ne vais pas proposer à l'utilisateur d'insérer un id pour le commentaire, cela ne se fait pas. Il sera auto-incrémenté en BDD.

Idem pour la date de son message. Ce n'est pas à lui de gérer cet aspect. Je vais utiliser une fonction prédéfinie prévue à cet effet.

Je prévois aussi un espace HTML pour visualiser les messages

```

<h2>Tous les messages envoyés</h2>
<div class="blockquote col-md-6 offset-md-1 p-5 text-justify shadow mt-5 bg-white
rounded">
  <h3 class="mb-5">Par: . Posté le :</h3>
  <p>Commentaire:</p>
  <p></p>
</div>

```

Cet aspect mis en place, je retourne en haut de mon fichier pour la suite du traitement PHP.

Première étape, j'initialise une variable `$erreur` sans lui donner de contenu, elle me sera utile dans le `if($_POST)` qui englobe la suite du code. Puis, je vérifie que mon champs pseudo va bien recevoir les informations que je veux et écarter le reste

```

$erreur='';
if($_POST){
  if(!isset($_POST['pseudo']) || !preg_match("#^[a-zA-Z0-9._-]{1,9}$#",
$_POST['pseudo']))){
    $erreur .= '<div class="alert alert-danger" role="alert">Erreur format pseudo !
</div>';
  }
}
}

```

J'utilise un `preg_match` pour encadrer le type de caractères que j'autorise.

Ici, les lettres majuscules et minuscules, les chiffres ainsi que qlq caractères spéciaux (. \_ et -). Le nombre de caractères pour le pseudo devra être compris entre 1 et 9.

Si un autre caractère est inséré, si leur nombre est supérieur à 9, ou si tout simplement aucune valeur n'est renseignée ( `isset` ) alors un message d'erreur sera généré pour le signaler

La vérification du champs message se fera différemment

```
if(!isset($_POST['message']) || iconv_strlen($_POST['message']) < 2 ||  
iconv_strlen($_POST['message']) > 200){  
    $erreur .= '<div class="alert alert-danger" role="alert">Erreur format  
message !</div>';  
}
```

Je vérifie encore qu'au préalable une donnée a été insérée dans le champs. Puis, avec `iconv_strlen` je contrôle la longueur de la chaîne de caractères. Pas moins de 2 et pas plus de 200. Sinon, message d'erreur généré.

Si mes deux champs ont été renseignés de la bonne manière, cela veut dire qu'aucun message d'erreur n'aura été généré, et donc que la variable `$erreur` n'aura reçu aucun contenu (c'est la signification du `empty($erreur)` ci-dessous, alors j'entamerai la procédure d'envoi de données en BDD

```
if(empty($erreur)){  
    $ajoutMessage = $pdo->prepare("INSERT INTO commentaire (pseudo, message,  
date_enregistrement) VALUES (:pseudo, :message, NOW())");  
}
```

Je vais le faire avec une requête préparée, plus sûre qu'avec un query

J'insère les valeurs reçues pour pseudo et message. La date d'enregistrement se fera quand à elle grâce à la fonction prédéfinie SQL `NOW()`

La requête préparée nécessite la syntaxe ci-dessus, avec les `:` dans la parenthèse de `Values`. Ce sont des `pointeurs nommés`, qui vont permettre de lier le champ en BDD avec la valeur reçue dans le formulaire (récupérée grâce à `$_POST` dans les `bindValue` ci-dessous)

```
$ajoutMessage->bindValue(':pseudo', $_POST['pseudo'], PDO::PARAM_STR);  
$ajoutMessage->bindValue(':message', $_POST['message'], PDO::PARAM_STR);  
$ajoutMessage->execute();
```

Les `bindValue` entrent aussi dans le cadre de la requête préparée et sécurisent l'envoi des données en BDD.

Je termine ma requête préparée obligatoirement en l'exécutant.

