

## **UE : Bases de données et développement back-end**

- **AA : Bases de données et développement back-end – Travaux pratiques**  
Antoine Malaise & Ivan Miller

Bachelier en Informatique  
orientation réseaux et télécommunications



# Index

Index.....	1
Chapitre 1 : PHP.....	4
1.1 Introduction au PHP .....	4
A. Histoire du PHP .....	4
B. Intérêts du PHP .....	5
C. Outils nécessaires.....	5
D. Pré-interprétation .....	6
E. Versions PHP.....	6
Exercice 1 : Hello World .....	7
1.2 Inclusions et redirections .....	8
Exercice 2 : Inclusions de fichiers .....	9
1.3 Variables.....	10
A. Définition.....	10
B. Noms de variables .....	10
C. Affectations, déclarations et destructions .....	11
D. Tester l'existence d'une variable .....	12
E. Affectations par référence .....	13
F. Constantes .....	13
G. Types de variables.....	14
H. Chaînes de caractères .....	16
I. <i>echo</i> .....	19
J. Concaténations .....	20
K. Variables dynamiques .....	20
L. Opérateurs de base .....	21
Exercice 3 : Opérations sur les variables.....	22
M. Tableaux.....	23
N. Tableaux indicés.....	24
O. Tableaux associatifs .....	24
P. Fonctions sur les tableaux.....	25
1.4 Dates.....	27
A. Préciser le fuseau horaire .....	27
B. <i>getdate()</i> .....	27
C. <i>date()</i> .....	28
D. Les <i>timestamp</i> .....	29
Exercice 4 : Utilisation de la date .....	30
1.5 Structures conditionnelles et itératives .....	31
A. <i>if</i> .....	31
B. <i>switch</i> .....	32
C. <i>while</i> .....	33
D. <i>do while</i> .....	33
E. <i>for</i> .....	34
F. <i>foreach</i> .....	34
Exercice 5 : Boucles .....	37
Exercice 6 : Générateur de noms de nains.....	38
1.6 Variables prédéfinies.....	39
A. Variables prédéfinies.....	39

B. GET et POST .....	40
Exercice 7 : Paramètres GET .....	41
C. Sessions .....	42
D. Cookies .....	43
E. Cookies et RGPD .....	44
1.7 Fonctions .....	45
A. Déclarer et utiliser des fonctions .....	45
B. Quelques fonctions mathématiques .....	46
1.8 Traitements des formulaires .....	47
A. Paramétrer les balises <i>form</i> .....	47
B. Récupérer les données du formulaire .....	47
C. Tester l'existence des données du formulaire .....	48
D. Exemple pratique .....	49
Exercice 8 : Formulaire de connexion .....	50
E. Réafficher les données du formulaire .....	51
F. Envoi de fichier .....	52
1.9 Sécurité .....	53
A. Introduction .....	53
B. Les attaques par injection .....	53
C. Les tests .....	55
D. À propos du hachage .....	56
Exercice 9 : Formulaire avec gestion des erreurs du visiteur .....	57
1.10 Regex (expressions régulières) .....	59
A. Améliorer la sécurité .....	59
B. Utiliser des motifs avec <i>preg_match()</i> .....	59
C. Types de motifs : POSIX ou PCRE ? .....	60
D. Syntaxe des motifs POSIX .....	60
E. Exercice : traduisez les 3 regex suivants en français .....	60
1.11 Fichiers plats .....	61
Exercice 10 : Formulaire avec fichiers plats .....	62
1.12 POO : Programmation orientée objet .....	63
A. Introduction POO .....	63
B. Classes et objets .....	63
C. Héritage .....	64
D. <i>stdClass</i> .....	64
E. Sérialisation .....	65
F. Espaces de noms ( <i>namespace</i> ) .....	66
1.13 PHP et MySQL : exemples .....	68
A. Principales commandes PDO .....	68
B. Se connecter à la base de données .....	69
C. Charger les noms des catégories depuis la BD .....	69
D. Charger un article précis .....	70
E. Exécuter une requête en boucle .....	71
F. Fonctions SQL : nombre, moyenne, somme, min et max .....	71
G. GROUP BY .....	71
H. UPDATE .....	72
I. INSERT .....	72
J. JOIN .....	73
Exercice 12 : Walking Pets .....	74

1.14 Coder comme des professionnels : PSR et MVC .....	78
A. Coder comme des professionnels.....	78
B. Les PSR.....	78
C. L'architecture Modèle-Vue-Contrôleur .....	79
D. Exemple MVC.....	80
Récapitulatifs HTML et CSS .....	82
Récapitulatif HTML.....	82
Récapitulatif des sélecteurs CSS.....	84
Sources .....	85



Le syllabus est parsemé d'exercices papier et d'exercices pratiques. Les fichiers nécessaires aux exercices pratiques sont téléchargeables sur l'eCampus (clé d'inscription : « Firefox »).



La présence de ce pictogramme indique qu'il s'agit d'informations complémentaires, pour les plus téméraires d'entre vous. Les chances d'être interrogé sur cette matière sont très faibles, mais vous pourriez en avoir besoin lors d'exercices ou de projets personnels.

Ivan Miller  
ivan.miller@heh.be



# Chapitre 1 : PHP

## 1.1 Introduction au PHP

### A. Histoire du PHP

En 1994, Rasmus Lerdorf, développeur originaire du Groenland, développe une bibliothèque en langage C permettant de comptabiliser le nombre de visiteurs sur son CV en ligne. Sur sa lancée, Rasmus ajoute de nombreuses fonctionnalités dont la communication avec des bases de données.

En 1995, Rasmus décide de publier son code sous le nom de PHP/FI (Personal Home Page / Form Interpreter) pour que tout le monde puisse en profiter.



En 1997, les étudiants Andi Gutmans et Zeev Suraski, redéveloppent le cœur de PHP/FI. Ce travail aboutit un an plus tard à la version 3 de PHP, devenu alors PHP: Hypertext Preprocessor, soulignant la particularité principale du langage : le PHP est interprété sur le serveur, avant l'envoi des informations au client (visiteur). Par la suite, Andi et Zeev

reprogramment le moteur interne de PHP, appelé Zend Engine (contraction de Zeev et Andi) et servant de base à la version 4 de PHP.

La version 6 projetant de basculer tout le fonctionnement du PHP en Unicode connaît des difficultés et des retards, si bien que c'est finalement une version PHP 5.3 qui sort en 2009 avec d'importantes nouveautés comme les *namespaces* et les fonctions anonymes. Le projet de version 6 est finalement abandonné.

En 2016, la version 7 du langage fait une entrée remarquée avec pour principale évolution, un sérieux gain de performances.

La version 8 est sortie fin 2020.

Actuellement, plus de 80% des sites sont codés en PHP, devant ASP.NET avec environ 15% et Java avec 2.5%. Parmi les concurrents mineurs, on retrouve les technologies ColdFusion, Python, Perl, Ruby et node.js.

Parmi les serveurs HTTP, les principaux concurrents d'Apache sont IIS (*Internet Information Services*), Lighttpd et Nginx.

## B. Intérêts du PHP

Si vous débutez en développement web, l'intérêt principal que vous tirerez de PHP est sans doute sa capacité à générer dynamiquement le contenu des pages web. Imaginez un site de vente en ligne dont le nombre d'informations à présenter est tellement important qu'il devient impossible pour le webmaster de créer un fichier HTML pour chaque article. Avec le PHP, le webmaster crée une seule page et fait varier son contenu en le chargeant par exemple depuis une base de données.

Pour contenir les informations à long terme, PHP s'appuie généralement sur des systèmes de gestion de bases de données, comme MySQL, MariaDB ou Oracle.

Parmi d'autres avantages, PHP permet également de scinder le code d'une page en plusieurs fichiers et de les inclure au besoin. Ceci permet de ne pas coder plusieurs fois les parties communes du site comme la navigation, l'en-tête ou le pied de page.

Comme il est interprété sur le serveur, le PHP garantit un niveau de sécurité que les langages interprétés côté client, comme JavaScript, ne peuvent pas atteindre. C'est pourquoi, le PHP est utilisé pour traiter la plupart des traitements de formulaires : formulaires de connexion, de recherche, d'inscription, d'achat en ligne, etc.

PHP permet également la gestion des sessions (connexions valables sur plusieurs pages), le hachage et l'encryptage des mots de passe, l'exploitation des fichiers plats et des fichiers XML, la récupération de paramètres GET dans les URL, la redirection des visiteurs vers d'autres URL, la gestion des dates, la programmation objet, etc.

## C. Outils nécessaires

Pour développer en PHP, vous devez installer un serveur local simulant un serveur distant. **WAMP** (Windows Apache MySQL PHP) est sans doute le meilleur et le plus connu des packages Windows permettant de disposer de l'ensemble des outils nécessaires :

- Serveur Apache
- Interpréteur de code PHP
- Base de données MySQL

**Laragon** et **XAMP** sont des alternatives intéressantes à WAMP et reprennent les mêmes outils. Si vous ne travaillez pas sous Windows, vous pouvez opter pour **MAMP** ou **LAMP**.

Une fois les outils installés, et pour pouvoir être exécutés, tous vos sites PHP doivent être enregistrés dans des sous-dossiers du dossier "www".

Prenez dorénavant l'habitude d'utiliser l'extension .php au lieu de .html lorsque vos codes contiennent du PHP.

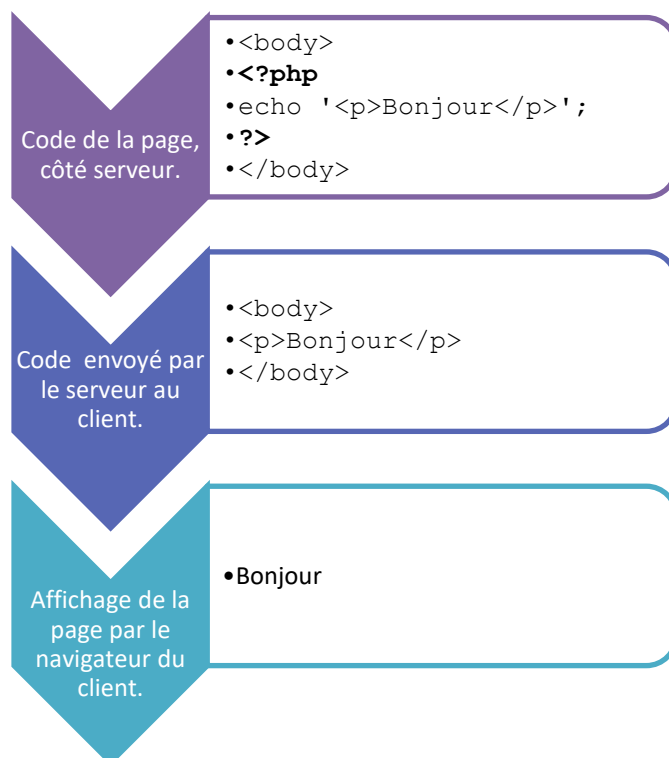
Pour coder les scripts PHP, il convient d'utiliser un éditeur de code comme Notepad++.

## D. Pré-interprétation

Le code PHP est incorporé dans le code HTML, c'est pourquoi une bonne connaissance de l'HTML est indispensable avant de débiter en PHP. Des connaissances en langage C ou en JavaScript sont appréciables suite aux nombreuses similitudes de syntaxe.

Dans un fichier *.php* vous pouvez à tout moment passer du code HTML au code PHP à l'aide des balises : `<?php ?>`

Votre page PHP contient donc plusieurs langages qui cohabitent : du HTML, du PHP et éventuellement d'autres langages comme du CSS ou du JavaScript. Le PHP se différencie de ces autres langages par son interprétation côté serveur, alors que les autres langages sont interprétés côté client, par le navigateur. Ceci implique que le visiteur n'a pas accès au code PHP.



**Figure 1 : exemple de code HTML et PHP ; le même code après l'interprétation du PHP par le serveur ; le résultat après l'interprétation de l'HTML par le navigateur.**

⇒ Lorsque vous visitez un site contenant du PHP et que vous affichez le code source des pages, vous ne voyez jamais une ligne de PHP. Toute l'interprétation du code a été faite au préalable au niveau du **serveur**.

## E. Versions PHP

Le langage PHP est actuellement à sa version 8 dont la principale nouveauté est un gain de 45% de vitesse d'exécution.

Le projet de version 6, visant à passer le fonctionnement du PHP en Unicode, a été abandonnée avant d'aboutir.

Les versions inférieures à la version 5 sont obsolètes et souffrent de méchantes failles de sécurité.

## Exercice 1 : Hello World

### Instructions pour les exercices

Sur l'eCampus, vous trouverez un dossier téléchargeable contenant les fichiers nécessaires aux exercices du syllabus.

Une fois cette suite logicielle installée, placez le dossier d'exercices dans le dossier « C:/wamp/www/ ». Veillez à éviter les accents et les espaces dans les noms de dossiers et de fichiers de ce dossier.

Le premier exercice ci-dessous consiste simplement à s'assurer du bon fonctionnement de WAMP/XAMP/Laragon et à vous habituer à son utilisation.

### Exemple : instruction PHP générant un titre <h1>

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Hello World</title>
</head>
<body>
  <?php
    echo '<h1>Hello World</h1>';
  ?>
</body>
</html>
```

### Exercice :

Au début, on a tendance à ouvrir les fichiers PHP en les double-cliquant ou en les exécutant depuis l'éditeur de texte. Perdez cette habitude et apprenez cette nouvelle manœuvre :

- Lancez WAMP, XAMP ou Laragon.
- Ouvrez votre navigateur.
- Accédez à l'adresse « 127.0.0.1 » ou « localhost » (ça dépend un peu des versions...)
- À l'aide de la navigation, accédez à « exercice1 », « index.php » devrait s'ouvrir (par défaut, sans fichier spécifié dans une url, c'est le fichier appelé *index* qui s'ouvre).
- Si vous avez bien suivi la manœuvre, le PHP a été interprété et vous constaterez un joli titre « Hello World ».

Remarquez dans cet exercice l'emploi de la commande *echo* servant à générer un contenu. Remarquez également le point-virgule servant à terminer une instruction PHP.



## 1.2 Inclusions et redirections

### A. Inclure un fichier

La bonne pratique du PHP veut que les parties répétitives d'un site web soient isolées dans des fichiers externes afin de ne pas être codées plusieurs fois. C'est le cas par exemple de l'en-tête, de la navigation, du pied de page, des déclarations de fonctions ou encore de l'accès à la base de données.

Les 4 instructions suivantes permettent d'inclure un fichier :

1. ***include***
2. ***include\_once*** : idem que *include* avec un test pour n'inclure le fichier que s'il n'a pas été inclus auparavant.
3. ***require*** : idem que *include* avec en plus la génération d'une erreur qui arrête l'interprétation du script si le fichier cherché n'existe pas (d'où le nom *require* pour indiquer que le fichier est requis).
4. ***require\_once*** : idem que *require* avec un test pour n'inclure le fichier que s'il n'a pas été inclus auparavant.

Exemple :

```
<?php
require_once 'include_connectBD.php' ;
include_once 'fonctions.php' ;
include 'include_nav.php' ;
?>
```

### B. Rediriger l'utilisateur

La fonction ***header()*** envoie un en-tête http brut au serveur. Cet en-tête peut contenir une nouvelle destination (*Location*) afin de rediriger le visiteur vers une autre URL.

La commande ***exit*** met fin à l'interprétation du code PHP. Il est de bon usage de suivre une redirection par un *exit* afin d'arrêter l'interprétation indésirable du reste de la page.

```
header('Location:index.php');
exit;
```

Attention, cette fonction *header()* doit impérativement être appelée avant le moindre contenu, que ce soit une balise HTML ou même un simple caractère, un espace ou un retour à la ligne. Pour son bon fonctionnement, il est donc obligatoire de commencer le fichier par une ouverture du PHP en ligne 1 !

## Exercice 2 : Inclusions de fichiers

Dans cet exercice, le site comporte des parties communes comme la navigation ou l'en-tête. Pour éviter de coder plusieurs fois ces parties communes, on les code dans des fichiers que l'on inclut aux pages.

### Exemple : fichier PHP incluant d'autres fichiers

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Inclusion de fichiers</title>
</head>
<body>
  <?php
    include '_header.php' ;
    include '_nav.php' ;
  ?>
  <h2>Accueil</h2>

</body>
</html>
```

### Exemple : contenu de « \_header.php »

```
<header>
  <h1>Inclusions de fichiers</h1>
</header>
```

### Exemple : contenu de « \_nav.php »

```
<nav>
  <ul>
    <li><a href="index.php">Accueil</a>
    <li><a href="administration.php">Administration</a>
    <li><a href="contact.php">Contact</a>
  </ul>
</nav>
```

### Exercice :

- Créez un fichier « \_footer.php » contenant un élément <footer> et un paragraphe avec vos nom et prénom. Puis, incluez ce fichier au fichier « index.php ».
- Codez une page « administration.php » et une page « contact.php » qui incluent également le *header*, la *nav* et le *footer*.
- Dans la page « administration.php », codez une redirection vers « index.php » de façon que les visiteurs ne puissent pas accéder à « administration.php ».

## 1.3 Variables

### A. Définition

Les variables sont des espaces mémoires permettant de stocker les données nécessaires au bon déroulement d'un script. Par exemple, si vous codez un script qui calcule la surface d'un rectangle, vous aurez sûrement besoin d'une variable pour stocker la longueur, une autre pour stocker la largeur et une dernière pour stocker le résultat.

D'un point de vue technique, les variables se distinguent les unes des autres par leur adresse mémoire, mais du point de vue du développeur, on leur choisit un nom pour les manipuler avec plus de facilité. Pour des questions de lisibilité du code, il est conseillé de choisir des noms explicites, par exemple : *\$longueur*, *\$surface* ou *\$surface\_rectangle*. Evitez les noms trop longs, trop courts ou inadéquats comme *\$surfaceDuPremierRectangleEnMetresCarres*, *\$x*, *\$toto* ou *\$machinTruc*.

**Les variables sont donc des conteneurs permettant de stocker des valeurs et repérées grâce à leur adresse ou à leur nom.**

### B. Noms de variables

Il appartient au développeur de choisir les noms de variables. Pour cela, il doit se conformer à deux ensembles de règles : les règles de bon sens qui demandent de choisir des noms de variables appropriés, représentatifs des valeurs contenues ; et les règles strictes qui dépendent du langage de programmation utilisé et qui empêcheraient l'interprétation du script si elles n'étaient pas respectées.

En PHP, les **noms des variables** :

- commencent par l'identifiant **\$**
- ne comportent que des lettres (accents permis mais déconseillés), des chiffres et le caractère **"\_"**
- ne comportent pas de chiffre juste après le **\$**
- sont sensibles à la casse
- n'ont pas de limite de longueur

Noms de variables permis :	Noms de variables interdits :
<code>\$poids</code> <code>\$_date</code> <code>\$age_2008</code> <code>\$nom2</code> <code>\$_123</code>	<code>\$2008age</code> <code>\$client.adresse</code> <code>\$*prenom</code> <code>\$nom de variable</code> <code>x</code>

**Exercice (papier) : déterminez si les noms de variables suivants sont corrects**

```

commentaire
$commentaire
$2020reduction
$reduction2020
$_prix
$x
$année
$date.naissance

```

**C. Affectations, déclarations et destructions**

**Affectation de variable** : L'affectation d'une variable est l'action de lui donner une nouvelle valeur. L'opérateur d'affectation est =

La partie affectée se trouve à gauche du symbole égal, la nouvelle valeur à droite.

```

<?php
$x = 10 ;
?>

```

**Déclaration de variable** : La déclaration d'une variable est une instruction demandant la réservation d'un espace mémoire et l'attribution d'un nom.

En PHP, la déclaration des variables est automatique : lorsque l'on affecte une variable, celle-ci est implicitement déclarée. En revanche, il est interdit d'utiliser la valeur d'une variable non déclarée, cela provoquerait une erreur.

<p><b>Manœuvre autorisée</b> : on stocke 10 dans \$x. \$x n'existant pas, il est implicitement déclaré.</p> <pre> &lt;?php \$x = 10 ; ?&gt; </pre>	<p><b>Manœuvre interdite</b> : on stocke \$b +10 dans \$a. Si \$b n'existe pas, impossible d'utiliser sa valeur.</p> <pre> &lt;?php \$a = \$b + 10 ; ?&gt; </pre>
--	---

**Destruction de variable** - La fonction `unset()` détruit une variable, libérant ainsi de la mémoire inutilement occupée.

```
unset($variableQuiNeSertPlusARien) ;
```

## D. Tester l'existence d'une variable

Les fonctions `isset()` et `empty()` permettent de contrôler l'existence d'une variable :

```
<?php
if (isset($var)) {
    //Si $var a une valeur quelconque, y compris 0
}
if (empty($var)) {
    //Si $var est inexistante, vaut 0 ou NULL
}
?>
```

Ces deux fonctions permettent entre autres d'éviter des erreurs d'interprétation, en testant l'existence d'une variable avant de l'utiliser.