Dans la partie HTML de mon code, à l'intérieur de `Body`, j'affiche mon message d'erreur (s'il y'en a un) pour avertir l'internaute qui remplit le formulaire

```
<body>  
<?= $erreur ?>
```

Je vais à présent scripter la requête qui va me permettre de récupérer les valeurs insérées en BDD

```
$requeteCommentaires = $pdo->query("SELECT * FROM commentaire");
```

Dans mon code HTML, je mets en place une boucle while pour récupérer tous les commentaires de la BDD

```
<?php while($commentaire = $requeteCommentaires->fetch(PDO::FETCH_ASSOC)): ?>
<div class="blockquote col-md-6 offset-md-1 p-5 text-justify shadow mt-5 bg-white
rounded">
  <h3 class="mb-5">Par: <?= $commentaire['pseudo'] ?>. Posté le <?=
$commentaire['date_enregistrement'] ?></h3>
  <p>Commentaire:</p>
  <p><?= $commentaire['message'] ?></p>
</div>
<?php endwhile ?>
```

Je vais à présent améliorer l’affichage.

Au lieu d’avoir comme précédemment un intitulé global

```
<h2>Tous les messages envoyés</h2>
```

Il sera intéressant d’afficher le nombre de messages total, grâce à la fonction prédéfinie `rowCount()`

```
<h2><?= $requeteCommentaires->rowCount() ?> messages envoyés</h2>
```

Il sera aussi intéressant d’afficher en premier les messages les plus récents

Pour cela, je vais compléter ma requête d’origine avec un `ORDER BY` ( `DESC` pour inverser l’ordre)

```
$requeteCommentaires = $pdo->query("SELECT * FROM commentaire ORDER BY
date_enregistrement DESC");
```

Enfin, je vais formater l’affichage de la date d’enregistrement pour la rendre plus lisible. Je vais modifier en conséquent ma requête d’origine

```
$requeteCommentaires = $pdo->query("SELECT pseudo, message,
DATE_FORMAT(date_enregistrement, '%d/%m/%Y') AS
datefr,DATE_FORMAT(date_enregistrement, '%H:%i:%s') AS heurefr FROM commentaire
ORDER BY date_enregistrement DESC");
```

Le \* (all) disparaît au profit du nom des indices. Cela me permet de formater `date_enregistrement` tel que je veux pour l’affichage de la date (auquel je donne un alias `datefr`) ainsi que de l’heure (qui aura aussi son alias)

Alias que je vais utiliser dans le code HTML

```
<h3 class="mb-5">Par: <?= $commentaire['pseudo'] ?>. Posté le <?=
$commentaire['datefr'] ?> à <?= $commentaire['heurefr'] ?></h3>
```

## 20 Approche sécurité

[Documentation officielle](#)

### Remarque importante

Dans le chapitre précédent, j'ai détaillé la procédure pour envoyer et récupérer les données d'un formulaire. Mais tout transit de données (par `$_POST` comme `$_GET`) devra être protégé. De l'erreur maladroite, mais surtout de l'acte de malveillance qui aura pour but de nuire à l'intégrité du site !

**C'est un aspect très important, qui engage la responsabilité du développeur**

### 20-1 Vérification des données d'un formulaire

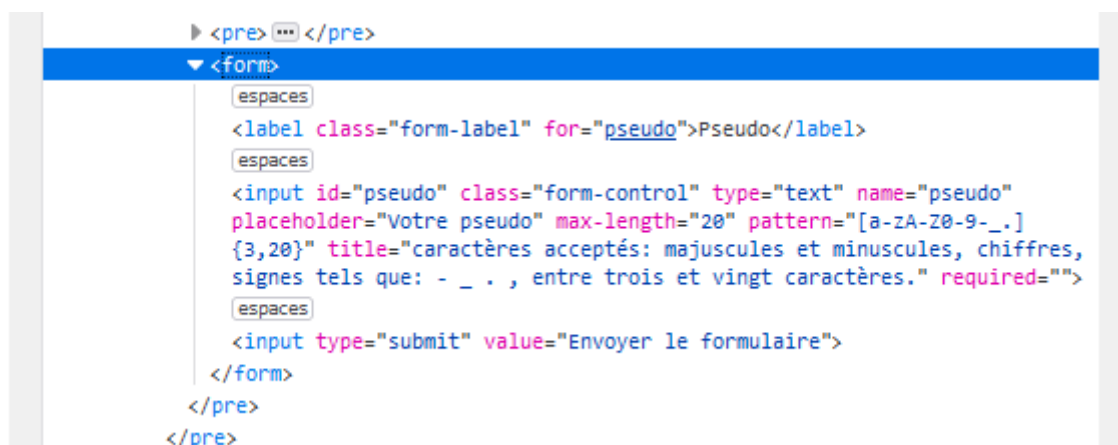
Dans le cadre de la **méthode POST**, il s'agira de vérifier que le user n'envoie pas, par exemple à la place d'une adresse email valide, un prénom, des chiffres etc...