	La variable est inexistante	La variable est de type NULL	La variable vaut 0 ou ""	La variable a une valeur différente de 0
<code>isset(\$var)</code>	<b>False</b>	<b>False</b>	<b>True</b>	<b>True</b>
<code>empty(\$var)</code>	<b>True</b>	<b>True</b>	<b>True</b>	<b>False</b>
<code>!isset(\$var)</code>	<b>True</b>	<b>True</b>	<b>False</b>	<b>False</b>
<code>!empty(\$var)</code>	<b>False</b>	<b>False</b>	<b>False</b>	<b>True</b>

Le « ! » est l'opérateur logique de négation : il donne la négation du résultat.

**Exercice (papier) : que valent \$x et \$toto à la fin de ce script ?**

```
<?php
$toto = 312 ;
$x = isset($toto);
unset($toto);
?>
```

## E. Affectations par référence

On distingue 2 types d'affectation :

1. Affectation par valeur :	2. Affectation par référence (&) :
<pre>&lt;?php \$A = 'Mons'; \$B = 'Bruxelles'; \$A = \$B; \$B = 'Charleroi'; ?&gt;</pre>	<pre>&lt;?php \$A = 'Mons'; \$B = 'Bruxelles'; \$A = &amp;\$B; \$B = 'Charleroi'; ?&gt;</pre>
A la fin de ce script, \$A vaut "Bruxelles"	A la fin de ce script, \$A vaut "Charleroi"

Dans le premier exemple (affectation par valeur), l'instruction `$A=$B;` affecte la **valeur** de `$B` à `$A`. Lorsque par la suite, on affecte la valeur "Charleroi" à `$B`, `$A` n'est pas affecté.

Dans le second exemple (affectation par adresse), l'instruction : `$A=&$B;` affecte l'**adresse** de `$B` à `$A`, ce qui crée un lien entre les deux variables. Lorsque par la suite, on affichera `$A`, ce sera la valeur de `$B` qui sera affichée.

**Exercice (papier) : que valent \$x, \$y et \$z à la fin de ce script ?**

```
$x = 'HTML';
$y = 'CSS';
$z = &$x;
$x = $y;
$y = 'PHP';
```

## F. Constantes

Si vous êtes amenés à utiliser de manière répétitive des valeurs devant rester constantes, PHP vous permet de déclarer des constantes, selon **2 syntaxes** possibles :

```
<?php
const WIDTH = 800 ;
define ("PI", 3.1415926535, TRUE);
?>
```

- « **WIDTH** » et « **PI** » : les noms de constantes ne commencent pas par « \$ ».
- Définir une constante avec `define()` permet de choisir si le nom est insensible à la casse (TRUE = insensible, FALSE = sensible). Ce qui signifie que vous pouvez faire appel à cette constante en l'appelant "**pi**", "**PI**", "**Pi**" ou "**pl**".

## G. Types de variables

Chaque variable PHP fait partie d'un de ces 8 types :

- entier : **integer** (-4 ou 36)
- réel (ou flottant) : **double** (51.8)
- booléen : **boolean** (True ou False)
- chaîne de caractères : **string** ("Martine" ou 'Guy')
- tableau: **array**
- objet : **object**
- ressource : **resource**
- néant : **NULL**

La fonction `gettype()` renvoie le type de la variable passée en paramètre.

```
<?php
$type = gettype ($var) ;
?>
```

Comme le type des variables n'est pas figé, il est parfois utile de le tester avec les fonctions suivantes :

<code>is_integer (\$x)</code>	<code>is_double (\$x)</code>	<code>is_array (\$x)</code>
<code>is_int (\$x)</code>	<code>is_numeric (\$x)</code>	<code>is_object (\$x)</code>
<code>is_float (\$x)</code>	<code>is_bool (\$x)</code>	<code>is_resource (\$x)</code>
<code>is_real (\$x)</code>	<code>is_string (\$x)</code>	<code>is_null (\$x)</code>

Ces fonctions ramènent une valeur booléenne (**TRUE/FALSE**). Elles conviennent donc à une utilisation dans une structure conditionnelle **if** :

```
<?php
if (is_int ($var)) { /*$var est de type entier*/ }
?>
```

**Exercice (papier) : de quels types sont \$x, \$y et \$z après ce script ?**

```
<?php
$x='PHP forever';
$y=isset ($a) ;
$z='20';
?>
```

**Le PHP est un langage à typage dynamique.** Cela signifie que le type des variables n'est pas figé et peut changer durant l'exécution des scripts. Il existe 2 façons de changer le type d'une variable :

➔ Le type peut changer automatiquement lors d'une affectation :

```
<?php
$x = true;    //$x est un booléen
$x = 23;      //$x devient un entier
$x = 0.0;     //$x devient un réel
$x = 'Bob';   //$x devient une chaîne de caractères
?>
```

➔ Le type peut aussi changer lorsqu'on le force en précisant le nouveau type entre parenthèses, devant la variable concernée :

```
<?php
$var = '7000 Mons';    //$var est de type string
$var = (int) $var;      //$var devient integer et vaut 7000
$var = (bool) $var;     //$var devient bool et vaut True
?>
```

Lorsqu'on force un changement de type, la valeur s'adapte automatiquement. Dans l'exemple ci-dessus, la variable contient "7000 Mons" et est de type *string*. Après le changement de type en *integer*, elle s'est adaptée et vaut 7000.

Les conversions possibles sont :

- **(int), (integer)** : conversion en entier
- **(bool), (boolean)** : modification en booléen
- **(float), (double), (real)** : modification en réel
- **(string)** : modification en chaîne de caractères
- **(array)** : modification en tableau
- **(object)** : modification en objet
- **(unset)** : modification en NULL

**Exercice (papier) : que valent \$a, \$b et \$c après ce script ?**

```
<?php
$a='73.48 kg';
$b=(double) $a;
$c=(integer) $b;
$a=(boolean) $c;
?>
```



## H. Chaînes de caractères

Pour contenir les chaînes de caractères, deux syntaxes sont admises : les **guillemets** ou les **apostrophes**. Pour choisir l'une ou l'autre syntaxe, 3 paramètres entrent en jeu :

1. Les guillemets causent des collisions avec les guillemets ; les apostrophes avec les apostrophes. Pour éviter les collisions, il faudra échapper les caractères avec `\` (*backslash*) :

```
<?php
$nom1 = 'Je m\'appelle Michael "Monty" Widenius';
$nom2 = "Je m'appelle Michael \"Monty\" Widenius";
?>
```

2. les guillemets permettent l'évaluation d'une variable mais pas les apostrophes :

```
<?php
$place = 'Mons';
$a = "J'habite $place "; //vaut : J'habite Mons
$a = 'J\'habite $place'; //vaut : J'habite $place
?>
```

3. dû à cette évaluation, les guillemets sont légèrement plus lents à l'interprétation. Ce qui fait qu'en général, les développeurs préfèrent utiliser les apostrophes.

### Caractères spéciaux accessibles avec un *backslash*

<code>\\</code>	Affiche un backslash (antislash)
<code>\'</code>	Affiche une apostrophe
<code>\"</code>	Affiche des guillemets
<code>\\$</code>	Affiche un \$
<code>\n</code>	Nouvelle ligne
<code>\r</code>	Retour chariot
<code>\t</code>	Tabulation horizontale
<code>\115</code>	Nombre octal affichant le caractère correspondant : M
<code>\x4D</code>	Nombre hexadécimal affichant le caractère correspondant : M

Remarque : si la chaîne est entre apostrophe, seuls les 2 premiers codes `\\` et `\'` fonctionnent.

**Fonctions liées à l'utilisation des chaînes de caractères** (pour une documentation complète, visitez : <http://php.net/manual/fr/ref.strings.php>)

#### Longueur

**strlen**(\$ch) renvoie le nombre d'octets d'une chaîne  
**iconv\_strlen**(\$ch) renvoie le nombre de caractères d'une chaîne

#### Majuscules

**strtolower**(\$ch) renvoie la chaîne avec tous les caractères en minuscules  
**strtoupper**(\$ch) renvoie la chaîne avec tous les caractères en majuscules  
**mb\_strtoupper**(\$ch, 'UTF-8') Idem, avec gestion des accents en UTF-8  
**ucwords**(\$ch) renvoie la chaîne avec des majuscules en début de chaque mot  
**ucfirst**(\$ch) renvoie la chaîne avec la première lettre du premier mot en majuscule

#### Espaces

**ltrim**(\$ch) renvoie la chaîne sans les espaces situés en début de chaîne  
**rtrim**(\$ch) renvoie la chaîne sans les espaces situés en fin de chaîne  
**trim**(\$ch) renvoie la chaîne sans les espaces situés en début et en fin de chaîne

#### Caractères spéciaux et formats d'encodage

**addslashes**(\$ch) renvoie la chaîne avec des "\" devant les caractères (") (') (\) et (NUL)  
**stripslashes**(\$ch) renvoie la chaîne sans les "\" devant ces mêmes caractères spéciaux  
**quotemeta**(\$ch) renvoie la chaîne avec des "\" devant les caractères " ' \ + \* ? [ ] ( ) \$ . et ^  
**htmlspecialchars**(\$ch) renvoie la chaîne avec les caractères & " ' < > convertis en entités de caractères interprétables par les navigateurs  
**htmlentities**(\$ch) renvoie la chaîne avec tous les caractères dont le code UNICODE est supérieur à 128 en entités de caractères interprétables par les navigateurs  
**html\_entity\_decode**(\$ch) fonction inverse  
  
**utf8\_encode**(\$ch) convertit une chaîne de ISO-8859-1 vers UTF-8  
**utf8\_decode**(\$ch) convertit une chaîne de UTF-8 vers ISO-8859-1  
  
**strip\_tags**(\$ch) renvoie la chaîne de caractères sans les balises HTML qu'elle comportait.

## Affichage de nombres

<code>number_format(\$x)</code>	Renvoie un nombre \$x sous forme de chaîne avec une virgule de séparation des milliers (par exemple : 7,550,000)
<code>number_format(\$x, \$d)</code>	Renvoie un nombre \$x sous forme de chaîne avec \$d décimales
<code>number_format(\$x, \$d, \$sd, \$sm)</code>	Renvoie un nombre \$x sous forme de chaîne avec \$d décimales, \$s1 comme séparateur de décimales et \$s2 comme séparateur de milliers.

## Recherche d'une sous-chaîne de caractères dans une chaîne de caractères

<code>strstr (\$ch1, \$ch2)</code>	renvoie tous les caractères de \$ch1 allant de la première occurrence de \$ch2 dans \$ch1 jusqu'à la fin de \$ch1.
<code>stristr (\$ch1, \$ch2)</code>	idem sans la sensibilité à la casse.
<code>strrchr (\$ch1, \$ch2)</code>	renvoie tous les caractères de \$ch1 allant de la dernière occurrence de \$ch2 dans \$ch1 jusqu'à la fin de \$ch1.
<code>substr_count (\$ch1, \$ch2)</code>	renvoie le nombre d'occurrences de \$ch2 dans \$ch1.
<code>substr (\$ch, \$start, \$len)</code>	renvoie un segment de la chaîne \$ch : si \$start est positif, par exemple 3, le segment commence à la 3 <sup>e</sup> lettre en partant du début et en comptant à partir de 0 ; si \$start est négatif, par exemple -3, le segment commence à la 3 <sup>e</sup> lettre en partant de la fin. Le paramètre optionnel \$len peut spécifier la longueur maximale du segment.
<code>str_replace (\$ch1, \$ch2, \$ch3)</code>	remplace toutes les occurrences de \$ch1 dans \$ch3 par \$ch2.

Exemple : ci-dessous, ***str\_replace()*** est utilisée pour supprimer tous les guillemets avant de générer *\$description* dans l'attribut *content*. Un guillemet dans la valeur de l'attribut signifierait la fin de cet attribut et entraînerait des erreurs HTML.

```
echo '<meta name="description" content="' . str_replace('"',
'', $description), '">';
```

<code>explode (\$ch1, \$ch2)</code>	Segmente la chaîne \$ch2 selon les occurrences de la chaîne \$ch1.
-------------------------------------	--

## Exemple : ci-dessous, on exploise une chaîne de caractères

```
$x = 'piece1 piece2 piece3 piece4 piece5 piece6';
$pieces = explode(" ", $x);
echo $pieces[0]; // piece1
echo $pieces[1]; // piece2
```

## I. *echo*

La commande *echo* permet de **générer** un contenu (HTML, CSS, JavaScript, etc.).

Exemple 1 : générer du texte

```
echo 'I love PHP' ;  
echo "I love PHP" ;  
  
echo 'J\'aime PHP' ;  
echo "J'aime PHP" ;
```

Exemple 2 : générer du code HTML

```
echo '<h1 id="titre">Cool</h1>';  
echo "<h1 id=\"titre\">Cool</h1>";
```

Exemple 3 : générer la valeur d'une variable

```
echo $a;  
echo "$a";
```

Exemple 4 : générer le nom d'une variable

```
echo '$a';
```

Exemple 5 : générer un peu de tout

```
echo '<p class="red">$x vaut <strong>', $x, '</strong> </p>' ;  
echo "<p class=\"red\">\$x vaut <strong>\$x</strong> </p>" ;
```

Exemple 6 : utilisez les **Short Open Tags** pour une écriture plus concise

```
<?php echo $a; ?> devient <?= $a; ?>
```

## J. Concaténations

La **concaténation** est une opération qui fusionne deux chaînes de caractères en une seule. L'opérateur de concaténation est le **point** '.'

Exemple 1 :

```
$chezmoi='Mons';  
$a='J\'habite '.$chezmoi.' et pas Charleroi.';  
echo $a;      //affiche : J'habite Mons et pas Charleroi.
```

À ne pas confondre avec la **virgule** qui permet de séparer des paramètres multiples dans une fonction ou dans certaines commandes comme **echo**.

Exemple 2 :

```
$chezmoi='Mons';  
echo 'J\'habite ', $chezmoi, ' et pas Charleroi';  
    //affiche : J'habite Mons et pas Charleroi.
```

## K. Variables dynamiques



Même s'il est plutôt rare d'utiliser cette particularité, en PHP le nom des variables peut lui-même être une variable. On parle alors de variable dynamique.

```
<?php  
$prix = 300 ;  
$x = 'prix';  
echo $$x ;    //affiche 300  
?>
```

L'utilisation d'**accolades** permet de construire un nom de variable à partir d'une expression.

```
<?php  
$prix_voiture = 3000 ;  
$x = 'voiture';  
echo ${'prix_'.$x} ;    //affiche 3000  
?>
```

## L. Opérateurs de base

### Opérateurs arithmétiques :

Addition	+	
Soustraction	-	
Multiplication	*	
Exponentiation	**	
Modulo	%	Reste de la division
Division	/	

### Opérateurs d'incrémentation :

Incrémentation	++\$i ;	ou	\$i++ ;
Décrémentation	--\$i ;	ou	\$i-- ;

### Opérateurs de concaténation :

Concaténation	.
---------------	---

### Opérateurs d'affectation :

Affectation	=
-------------	---

### Opérateurs d'affectation élargie :

\$i += \$k ;	revient à	\$i = \$i + \$k ;
\$i -= \$k ;	revient à	\$i = \$i - \$k ;
\$i *= \$k ;	revient à	\$i = \$i * \$k ;
\$i /= \$k ;	revient à	\$i = \$i / \$k ;
\$i %= \$k ;	revient à	\$i = \$i % \$k ;
\$i .= \$k ;	revient à	\$i = \$i.\$k ;

### Opérateurs logiques :

AND	et	
OR	ou	
&&	et	(niveau de priorité plus élevé que AND)
	ou	(niveau de priorité plus élevé que OR)
XOR	ou exclusif	
!	contraire de...	

### Opérateurs relationnels :

Plus grand que	>
Plus petit que	<
Plus grand ou égal à	>=
Plus petit ou égal à	<=
Égal à	==
Différent de	!=
Égal à (valeur ET type)	===
Différent de (valeur ET type)	!==

## Exercice 3 : Opérations sur les variables

Dans cet exercice, le prix d'un article est converti et affiché dans différentes monnaies.

### Exemple : fichier PHP incluant d'autres fichiers

```
<?php
$taux_USD = 0.884 ;
$taux_GBP = 1.225 ;
$taux_SEK = 0.096 ;
?><!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Opérations sur les variables</title>
</head>
<body>
<?php
?>
</body>
</html>
```

### Exercice :

- Complétez le <body> avec l'initialisation d'une variable \$prix\_EUR valant 18,50 et affichez ce prix en euros dans une balise <p>.
- Complétez le <body> pour afficher les prix dans les 3 autres monnaies calculés à partir des variables \$taux\_USD, \$taux\_GBP et \$taux\_SEK.
- Veillez à ce que chaque prix affiché dans cet exercice comporte exactement 2 décimales, que les décimales soient séparées des unités par une virgule et que les milliers soient séparés par un espace (voir la fonction *number\_format()*) :

```
Prix en euros : 18.50 €
Prix en dollars : 20.93$
Prix en livres sterling : 15.10 £
Prix en couronnes suédoises : 192.71 SEK
```

- Si vous modifiez le prix en euros dans le code et que vous rechargez la page, les autres prix doivent s'adapter.

## M. Tableaux

Les tableaux sont des variables permettant de stocker plusieurs valeurs sous un même nom.

En PHP, les tableaux sont dynamiques, c'est-à-dire que l'on peut en modifier le nombre de valeurs pendant l'exécution du script. Contrairement aux tableaux statiques dont le nombre d'éléments est figé lors de leur déclaration. Cet avantage a cependant un coût en mémoire : les valeurs stockées dans ces tableaux peuvent prendre un espace mémoire jusqu'à 20 fois supérieur.

Contrairement au langage C, en PHP les valeurs d'un même tableau peuvent être de types différents : *integer*, *double*, *boolean*, *string*, ou même *array*, ce qui permet de créer des tableaux de tableaux, c'est-à-dire des tableaux à plusieurs dimensions.

Exemple de déclaration de tableau :

```
$navigation = array('accueil', 'galerie', 'contact');
```

Exemple de déclaration de tableau avec indices numériques :

```
$navigation = array(0=>'accueil', 1=>'galerie', 2=>'contact');
```

À des fins de débogage, deux façons d'afficher tout un tableau (ou un objet) :

```
var_dump($navigation);  
print_r ($navigation);
```

**Il existe 2 types de tableaux : les tableaux indicés et les tableaux associatifs.**

**Dans les tableaux indicés, chaque élément est repéré par un indice numérique, c'est-à-dire un entier (en général à partir de 0).**

**Dans les tableaux associatifs, les éléments sont repérés par des indices textuels, c'est-à-dire des chaînes de caractères.**



## N. Tableaux indicés

Exemple de tableau indicé :

```
$tab[0] = 2007;
$tab[1] = 'ISIMS';
$tab[50] = 3.875;
$tab[] = TRUE;
echo 'Nombre d\'éléments = ', count($tab);
```

- ➔ Le fait d'affecter une valeur à la variable **\$tab[0]**, et le fait que cette valeur soit suivie d'un indice entre crochets **[0]**, fait de **\$tab** un tableau. C'est donc la présence des crochets qui indique que cette variable est de type *array*.
- ➔ La fonction **count(\$tab)** va donner le nombre d'éléments de ce tableau, et tenez-vous bien, ce nombre est 4 ! PHP permet d'utiliser des indices non successifs (dans l'exemple, on passe de 1 à 50) sans que les éléments intermédiaires n'existent inutilement.
- ➔ Le dernier élément du tableau **\$tab[] = TRUE;** n'a pas d'indice précisé. Dans ce cas, son indice sera celui qui suit le dernier indice existant, c'est-à-dire 51.

## O. Tableaux associatifs

Contrairement aux tableaux indicés où les indices sont des nombres entiers, les tableaux associatifs ont des indices de type chaînes de caractères.

```
$tab['an'] = 2020 ;
```

Attention aux interférences entre guillemets et apostrophes lors de l'utilisation d'un tableau associatif dans une chaîne de caractères :

```
echo "Nous sommes en $tab['an']"; //provoque une erreur !
echo "Nous sommes en $tab[\"an\"]"; //provoque une erreur !
echo "Nous sommes en {$tab['an']}"; //correct
echo "Nous sommes en {$tab[\"an\"]}"; //correct
echo 'Nous sommes en ', $tab['an']; //correct
echo 'Nous sommes en ', $tab[\"an\"]'; //correct
```

Les 2 premiers **echo** ci-dessus provoquent une erreur due à l'interférence entre les paires de guillemets ou paires d'apostrophes. Cette erreur est cependant corrigible en ajoutant des **accolades** ou en sortant la variable de la chaîne de caractères.