Illustration avec le code HTML ci-dessous ( dans `entrainement.php` ), reprenant la phase d'inscription d'un utilisateur (au moment de renseigner son pseudo)

```
<form>
  <label class="form-label" for="pseudo">Pseudo</label>
  <input class="form-control" type="text" name="pseudo" id="pseudo"
placeholder="Votre pseudo"
  max-length="20" pattern="[a-zA-Z0-9-_.]{3,20}" title="caractères acceptés:
  majuscules et minuscules, chiffres, signes tels que: - _ . , entre trois et vingt
caractères."
  required>
  <input type="submit" value="Envoyer le formulaire">
</form>
```

Si je tente de sécuriser l'envoi des données avec un `max-length` et un `pattern` tel qu'au-dessus, cela ne sera suffisant que pour un utilisateur lambda.

Pour un utilisateur malintentionné, il lui suffira de faire F12, et d'aller les supprimer dans le code affiché dans l'inspecteur.



Désormais ce barrage est effacé du code et le hacker pourra envoyer ce que bon lui semble, de farfelu comme dangereux en BDD

Toutes les contraintes faites coté **Front/navigateur** seront facilement contournables. Il faudra donc coder ces mêmes contraintes dans le **script PHP**

Voici a quoi ressemblerait la vérification pour le pseudo. La syntaxe que j'ai choisi étant une possibilité parmi d'autres, pour "cadrer" un pseudo

```
if($_POST){  
    if(!isset($_POST['pseudo']) || !preg_match('#^[a-zA-Z0-9-_.]{3,10}$#',  
$_POST['pseudo'])){  
        $erreur .= '<div class="alert alert-danger" role="alert">Erreur format pseudo !  
</div>';  
    }  
}
```

Avant tout, en ligne 1 je commence par vérifier si, globalement, j'ai bien reçu des données avec la méthode POST

Si j'ai plusieurs formulaires dans un même fichier, il faudra donner un nom a chacun d'eux, dans le **name** du bouton de validation. La syntaxe de la vérification globale deviendra **if(\$\_POST[nameDuButton]){}**

Et ce n'est qu'ensuite que je commencerai à "bordurer" une par une, mes différentes contraintes

Notez que je vais vérifier les cas où la condition n'est pas remplie, plutôt que de lister tous les cas où la condition est vérifiée

Il est plus sur et rapide de référencer les cas qui ne rentrent pas dans le cadre voulu, plutôt que d'essayer de valider différentes configurations, difficiles a toutes recenser et imaginer

Ainsi, en premier, et cela sera le cas pour tous les champs suivants de mon formulaire, je vérifie que j'ai bien reçu une information dans mon champs **['pseudo']**. Je vérifie avec un **!isset**, qui signifie "n'existe pas" ( **!/not** étant une négation). Si aucune donnée n'y a été insérée, alors je ne validerai pas le formulaire (si ma base de données exige que ce champs soit fourni)

En second lieu, je focalise sur la contrainte liée au champs **['pseudo']** a proprement parlé. Je la débute par un **!preg\_match**. Ce dernier me permet de poser un cadre strict sur les caractères demandés, leur nombre etc... Si cela ne correspond pas au format que j'exige (ici, des lettres minuscules et majuscules, des chiffres, les signes - \_ . , de trois caractères minimum et max dix), alors le formulaire ne sera pas validé. De plus, je génère un message à l'utilisateur pour lui signifier qu'il y a une erreur sur ce champs (message qui pourra etre plus précis que celui du dessus, très succinct)

Je n'utiliserai pas tout le temps **preg\_match**. Pour un nom, un prénom, une adresse, je passerai plutôt par le contrôle de la longueur de la chaine de caractères avec **iconv\_strlen** . D'autres développeurs utiliseront pour ce même contrôle **strlen** (voir le [chapitre 8 sur les fonctions prédéfinies](#)), les deux options sont possibles et cohérentes



En revanche, pour le code postal, je ferai à nouveau appel à `preg_match`, sachant que ce dernier, en France, devra obligatoirement être fourni sous la forme de cinq chiffres

```
if(!isset($_POST['code_postal']) || !preg_match('#^[0-9]{5}$#',
$_POST['code_postal'])){
    $erreur .= '<div class="alert alert-danger" role="alert">Erreur format code
postal !</div>';
}
```