## P. Fonctions sur les tableaux

<code>count(\$tab)</code>	Renvoie le nombre de valeurs du tableau
<code>in_array(\$v, \$tab)</code>	Renvoie <i>True</i> si la valeur <i>\$v</i> appartient à un tableau
<code>array_key_exists(\$k, \$tab)</code> <code>key_exists(\$k, \$tab)</code>	Renvoie <i>True</i> si la clé <i>\$k</i> existe dans un tableau Alias de la fonction précédente
<code>array_key_first(\$tab)</code> <code>array_key_last(\$tab)</code>	Renvoie la première clé d'un tableau Renvoie la dernière clé d'un tableau
<code>array_sum(\$tab)</code>	Renvoie la somme des valeurs du tableau
<code>\$tab=array_fill(\$a, \$n, \$v)</code>	Affecte la valeur <i>\$v</i> à <i>\$n</i> éléments en commençant à l'indice <i>\$a</i> .
<code>sort(\$tab)</code>	Trie le tableau
<code>asort(\$tab)</code>	Trie le tableau et conserve l'association des index
<code>arsort(\$tab)</code>	Trie le tableau dans l'ordre inverse et conserve l'association des index
<code>ksort(\$tab)</code>	Trie le tableau selon les clés par ordre croissant
<code>krsort(\$tab)</code>	Trie le tableau selon les clés par ordre décroissant
<code>shuffle(\$tab)</code>	Mélange le tableau
<code>array_rand(\$tab)</code>	Renvoie une clé (ou plusieurs si second paramètre) piochée aléatoirement parmi les clés du tableau
<code>each(\$tab)</code>	Retourne chaque paire clé/valeur une à une
<code>reset(\$tab)</code>	Remet le pointeur interne du tableau à zéro

Ces quelques fonctions ne constituent qu'un maigre échantillon de toutes les fonctions sur les tableaux. Voir la doc complète : <http://php.net/manual/fr/ref.array.php>

**Exercice (papier) : quels seront les résultats générés par les *echo* suivants ?**

```
<?php
$nom='Roger';
$tab[0]='Mons';
$adresse['pays']='Belgique';

echo $nom;
echo "$nom";
echo '$nom';

echo $tab[0];
echo "$tab[0]";
echo '$tab[0]';

echo $adresse["pays"];
echo $adresse['pays'];
echo '$adresse["pays"]';
echo "$adresse['pays']";
echo "$adresse["pays"]";
echo "{$adresse["pays"]}";

echo $nom." habite ".$tab[0];
echo $nom," habite ",$tab[0];
echo "$nom habite $tab[0]";
echo '$nom habite $tab[0]';
?>
```



## 1.4 Dates

### A. Préciser le fuseau horaire

Pour se prémunir contre les problèmes de fuseau horaire et d'heures d'été/hiver, mieux vaut spécifier l'instruction suivante avant de manipuler des dates :

```
date_default_timezone_set('Europe/Brussels');
```

Cette fonction définit le décalage horaire par défaut de toutes les fonctions manipulant la date et l'heure.

### B. *getdate()*

Première solution pour manipuler les dates, la fonction ***getdate()*** ramène la date actuelle sous forme de tableau associatif dont voici les 11 indices :

'year'	contient l'année en entier de 4 chiffres.
'month'	contient le mois de <i>January</i> à <i>December</i> .
'mon'	contient le mois sous forme d'entier de 1 à 12.
'wday'	contient le jour de la semaine sous forme d'entier de 0 à 6.
'weekday'	contient le jour de la semaine de <i>Sunday</i> à <i>Saturday</i> .
'mday'	contient le jour du mois sous forme d'entier de 1 à 31.
'yday'	contient le jour de l'année de 1 à 366.
'hours'	contient l'heure de 0 à 23.
'minutes'	contient les minutes de 0 à 59.
'seconds'	contient les secondes de 0 à 59.
0	contient le <i>timestamp</i> .

Exemple d'affichage de la date :

```
$now=getdate();  
echo "{$now['mday']} / {$now['mon']} / {$now['year']}";
```

### C. *date()*

Deuxième solution pour manipuler les dates, la fonction ***date()*** peut ramener de nombreuses informations à propos de la date selon les paramètres passés. Il existe plus de trente paramètres possibles dont voici les plus utiles :

<code>date('Y')</code>	ramène l'année en quatre chiffres
<code>date('y')</code>	ramène l'année en deux chiffres
<code>date('L')</code>	ramène 1 si l'année est bissextile, sinon 0
<code>date('m')</code>	ramène le mois en deux chiffres
<code>date('n')</code>	ramène le mois en 1 ou 2 chiffres
<code>date('F')</code>	ramène le mois en toutes lettres en anglais
<code>date('M')</code>	ramène les 3 premières lettres du mois en anglais
<code>date('t')</code>	ramène le nombre de jour du mois, entre 28 et 31
<code>date('d')</code>	ramène le jour du mois en 2 chiffres, de 01 à 31
<code>date('j')</code>	ramène le jour du mois, de 1 à 31
<code>date('l')</code>	ramène le jour de la semaine en toutes lettres en anglais
<code>date('D')</code>	ramène le jour de la semaine en 3 lettres, de Mon à Sun
<code>date('w')</code>	ramène le jour de la semaine en 1 chiffre, de 0 (dimanche) à 6 (samedi)
<code>date('N')</code>	ramène le jour de la semaine en 1 chiffre, de 1 (lundi) à 7 (dimanche)
<code>date('z')</code>	ramène le jour de l'année, de 0 à 365
<code>date('A')</code>	ramène AM ou PM
<code>date('g')</code>	ramène l'heure de 1 à 12
<code>date('G')</code>	ramène l'heure de 0 à 23
<code>date('h')</code>	ramène l'heure de 01 à 12
<code>date('H')</code>	ramène l'heure de 00 à 23
<code>date('I')</code>	ramène 1 si l'heure d'été est activée, sinon 0
<code>date('T')</code>	ramène l'abréviation du fuseau horaire
<code>date('i')</code>	ramène les minutes de 00 à 59
<code>date('s')</code>	ramène les secondes de 00 à 59
<code>date('T')</code>	ramène l'abréviation du fuseau horaire

La fonction *date()* peut également combiner les paramètres comme ceci :

```
echo date('j/n/Y');
```

## D. Les *timestamp*

Un *timestamp*, est le nombre de secondes écoulées entre le 1<sup>er</sup> janvier 1970 à 0 h 0 (début de l'époque UNIX) et un moment donné.

La fonction *date()* peut prendre en deuxième paramètre un *timestamp* pour travailler sur un autre moment que le moment actuel.

```
echo date('j/n/Y', 1546784970);
```

De même manière, la fonction *getdate()* peut également prendre en paramètre un *timestamp* afin de travailler sur un moment donné plutôt que sur le moment actuel.

```
$now=getdate(1546784970);
```

La fonction *time()* permet d'obtenir le *timestamp* actuel. La fonction *microtime(true)* permet d'obtenir le *timestamp* sous forme de réel avec les microsecondes en virgules.

```
echo time();
```

La fonction *mktime()* renvoie le *timestamp* d'une date, à partir de 6 paramètres : *mktime(heures, minutes, secondes, mois, jours, années)*

```
echo date('j/n/Y', mktime(16, 14, 0, 2, 15, 2017));
```

La fonction *strtotime()* renvoie le *timestamp* d'une date passée sous forme de chaîne de caractères. Cette chaîne de caractères supporte des écritures diverses.

```
echo strtotime("22 October 2021");  
echo strtotime("2021-10-22");  
echo strtotime("+1 day");
```

**Exercice (papier) : Quelle instruction PHP permet de connaître le jour de la semaine de votre naissance ?**

## Exercice 4 : Utilisation de la date

### Exemple :

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Jours avant le weekend</title>
</head>
<body>
    <h1>Vivement le weekend !</h1>
    <?php
        //Fuseau horaire
        date_default_timezone_set('Europe/Brussels');
        $jours = array('dimanche', 'lundi', 'mardi', 'mercredi',
            'jeudi', 'vendredi', 'samedi');

        //Calcul du nombre de jours restant avant samedi
        $joursAvantWE = 6 - date('w') ;

        //Affichage
        echo '<p>Vivement ', $jours[6], ' !</p>';
        echo '<p>Nous sommes ', $jours[date('w')], '... plus que
            ', $joursAvantWE, ' jours avant ', $jours[6], '.</p>';
    ?>
</body>
</html>
```

### Exercice :

- Complétez le code afin d'afficher la date actuelle sous ce format :  
**Lundi, le 4 septembre 2017, 09h36.**
- Proposez un script PHP permettant d'afficher le nombre de secondes qui se sont écoulées depuis votre naissance.  
**Il s'est écoulé xxx secondes depuis votre naissance.**
- Utilisez ce nombre de secondes pour afficher le nombre de jours.  
**Il s'est écoulé xxx jours depuis votre naissance.**

## 1.5 Structures conditionnelles et itératives

Les **structures conditionnelles** sont des **tests** permettant d'exécuter ou pas des instructions.

Les **structures itératives** sont des **boucles** permettant de répéter des instructions.

### A. *if*

L'instruction *if* est la structure conditionnelle la plus basique. On la retrouve dans tous les langages de programmation (avec une syntaxe parfois légèrement différente). Elle permet de poser une condition avant certaines instructions.

```
if (/*condition*/) {  
    //instructions 1  
}  
else {  
    //instructions 2  
}
```

Une condition n'a que deux résultats possibles : vrai ou faux !

Si la condition est vraie, les instructions 1 sont exécutées, **sinon** les instructions 2 sont exécutées.

#### Remarques :

- **La condition est toujours entre parenthèses.** Dans l'exemple ci-dessus elle a été remplacée par un commentaire ce qui n'est évidemment pas valide.
- Les accolades ne sont pas obligatoires lorsque le *if* débouche sur une instruction unique :

```
if (/*condition*/) //instructions 1
```

- L'utilisation de *elseif* (ou *else if*) permet de poser une nouvelle condition dans le cas où la première est fausse. Les instructions 4 du *else* ne se réalisent dans ce cas que lorsque toutes les conditions sont fausses :

```
if (/*condition 1*/) //instructions 1  
else if (/*condition 2*/) //instructions 2  
elseif (/*condition 3*/) //instructions 3  
else //instructions 4
```

- Il est possible de poser des conditions multiples liées par les opérateurs logiques (et, ou, etc.). Dans cet exemple, on demande si \$x est supérieur à 20 et divisible par 2 :

```
if (($x >= 20) AND ($x % 2 == 0)) //instructions
```



## B. *switch*

L'instruction *switch* est une structure conditionnelle permettant de faire plusieurs tests de valeurs sur le contenu d'une même variable.

Exemple de syntaxe :

```
switch ($variable) {  
    case 1 : //instructions 1  
        break;  
    case 2 : //instructions 2  
        break;  
    case 3 : //instructions 3  
        break;  
    default : //instructions par défaut  
}
```

Les parenthèses qui suivent le mot clé ***switch*** contiennent une variable dont la valeur est testée successivement par chaque ***case***. Lorsque la variable testée est égale à une des valeurs suivant un *case*, la liste d'instructions qui suit est exécutée.

Le mot clé ***break*** indique la sortie de la structure. Il peut également être utilisé dans une structure itérative pour « casser » la répétition.

Le mot clé ***default*** précède les instructions qui seront exécutées si l'expression n'est égale à aucune des valeurs proposées dans les *case*.

### C. *while*

```
while (/*condition*/) {  
    //instructions  
}
```

Cette structure itérative exécute les instructions **tant que** (*while* signifie « *tant que* ») la condition est vraie.

Le risque est que la condition reste vraie... toujours ! On appelle cela une **boucle infinie**, c'est-à-dire un plantage du script qui n'arrive jamais à sortir de la boucle.

#### Exemple

```
$x=82;  
while ($x%2==0) {  
    $x/=2;  
}
```

#### Traduction

\$x vaut 82.  
Tant que le reste de la division de \$x par 2 vaut 0 :  
diviser \$x par 2.

### D. *do while*

```
do {  
    //instructions  
} while (/*condition*/);
```

La structure itérative **do while** est semblable à **while** à ceci près que la condition se trouve en fin de boucle au lieu du début. Le test de la condition ayant lieu après les instructions, celles-ci sont toujours exécutées au moins une fois.

Cette boucle convient lorsque la condition porte sur un élément n'étant pas initialisé avant la boucle (comme l'élément n'a pas de valeur, il est impossible de l'utiliser).

#### Exemple

```
do {  
    $x=rand(1,20);  
} while (in_array($x,$tab));
```

#### Traduction

Répéter : Affecter à \$x une valeur aléatoire entre 1 et 20 tant que la valeur de \$x est comprise dans \$tab

## E. *for*

```
for (/*affectation*/ ; /*condition*/ ; /*progression*/) {
    //instructions;
}
```

La troisième structure itérative est l'instruction *for*. Comme pour l'instruction *while*, la condition se fait avant les instructions, mais cette fois avec une écriture condensée comprenant l'affectation du compteur, la condition et la progression (incrémentation par exemple) du compteur.

Exemple	Traduction
<pre><code>for (\$i=0 ; \$i&lt;=10 ; \$i++) {     echo \$i; }</code></pre>	Pour \$i allant de 0 à 10 en progressant de 1 : générer \$i.

## F. *foreach*

L'instruction *foreach* permet de répéter des instructions **pour chaque** élément d'un tableau.

L'avantage de cette boucle est qu'elle ne nécessite pas de connaître le nombre d'éléments du tableau ni la valeur des indices. La boucle parcourt tout le tableau du début jusqu'à la fin et exécutera autant de fois les instructions qu'il y a d'éléments dans le tableau.

Dans l'exemple ci-dessous, le mot-clé **as** permet de transposer dans *\$valeur*, les valeurs trouvées dans le tableau *\$\_SERVER*.

```
foreach ($_SERVER as $valeur) {
    echo '<p> $valeur';
}
```

Version améliorée permettant de récupérer l'indice (*\$key*) et la valeur :

```
foreach ($tableau as $key=>$val) {
    echo '<p>', $key, '=', $val;
}
```

Exemple	Traduction
<pre><code>foreach(\$trucs as \$x) {     echo \$x; }</code></pre>	Pour chaque valeur de \$trucs transposée dans \$x : générer \$x.

**Exercice (papier) : que génèrent exactement les scripts suivants ?**

1.

```
$prix = 48.85 ;  
$prix = (integer) $a ;  
echo "prix : $prix" ;
```

2.

```
$poids=48.2 ;  
if ( is_double($poids) ) {  
    echo $poid ;  
}
```

3.

```
$age=20 ;  
if (($age>=18) && ($age<=60)) {  
    echo 'Welcome' ;  
}
```

4.

```
for ($i=5 ; $i>0 ; $i--) {  
    echo $i ;  
}
```

5.

```
$x=0 ;  
for ($i=0 ; $i<5 ; $i++) {  
    $x+=$i ;  
}  
echo $x ;
```

6.

```
$i=0 ;  
for ($i=5 ; $i<=5 ; $i++) {  
    echo $i ;  
}  
echo $i ;
```

7.

```
$tab[0]='ouf' ;  
if ( !empty($tab[0]) ) {  
    echo $tab[0] ;  
}  
else echo 'ko' ;
```

8.

```
$tab[0]='ouf';
if ( isset($tab[0]) ) {
    echo '$tab[0]' ;
}
else echo 'ok';
```

10.

```
$tab[0]=7;
$tab[]=4;
$tab[9]=9;
$tab[]=2;
foreach ($tab as $x) {
    echo $x;
}
```

11.

```
$i=1 ;
$x="valeur : ";
do {
    $i+=1;
    $x.=$i;
} while ($i<3);
echo $x;
```

### Exercice (papier) : boucles et conditions

Soit un tableau `$noms[]` contenant les noms des étudiants d'une classe et un tableau `$localites[]` contenant les adresses des étudiants d'une classe (les indices des deux tableaux concordent par étudiants). Affectez à une variable `$nb_etudiants` le nombre d'étudiants en utilisant la fonction `count()`.

```
$noms = array('Bob', 'Kévin', 'Julien', 'Brad', 'Timmy');
$localites = array('7000 Mons', '7012 Jemappes', '7050 Jurbise',
'', '7000 Mons');
$nb_etudiants = ...
```

À l'aide d'une boucle `foreach()`, générez le résultat suivant (utilisation de `<ul>` et `<li>`) :

- Bob, localité : 7000 Mons
- Kévin, localité : 7012 Jemappes
- Julien, localité : 7050 Jurbise
- Brad, localité : inconnue
- Timmy, localité : 7000 Mons

Attention aux localités vides qui donnent lieu à « inconnue ».

## Exercice 5 : Boucles

**Exemple : script générant un tableau à 2 dimensions de 4 à 10 tirages de 6 numéros pour le lotto.**

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Tirages du Lotto</title>
</head>
<body>
  <h1>Tirages du Lotto</h1>
  <?php
    $nb_tirages = rand(4,10) ;
    for ($tirage=1 ; $tirage<=$nb_tirages ; $tirage++) {
      for ($i=0 ; $i<6 ; $i++) {
        $numeros[$tirage][$i] = rand(1,50);
      }
    }
  ?>
</body>
</html>
```

### Exercice :

- A la suite de ce script, et en utilisant des boucles *foreach()*, affichez les résultats dans un <table>.
- Complétez le script ci-dessus, de façon qu'à chaque tirage, après avoir pioché les 6 numéros, le tirage soit trié avec la fonction *sort()*.

15	20	24	31	36	50
5	20	21	23	38	43
13	16	18	22	48	49
10	35	35	36	38	49
29	32	39	42	43	46
7	8	13	14	39	44
7	21	25	26	31	39

## Exercice 6 : Générateur de noms de nains

**Exemple : ce script incomplet comporte 3 tableaux de chaînes de caractères**

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Générateur de noms de nains</title>
  <style>
    body { font-family:AR JULIAN, Arial; }
  </style>
</head>
<body>
  <h1>Noms de nains</h1>
  <?php
    $nain_debut=array('A','Ba','Bo','Bu','Bra','Bre','Bro','Da','Dra','D
    ro','Du','Ga','Go','Gu','Gra','Gri','Gro','I','Ka','Ko','Ku','O','U'
    ,'Ta','Ti','To','Tu');
    $nain_liaison=array('ka','ko','la','lo','ra','rba','ro','rbo');
    $nain_fin=array('ban','dar','dir','dor','dur','gal','gan','gar','gor
    ','grim','gur','kan','lan','lar','lek','li','lin','lion','lir','rak'
    ,'ran','rek','rgrim','rgor','rik','ril','rion','rok','ron','tar','tr
    ek','tron');
  ?>
</body>
</html>
```

**Exercice :** complétez ce script PHP afin qu'il génère aléatoirement 20 noms de nains. Les noms de nains sont composés de 2 ou 3 syllabes piochées aléatoirement et concaténées pour former un nom.



1. Initialisez 3 nouvelles variables avec le nombre de valeurs de chaque tableau. Pour cela, utilisez la fonction **count()**.
2. Dans une boucle de 20 itérations :
  - a. piochez aléatoirement (fonction **rand()**) une syllabe de début et enregistrez-la dans un tableau \$noms[ ].
  - b. piochez un nombre au hasard de façon à ce qu'il y ait 40% de chances qu'une syllabe de liaison soit piochée et concaténée à la première syllabe.
  - c. pour finir, piochez une syllabe de fin et concaténez-la au reste.
3. Triez le tableau de noms par ordre alphabétique avec la fonction **sort()**.
4. Dans une nouvelle boucle (*foreach* cette fois), générez le code HTML avec les 20 noms de nains dans une liste <ul>.
5. Accordez à chaque élément de liste, une classe « couleur0 » ou « couleur1 » grâce à l'opération modulo. Ensuite en CSS, imposez 2 couleurs de texte différentes.
6. Rectifiez la boucle du point 2, afin que chaque nom soit différent. Pour cela, utilisez une boucle *do while()* et la fonction **in\_array()**.

## 1.6 Variables prédéfinies

### A. Variables prédéfinies

Les variables prédéfinies (ou variables superglobales) sont des variables de type tableau associatif, qui sont prédéclarées : elles existent nativement dans toutes vos pages PHP sans que vous n'ayez à les déclarer et ont des rôles particuliers.

Voici les principales variables prédéfinies :

<code>\$GLOBALS</code>	Contient le nom et la valeur de toutes les variables globales du script. Les noms des variables sont les clés du tableau (exemple : <b><code>\$GLOBALS["var"]</code></b> ).
<code>\$_COOKIE</code>	Contient le nom et la valeur des cookies enregistrés sur le poste client.
<code>\$_ENV</code>	Contient le nom et la valeur des variables d'environnement qui sont changeantes selon les serveurs.
<code>\$_FILES</code>	Contient le nom des fichiers téléchargés à partir du poste client.
<code>\$_GET</code>	Contient le nom et la valeur des données issues de l'URL (ou d'un formulaire envoyé par la méthode GET).
<code>\$_POST</code>	Contient le nom et la valeur des données issues d'un formulaire envoyé par la méthode POST.
<code>\$_REQUEST</code>	Contient les noms des variables prédéfinies <b><code>\$_GET</code></b> , <b><code>\$_POST</code></b> , <b><code>\$_COOKIE</code></b> , <b><code>\$_FILES</code></b> .
<code>\$_SERVER</code>	Contient les informations relatives au serveur Web, comme la langue du client ( <code>HTTP_ACCEPT_LANGUAGE</code> ), l'adresse du serveur ( <code>SERVER_ADDR</code> ) ou le nom du script en cours d'exécution ( <code>PHP_SELF</code> ).
<code>\$_SESSION</code>	Contient l'ensemble des noms et valeurs des variables de session, enregistrées sur le serveur et identifiées via un cookie de session. Lorsque vous vous connectez sur un site, c'est grâce à cette variable de session que votre connexion est effective sur toutes les pages du site.



### Exemples d'utilisation des variables prédéfinies :

Soit une variable quelconque `$x`, les deux instructions suivantes permettent d'afficher sa valeur :