D'autres préféreront cette syntaxe, avec `ctype_digit` (plutôt que `is_numeric`, peut-être moins adapté). Ainsi, je vérifie que je reçois bien un nombre entier, dont la longueur n'est pas différente de 5

```
if(!isset($_POST['code_postal']) || !ctype_digit($_POST['code_postal']) ||
iconv_strlen($_POST['code_postal']) !== 5){
    $erreur .= '<div class="alert alert-danger" role="alert">Erreur format code
postal !</div>';
}
```

Voici un dernier exemple, car différent, pour contrôler que j'ai bien reçu un format valide pour une adresse email

```
if(!isset($_POST['email']) || !filter_var($_POST['email'], FILTER_VALIDATE_EMAIL)){
    $erreur .= '<div class="alert alert-danger" role="alert">Erreur format email
!</div>';
}
```

A nouveau, en tout premier, je vérifie que j'ai bien reçu une donnée dans le champ email, puis je contrôle son contenu. Cette fois, je vais utiliser `filter_var` en lui appliquant le format prévu par `FILTER_VALIDATE_EMAIL`, et cela sera suffisant pour que tout autre format qu'une adresse email soit refusé

Je précise que tous ces contrôles devront être inclus dans ma condition de départ `if($_POST)`

Pour conclure, une fois vérifié que toutes mes conditions sont remplies. Qu'à aucun moment ma variable `$erreur` n'a reçu de contenu. Que `if(empty($erreur))`, alors je pourrai commencer ma procédure de validation de formulaire, tout en continuant à sécuriser l'envoi de données, cette fois avec des `requêtes préparées`

## 20-2 Les requêtes préparées

[Documentation officielle](#)

Faire une **requête préparée** à un double avantage

Elle ne sera préparée, analysée qu'une seule fois, mais exécutable autant de fois que l'on souhaite. Elle permet ainsi un gain de ressources et de vitesse d'exécution

Mais, ce qui va m'intéresser dans ce sous-chapitre, c'est de sécuriser mon site en empêchant les [injections SQL](#)

Une **injection SQL** se fait, par exemple, par le biais d'un formulaire d'inscription. Elle aura pour but de modifier la BDD (supprimer intégralement une table, donner des droits d'admin a un nouvel utilisateur etc...)

Faire une **requete préparée** est un des moyens pour les quasi annihiler

Voici deux exemples de syntaxes, la première, **préparée**, la seconde, avec un **query**, plus rapide, simple mais perméable.

Je vais insérer deux données, une de type **string** et l'autre de type **integer**

```
if(empty($erreur)){
    $inscrireUser = $pdo->prepare("INSERT INTO membre (pseudo, code_postal) VALUES (:pseudo, :code_postal)");
    $inscrireUser->bindValue(':pseudo', $_POST['pseudo'], PDO::PARAM_STR);
    $inscrireUser->bindValue(':code_postal', $_POST['code_postal'], PDO::PARAM_INT);
    $inscrireUser->execute();
}
```

La même, avec **query**

```
if(empty($erreur)){
    $inscrireUser = $pdo->query("INSERT INTO membre (pseudo, code_postal) VALUES ($_POST[pseudo], $_POST[code_postal])");
}
```

Dans les deux cas, je commence par vérifier que ma variable **\$erreur** n'a pas reçu de contenu ( **if(empty(\$erreur))** )

Ensuite, pour ma **requete préparée**, je vais faire appel à ce que l'on appelle un **marqueur nommé** : pour ajouter les différents paramètres/données ( **:pseudo** et **:code\_postal** )

A la ligne suivante, dans mon **bindValue**, je vais faire correspondre pour chaque marqueur, sa valeur reçue dans le formulaire. Je vais enfin indiquer avec **PDO::PARAM** le type que je veux/dois recevoir ( **\_STR** pour une chaîne de caractères, **\_INT** pour un entier )

**bindValue** permet d'associer une valeur à un paramètre. Il n'est pas obligatoire. D'autres lui préféreront **bindParam**. C'est un choix, et les deux syntaxes sont assez ressemblantes

Une fois terminé tous mes **bindValue**, il ne me reste qu'à **execute()** ma requête.