```
echo $GLOBALS['x'];
echo $x;
```

Si vous souhaitez afficher l'entièreté du contenu d'un de ces tableaux prédéfinis, voici une solution à l'aide d'une boucle *while* combinée à un *each()* :

```
reset($GLOBALS);
while ( $gl=each($GLOBALS) ){
    echo 'Élément ', $gl[0], ' => ', $gl[1];
}
```

**Exercice (papier) :** Comment afficher l'adresse IP du serveur, la langue du navigateur client et le nom du script en cours d'utilisation ?

## B. GET et POST

Les variables prédéfinies GET et POST permettent de passer des paramètres (noms et valeurs) via la requête HTTP.

Alors que les paramètres passés en GET sont visibles dans l'URL, les paramètres passés en POST sont dissimulés aux yeux du visiteur. Ceci ne les rend pas forcément plus sécurisés : méfiez-vous toujours des données récupérées en GET ou en POST.

```
<form action="..." method="post">
    <input type="text" name="pseudo">
    <input type="checkbox" name="heureux" value="oui">
</form>
```

Dans le cas d'un **formulaire**, ici en méthode POST, les paramètres envoyés sont des paires « nom=valeur », correspondant aux attributs *name* et *value* des champs du formulaire. Alors que le champ texte (comportement classique) est toujours envoyé, même si sa valeur est vide, les champs *checkbox* et *radio* ne sont envoyés que si leur case est cochée par l'utilisateur.

`index.php?x=32&nom=Baroud`

Lors du passage de paramètres en GET, le point d'interrogation marque la séparation entre l'URL et les paramètres. Les paramètres sont des paires « nom=valeur » séparées par le symbole « & ».


Notez que dans le code HTML, il est demandé de remplacer les « & » par « &amp; ».

## Exercice 7 : Paramètres GET

**Exemple : script testant l'existence d'un paramètre GET dans l'URL et récupérant sa valeur**

```
<?php
//numéro de page par défaut, au cas où absence de GET
$page = 1 ;
//récupération du paramètre GET s'il existe et conversion
(int) pour le sécuriser
if (isset($_GET['page'])) $page = (int) $_GET['page'] ;

?><!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Paramètres GET</title>
</head>
<body>
    <h1>Page <?php echo $page ; ?>/3</h1>
    <nav>
        <ul>
            <li><a href="index.php?page=1">Page 1</a>
            <li><a href="index.php?page=2">Page 2</a>
            <li><a href="index.php?page=3">Page 3</a>
        </ul>
    </nav>
</body>
</html>
```



### Exercice :

Les paramètres GET sont facilement modifiables par les utilisateurs. À partir du script ci-dessus :

- Ajoutez un test sur la valeur du paramètre : si celle-ci est plus petite que 1 ou plus grande que 3, codez une redirection vers la page 1. Testez l'efficacité de votre code en modifiant le paramètre GET directement dans le champ URL de votre navigateur, en essayant d'accéder aux pages 88, -15 et 0.
- Ajoutez un lien à la navigation : « Page suivante » redirigeant le visiteur vers la page suivante (de 1 vers 2 ; de 2 vers 3 et de 3... vers 1 !)
- Ajoutez un lien à la navigation : « Page précédente » redirigeant le visiteur vers la page précédente (de 3 vers 2 ; de 2 vers 1 et de 1... vers 3 !)
- Ajoutez du CSS interne (<style>) pour que la couleur du texte dépende de la page visitée : texte en rouge sur la page 1, en bleu sur la page 2 et en vert sur la page 3.



## C. Sessions

La variable prédéfinie `$_SESSION[ ]` permet de stocker des valeurs qui perdurent d'une page à l'autre d'un même domaine. Ainsi, le visiteur connecté peut bénéficier de sa connexion sur toutes les pages du site, sans devoir s'identifier sur chacune. Vous pouvez stocker dans la session le nom du visiteur, son email, son rôle, etc.

Comme les autres variables prédéfinies, `$_SESSION[ ]` existe par défaut sur toutes nos pages PHP (bien que vide au départ). Le contenu de cette variable de session est stocké temporairement côté serveur et lié au client par un identifiant enregistré dans un cookie de session. Cette liaison avec la session doit être débutée par l'instruction **`session_start()`** qui doit obligatoirement précéder le `<!DOCTYPE html>` :

```
<?php session_start(); ?>
<!DOCTYPE html>
```

Remarque : le choix du format UTF-8 avec BOM peut bloquer l'utilisation de session.

Pour entrer une valeur dans la variable de session, il suffit de l'affecter comme suit :

```
$_SESSION['administrateur']=1 ;
```

Ensuite, le contenu de vos pages peut dépendre des valeurs de session, par exemple lorsqu'une page est réservée aux administrateurs, ou pour une fonctionnalité réservée à certains utilisateurs. Pour cela, il suffit de tester certaines valeurs en session. Par exemple :

```
<?php
if (isset($_SESSION['nom']))
    echo '<p>Bienvenue', $_SESSION['nom'];
else echo '<p>Vous n'êtes pas connecté.';
?>
```

Ou encore :

```
<?php
if(!empty($_SESSION['admin'])) {
    echo '<p>Vous êtes connecté en administrateur.';
}
else{
    header('Location:index.php'); exit() ;
}
?>
```

Les variables de session sont donc particulièrement employées dans le cas des comptes utilisateurs, mais aussi pour stocker le contenu d'un panier sur un site d'e-commerce, ou simplement pour stocker l'une ou l'autre préférence de l'utilisateur, comme la langue du site. Les données ainsi enregistrées seront accessibles jusqu'à la fermeture du navigateur ou la destruction de la session. Il vous est possible de forcer la destruction de cette session, par exemple afin de déconnecter le visiteur :

```
<?php session_destroy(); ?>
```

## D. Cookies

La variable `$_COOKIE` permet de stocker des valeurs sur le long terme, sur le poste client. Par exemple, sur votre site d'e-commerce, vous souhaitez que le visiteur abandonnant ses achats retrouve son panier lors de sa prochaine visite. Ou encore, vous souhaitez pré-compléter un formulaire avec les valeurs entrées lors d'une visite précédente.

**Remarque :** Notez qu'une base de données permet également de stocker des données sur le long terme. Mais cette solution surcharge votre serveur au lieu du poste client, ce qui n'est pas négligeable si vous avez des milliers de visiteurs. De plus, le cookie peut être enregistré sans authentification du visiteur, alors que les données en BD propres à un visiteur doivent faire l'objet d'une authentification.

Les cookies ont tendance à avoir mauvaise réputation. À l'origine créés pour soutenir l'expérience utilisateur, ils sont de plus en plus souvent utilisés pour y stocker des informations sur vos activités en ligne... et les mettre à disposition d'autres domaines. Pourtant, le cookie n'est pas dangereux à proprement parler : n'allez pas confondre cookie et virus.

Chaque cookie a un nom et une date d'expiration. La loi exige que la durée de vie des cookies soit limitée dans le temps (max. 13 mois en France).

```
<?php
$value = 'RicHunter' ;
setcookie('pseudo', $value, time()+3600) ;
?>
```

Voici l'écriture d'un cookie dont le nom « pseudo » est associé à la valeur « RicHunter ». La date d'expiration du cookie est fixée dans une heure (*timestamp* actuel + 3600s).

```
<?php
echo $_COOKIE['pseudo'];
?>
```

Et l'utilisation de ce paramètre stocké sous forme de cookie

## E. Cookies et RGPD

Depuis 2018, le **RGPD** (Règlement général sur la protection des données) oblige les sites web à avertir ou à demander l'autorisation aux visiteurs de l'utilisation de cookies lorsque ceux-ci permettent d'identifier un individu. Oui, c'est le début de ces horribles pop-ups qui polluent actuellement la quasi-totalité des sites web...

Les cookies fonctionnels échappent à cette réglementation. Par exemple :

- Les cookies de session ;
- Les cookies persistant quelques heures après la durée de la session pour conserver temporairement un panier d'achat ou pour les valeurs de formulaires ;
- ...

Les **cookies concernés** sont principalement :

- Les cookies liés à la publicité ;
- Les cookies de réseaux sociaux ;
- Certains cookies de mesure d'audience collectant plus d'informations.

Il y a lieu de distinguer les **cookies propriétaires** des **cookies tiers**. Alors que les cookies propriétaires sont déposés par un site pour une utilisation exclusivement réservée à ce site, les cookies tiers sont mis à disposition d'autres sites. La finalité des cookies tiers est souvent d'ordre commercial : vous visitez un site e-commerce et les articles visualisés vous poursuivent dans des annonces commerciales sur d'autres sites.

Récemment, les navigateurs ouvrent la chasse aux cookies tiers. Firefox et Safari bloquent dorénavant par défaut les cookies tiers. Chrome a annoncé sa probable intention de suivre cette décision. Les pratiques d'e-marketing vont devoir prendre un nouveau virage dans les années à venir...

Dans le cas du placement d'un **pop-up RGPD**, tant que la personne n'a pas donné son consentement, les cookies ne peuvent pas être enregistrés ou lus sur son terminal. L'internaute doit être informé par l'apparition d'un bandeau avec les obligations suivantes :

- Les finalités précises des cookies utilisés ;
- La possibilité de s'opposer à ces cookies et de changer les paramètres en cliquant sur un lien « en savoir plus et paramétrer les cookies » ;
- Poursuivre la navigation sans interagir avec le popup ne vaut plus pour consentement depuis 2020 ;

La mise en place d'un « **cookie wall** » est une pratique visant à empêcher l'accès à un site web ou à une application mobile aux visiteurs refusant les cookies non fonctionnels. Cette pratique est interdite par le RGPD.

## 1.7 Fonctions

### A. Déclarer et utiliser des fonctions

Voici une fonction nommée « somme », prenant 2 paramètres et renvoyant un résultat que l'on stocke dans une variable nommée « \$somme » :