## 20-3 Exemple d'injection SQL

Pour illustrer ce cas précis, je vais créer un nouveau dossier **/securite/** avec à l'intérieur un fichier nommé **connexion.php**

Je commence par créer cette base de données (avec un **id\_membre** auto-incrémenté et un **pseudo** «unique», deux membres ne pouvant avoir le même)

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
<input type="checkbox"/>	1 id_membre	int(3)			Non	Aucun(e)		AUTO_INCREMENT
<input checked="" type="checkbox"/>	2 pseudo	varchar(20)	utf8mb4_general_ci		Non	Aucun(e)		
<input type="checkbox"/>	3 mdp	varchar(20)	utf8mb4_general_ci		Non	Aucun(e)		
<input type="checkbox"/>	4 nom	varchar(20)	utf8mb4_general_ci		Non	Aucun(e)		
<input type="checkbox"/>	5 prenom	varchar(20)	utf8mb4_general_ci		Non	Aucun(e)		
<input type="checkbox"/>	6 email	varchar(20)	utf8mb4_general_ci		Non	Aucun(e)		

Avec la sélection :

---

Index

Action	Nom de l'index	Type	Unique	Compressé	Colonne	Cardinalité	Interclassement
<input type="button" value="Éditer"/> <input type="button" value="Renommer"/> <input type="button" value="Supprimer"/>	PRIMARY	BTREE	Oui	Non	id_membre	0	A
<input type="button" value="Éditer"/> <input type="button" value="Renommer"/> <input type="button" value="Supprimer"/>	pseudo	BTREE	Oui	Non	pseudo	0	A

J'ajoute directement en BDD qlq membres

Options

	id_membre	pseudo	mdp	nom	prenom	email
<input type="checkbox"/> <input type="button" value="Éditer"/> <input type="button" value="Copier"/> <input type="button" value="Supprimer"/>	1	Iron Man	hommeDeFer	Stark	Tony	ironman@gmail.com
<input type="checkbox"/> <input type="button" value="Éditer"/> <input type="button" value="Copier"/> <input type="button" value="Supprimer"/>	2	Black Widow	veuveNoire	Romanoff	Natasha	blackwidow@gmail.com

Je vais a présent me connecter à cette base de données ainsi que mettre en place mon formulaire de connexion dans le fichier `connexion.php`

```
$pdo = new PDO('mysql:host=localhost;dbname=securite', 'root', '',
array(PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING, PDO::MYSQL_ATTR_INIT_COMMAND =>
'SET NAMES utf8'));
```

```
<h1 class="my-5 text-center">Espace Connexion</h1>
<form class="py-5" method="post">
  <div class="mb-3 col-3">
    <label for="pseudo" class="form-label">Pseudo</label>
    <input type="text" class="form-control" id="pseudo" name="pseudo"
placeholder="Votre pseudo">
  </div>
  <div class="mb-3 col-3">
    <label for="mdp" class="form-label">Mot de passe</label>
    <input type="text" class="form-control" id="mdp" name="mdp" placeholder="Votre
mot de passe">
  </div>
```

```
<button type="submit" class="btn btn-primary">Connexion</button>
</form>
```

Volontairement, l'attribut `type` de l'input mdp est = à `text` et non `password` (cela sera plus pratique pour la démonstration)

Ceci mis en place, je vais à présent faire un traitement PHP me permettant de vérifier si la personne qui vient de vouloir se connecter existe bien en BDD

Pour cela, je vais d'abord faire une requête SELECT

```
if($_POST){
    $rechercheMembre = $pdo->query("SELECT * FROM membre WHERE pseudo = 
    '$_POST[pseudo]' AND mdp = '$_POST[mdp]' ");
}
```

Puis, toujours dans le `if($_POST)`, un `fetch()` pour récupérer les informations

```
if($_POST){
    $rechercheMembre = $pdo->query("SELECT * FROM membre WHERE pseudo = $_POST[pseudo]
    AND mdp = $_POST[mdp] ");

    $membre = $rechercheMembre->fetch(PDO::FETCH_ASSOC);
}
```

Une fois ceci mis en place, je vais avertir l'utilisateur s'il est reconnu en tant qu'inscrit ou non

Voici le code (dans un `if($_POST)`), puis les explications

```
<h2 class="my-5 text-primary">Vos informations de connexion</h2>
<?php if($_POST): ?>
    <?php if(!empty($membre)): ?>
        <?= print_r($rechercheMembre) ?>
        <h3 class="text-success">Félicitations, vous êtes connecté</h3>
        <p>Vous êtes: <?= $membre['pseudo'] ?></p>
        <p>Votre email est: <?= $membre['email'] ?></p>
    <?php else: ?>
        <h3 class="text-danger">Votre connexion a échoué</h3>
    <?php endif ?>
<?php endif ?>
```

Si `$membre` a reçu un contenu, c'est à dire si j'ai retrouvé en bdd un membre dont le pseudo **ET** le mdp sont similaires à ceux renseignés dans le formulaire, alors je lui adresse un message de félicitations et je récupère par la même occasion diverses informations le concernant pour les lui afficher

Dans le cas contraire, si `empty($membre)`, c'est à dire qu'aucune information ne correspond, ou seulement l'une des deux, c'est que ce n'est pas le bon utilisateur.

Je lui envoie donc un message d'erreur

Cela semble bien fonctionner, sauf qu'avec le cas de figure qui va suivre, on va voir que tout n'est pas si au point que cela

### exemple d'attaque par injection SQL

Dans le champs pseudo, écrivez le nom du pseudo de l'inscrit, suivi d'un ' ainsi que d'un #

Pseudo

Iron Man'#

N'écrivez rien dans le champs mdp, puis validez avec le bouton

Cela fonctionne, avec pourtant un pseudo erroné, inexistant en BDD, et sans avoir eu à renseigner le mdp

Explications:

Je fais un `print_r($rechercheMembre)`, pour voir la différence entre la requête exécutée sans l'attaque sql, et dans le cas de l'attaque sql (je le positionne dans le `!empty()` )

```
<?php if(!empty($membre)): ?>
<?= print_r($rechercheMembre) ?>
<h3 class="text-success">Félicitations, vous etes connecté</h3>
```

Dans le cas d'une connexion normale

```
SELECT * FROM membre WHERE pseudo = 'Iron Man' AND mdp = 'hommeDeFer'
```

Dans le cas d'une injection SQL

```
SELECT * FROM membre WHERE pseudo = 'Iron Man'#' AND mdp = "
```

L'apostrophe permet de stopper le bon quote, et le # qui le suit immédiatement permet de mettre toute la suite en commentaire (donc de comparer aussi le mdp pour pouvoir se connecter

Si je ne connais pas le pseudo d'un inscrit (que je peux aisément voir sur un forum) je pourrai faire une injection SQL au niveau du mdp (écrire ceci dans le champs mdp et rien dans pseudo)

Mot de passe

' OR id\_membre = '1

Résultat du `print_r()`

```
SELECT * FROM membre WHERE pseudo = " AND mdp = " OR id_membre = '1'
```

L'apostrophe me permet de fermer le champs mdp et d'introduire le reste de la syntaxe.

Désormais ma requête me permet d'ignorer le fait qu'aucun membre n'a de pseudo vide ou mdp vide, et de passer au cas où un membre aurait un id=1, probablement vrai, j'accède donc ainsi à toutes les infos le concernant, mail, mdp, adresse, etc....tout ce qui aura pu être renseigné lors de l'inscription

Dans le même ordre d'idées, en pire, le hacker pourrait aller encore plus loin en modifiant des données stockées, détruire la bdd etc...

### Contrer cette injection SQL

Un des moyens pour contrer cette injection SQL serait d'utiliser la fonction prédéfinie `htmlspecialchars()`

Il me faudrait coder ceci dans mon traitement PHP avant la requête

```
$_POST['pseudo'] = htmlspecialchars($_POST['pseudo'], ENT_QUOTES);  
$_POST['mdp'] = htmlspecialchars($_POST['mdp'], ENT_QUOTES);
```

`htmlspecialchars()` n'est pas la seule à pouvoir neutraliser les caractères spéciaux. Il en existe d'autres, comme on va le voir dans le sous-chapitre suivant

## 20-4 Failles XSS

En dernier lieu, et c'est peut-être ce qu'il y a de plus simple à faire (au sens rapide), il faudra se prémunir des **failles XSS**. Il s'agira pour la personne malveillante, comme pour les injections SQL, d'envoyer du code via un formulaire ou l'URL, pour affecter directement le site (et non plus la BDD)

Pour cela, j'ai à ma disposition deux [fonctions sur les chaînes de caractères](#). Il s'agit de `htmlspecialchars` et de `strip_tags`, même si d'autres existent pour cela

Je remets, pour la démonstration, le formulaire du [chapitre sur la méthode POST](#)

Si je décide de mettre ce simple bout de code **CSS** dans l'input destiné à recevoir le pseudo ou la description

```
<style>  
body{  
    display:none;  
}  
</style>
```

Faites le test. Vous n'aurez ensuite qu'à fermer l'onglet de cette page, puis recharger le cours. Tout le contenu sera à nouveau disponible

La page entière va disparaître. En fait, elle sera juste invisible, du fait du `display:none` (vérifiez en affichant le code source de la page). Tout est encore là, mais inexploitable désormais.

Tout reviendra comme avant, mais si ce formulaire avait été validé pour aller alimenter la BDD, les dégâts auraient été plus long à réparer

En premier lieu, cela aurait affecté tous les utilisateurs. Puis il aurait fallu aller nettoyer la base de données pour retrouver la situation initiale. Et ce, avec juste un bout de code que je vais qualifier de "gentil".

Je reprends le même formulaire, que je vais protéger cette fois

```
<form method="POST" action="">  
  
    <label for="prenom">Prénom</label>
```

```

<input type="text" name="prenom" id="prenom" placeholder="Votre prénom">

<label for="description">Description</label>
<textarea name="description" id="description" rows="3"
placeholder="Description">
</textarea>

<label for="annee">Année de naissance</label>
<select name="annee" id="annee">
<?php
    for($i = date('Y'); $i >= date('Y') - 100; $i--){
        echo '<option>' . $i . '</option>';
    }
?>
</select>
<br>
<input type="submit" name="bouton3" value ="Soumettre">
</form>

<?php if(isset($_POST['bouton3'])): ?>
<ul>
<li>Prénom: <?= htmlspecialchars($_POST['prenom']) ?></li>
<li>Description: <?= htmlspecialchars($_POST['description']) ?></li>
<li>Année de naissance: <?= $_POST['annee'] ?></li>
</ul>
<?php endif; ?>

```

## 20-5 Les redirections

Pour terminer ce chapitre, je vais aborder un aspect légèrement différent, mais toujours en relation avec la sécurité d'un site

Il ne s'agira plus de protéger le transit de données, mais de gérer les autorisations des visiteurs et utilisateurs sur les différentes pages du site.

De manière basique, je vais devoir protéger le back-office de toute personne qui n'aura aucune habilitation à y faire quoique ce soit. Seul un administrateur, ou par extension l'équipe administrative, pourra se rendre sur ces pages "sensibles"

Un des moyens que j'ai à ma disposition, c'est la redirection de toute personne non-autorisée, vers une page moins sensible du site

Pour cela, je vais faire appel à la **fonction réseaux : header()** , avec l'entête (location:) , suivi de l'URL de la page où je veux l'envoyer

Voici par exemple un script pour "renvoyer" toute personne non-autorisée d'une des pages du back-office

```

if(!internauteConnecteAdmin()){
    header('location:../connexion.php');
    exit();
}

```

}

Ma *condition* de départ sera : si internaute connecté n'est pas admin (avec le "not" ! ).C'est une fonction d'utilisateur que j'ai codé au préalable, et ses droits seront plus restrictifs que `internauteConnecte()`, *fonction* pour des utilisateurs basiques

Donc, si l'internaute ne possède pas des droits d'admin, alors il sera rebasculé automatiquement vers la page de connexion

Je ne le renvoie pas vers sa page de profil, car en fait je ne sais même pas si c'est quelqu'un qui est connecté. Il est malheureusement probable que ce soit un individu mal-intentionné, qui entre une URL hypothétique dans la barre d'adresse. Je l'envoie vers la page de connexion du site, et s'il était déjà connecté, une autre redirection le rebasculera automatiquement vers son profil

Il faut veiller à protéger chaque page selon le statut de l'internaute

Dans le même ordre d'idées, s'il est déjà connecté, alors je lui interdirai d'aller sur les pages d'inscription ou de connexion. Il n'a rien à y faire, il est déjà connecté !

Inversement, la page profil sera "interdite" pour quelqu'un de non-connecté

Dernier point concernant les redirections : je peux en faire une pour une inscription réussie, vers la page connexion. Puis une autre vers la page profil pour une connexion réussie, avec à chaque fois un message pour en informer le user

Dans ce cas là, il ne sera pas question de sécurité, mais la volonté d'améliorer l'expérience utilisateur (UX), de lui rendre sa navigation agréable et plus facile.