```
<?php
function somme ($param1, $param2) {
    $result = $param1 + $param2;
    return $result ;
}
$somme = somme (15, 2) ;
?>
```

Remarques :

- Les paramètres et l'instruction *return* ne sont pas obligatoires.
- Dans cet exemple, la variable *\$result* est locale à la fonction.

L'exemple suivant montre comment utiliser des **variables globales** dans une fonction à l'aide du mot-clé *global* :

```
<?php
$a = 1;
$b = 2;
function pythagore () {
    global $a, $b;
    $result = sqrt ($a*$a + $b*$b);
    return $result ;
}
echo pythagore ();
?>
```

L'exemple suivant montre comment coder un paramètre optionnel (le paramètre \$triangle possède une valeur par défaut et n'est pas passé à chaque appel) :

```
<?php
function surface ($param1, $param2, $triangle=False) {
    if ($triangle==True) return $param1 * $param2 / 2 ;
    else return $param1 * $param2;
}

$aire_rectangle = surface (3, 4);
$aire_triangle = surface (3, 4, True);
?>
```

## B. Quelques fonctions mathématiques

<code>abs (\$x)</code>	Renvoie la valeur absolue de \$x
<code>ceil (\$x)</code>	Renvoie l'arrondi au supérieur
<code>dechex (\$x)</code>	Convertit de décimal vers hexadécimal
<code>floor (\$x)</code>	Renvoie l'arrondi à l'inférieur
<code>hexdec (\$x)</code>	Convertit d'hexadécimal vers décimal
<code>hypot (\$x, \$y)</code>	Renvoie la longueur de l'hypoténuse d'un triangle rectangle : $\sqrt{x^2 + y^2}$
<code>max (\$x, \$y, \$z)</code> <code>max (\$tab)</code>	Renvoie le maximum
<code>min (\$x, \$y, \$z)</code> <code>min (\$tab)</code>	Renvoie le minimum
<code>pi ()</code>	Renvoie la valeur de Pi
<code>pow (\$x, \$y)</code>	Renvoie \$x exposant \$y
<code>rand (\$x, \$y)</code>	Renvoie un nombre entier aléatoire entre \$x et \$y
<code>round (\$x)</code>	Renvoie l'arrondi au plus proche
<code>round (\$x, \$y)</code>	Renvoie l'arrondi (avec \$y décimales) au plus proche

Autres fonctions mathématiques : <http://php.net/manual/fr/ref.math.php>

## 1.8 Traitements des formulaires

### A. Paramétrer les balises *form*

```
<form action="form.php" method="post"> ... </form>
```

#### Quelle méthode utiliser : GET ou POST ?

GET et POST sont les 2 méthodes d'envoi de données au serveur.

POST dissimule les données du formulaire, alors que GET les affiche dans l'URL. À savoir que si les données sont contenues dans l'URL, le visiteur peut les lire et les modifier.

On opte donc pour la méthode POST lorsqu'il s'agit d'une inscription, d'une commande, d'un message, ou de tout autre formulaire demandant un certain niveau de discrétion. On choisit la méthode GET pour les formulaires de recherche où l'on permet au visiteur de modifier à sa guise les données.

#### Comment choisir la destination du formulaire ?

L'attribut **action** de la balise **<form>** précise l'url de la page ou du script qui sera appelé après la validation du formulaire. Ceci dépend de l'architecture de votre site et des besoins rencontrés.

Appeler le fichier contenant le formulaire est une bonne habitude car cela permet de traiter le formulaire sur la même page que son affichage : si le formulaire est mal complété, il sera aisé de le réafficher en évitant de perdre les données correctes et en signalant les données incorrectes.

### B. Récupérer les données du formulaire

Les données transmises par la soumission d'un formulaire sont récupérables en PHP via la variable prédéfinie **\$\_GET** ou **\$\_POST** correspondant à la méthode choisie. Ces deux variables sont des tableaux associatifs dont les indices correspondent aux attributs **name** et dont les valeurs sont les attributs **value** des champs de formulaire.

**Exemple 1** : le champ texte ci-dessous attend le nom de la personne. Lorsque le visiteur modifie la valeur du champ, il en modifie l'attribut *value*.

**Nom** : `<input type="text" name="pseudo" size="20">`

Après soumission du formulaire, on obtiendra le nom de cette personne en utilisant une des variables suivantes :

```
$_GET['pseudo'] //si on utilise la méthode GET  
$_POST['pseudo'] //si on utilise la méthode POST
```



**Exemple 2 :** les **listes de sélection** contiennent des options présentant des attributs *value*. Lorsque le visiteur sélectionne une option de la liste, celle-ci passe *selected*. La *value* de la liste est la *value* de l'option *selected*. La soumission du formulaire enverra cette paire *name/value*.

```
<select name="diplome">
    <option value="bac" selected> Bachelier
    <option value="mas"> Master
</select>
```

**Exemple 3 :** les **champs multilignes** ne possèdent pas d'attribut *value*. Lorsque le visiteur y entre du texte, il en modifie le contenu. La valeur envoyée au serveur est donc le contenu de cette balise *textarea*.

```
<textarea name="commentaire">Patati patata...</textarea>
```

**Exemple 4 :** les boutons **radio** et **checkbox** présentent une *value* déjà codée en HTML. Lorsque le visiteur coche une case, c'est l'attribut *checked* qui est modifié. Lors de la soumission du formulaire, seuls les paires *name/value* des cases cochées sont envoyées au serveur.

```
<input type="radio" name="couple" value="oui"> Oui
<input type="radio" name="couple" value="non"> Non
```

**Exemple 5 :** pour les champs ramenant plusieurs valeurs comme les cases à cocher **checkbox**, le nom est suivi d'une paire de crochets indiquant qu'un tableau de valeurs est envoyé au serveur.

```
<input type="checkbox" name="sports[]" value="foot">
<input type="checkbox" name="sports[]" value="judo">
```

Côté serveur, nous exploiterons un tableau de valeurs.

```
foreach ( $_POST['sports'] as $sport ) { ... }
```

## C. Tester l'existence des données du formulaire

Attention, ces variables `$_POST` et `$_GET` n'existent qu'après **soumission du formulaire**. Il est donc nécessaire de douter de leur existence en la testant systématiquement avant leur manipulation.

Les fonction **isset()**, **empty()** répondent parfaitement à ce besoin :

```
if (isset($_POST["rue"]))

if (empty($_GET['recherche']))
```

Pour contrôler avec plus de précision les données provenant du formulaire, il est recommandé d'utiliser les **motifs** (voir suite du cours). Les motifs permettent par exemple de savoir si le nom n'est composé que de lettres, si le code postal n'est composé que de chiffres, si l'adresse email contient bien le symbole @, etc.

## D. Exemple pratique

```
<?php
$form['sexe']='H' ; //Valeur par défaut du champ 'sexe'
//On teste si le formulaire a été envoyé
if (isset($_POST['submitted'])) {
    $form_ok=true;
    //On teste toutes les données une à une (le nom, l'email, etc.)
    //Pour le champ 'sexe', l'utilisateur peut choisir H ou F, mais on doit
    vérifier s'il n'a pas entré d'autres valeurs
    if ($_POST['sexe']=='H' or $_POST['sexe']=='F') {
        $form['sexe']=$_POST['sexe'];
    }
    else {
        $form_ok=false;
    }
    //Après avoir testé tous les champs, on teste si le formulaire est
    toujours ok
    if ($form_ok==true) {
        //Ici, les données envoyées sont acceptées. On peut par exemple les
        enregistrer en BD ou les envoyer par email...

        //Ensuite, on redirige vers une autre page. Ceci évite que les
        données s'enregistrent plusieurs fois lorsque le visiteur rafraîchit la
        page... On peut même rafraîchir vers la même page : le traitement n'aura plus
        lieu car cette nouvelle requête http ne contient pas de paramètre POST
        header('Location:index.php'); exit() ;
    }
}
?><!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Formulaire PHP</title>
</head>
<body>
<?php
//On n'affiche pas le formulaire si la soumission est correcte
if (empty($form_ok)) {
?>
<form action="index.php" method="post">
    <fieldset><legend>Identité</legend>
    <p><input type="radio" value="H" name="sexe" <?php if
($form['sexe']=='H') echo 'checked'; ?>>Homme
    <p><input type="radio" value="F" name="sexe" <?php if
($form['sexe']=='F') echo 'checked'; ?>>Femme
    <input type="submit" name="submitted" value="Envoyer">
    </fieldset>
</form>
<?php } ?>
</body>
</html>
```

Dans cet exemple, la page commence par un test en PHP, pour déterminer si le formulaire a été envoyé ou non. Les deux cas sont effectivement possibles : soit le visiteur arrive sur la page pour la première fois (pas de données soumises), soit le visiteur a rempli et validé le formulaire... Dans les deux cas, c'est la même page qui est chargée !

## Exercice 8 : Formulaire de connexion

### Exemple :

```
<?php
session_start(); //démarrage de la session
//si le formulaire de connexion a été soumis...
if ( isset($_POST['connexion']) ) {
    //on réceptionne, on trime les chaînes et on hashe le mot de passe
    $login = trim($_POST['login']) ;
    $password = SHA1( trim($_POST['password']) ) ;

    //si le login et le mot de passe (bill) sont bons...
    if ( $login=='bill' and
$password=='c692d6a10598e0a801576fdd4ecf3c37e45bfb4' ) {
        $_SESSION['nom']='Bill'; //on enregistre en SESSION
        header('Location:index.php'); //on redirige pour vider $_POST
        exit();
    }
}
//si le formulaire de déconnexion a été soumis...
else if ( isset($_POST['deconnexion']) ) {
    session_destroy(); //on détruit la session
}
?><!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Formulaire de connexion</title>
</head>
<body>
    <?php
    if ( empty($_SESSION) ) { // utilisateur non connecté
        ?>
        <h1>Connexion</h1>
        <form method="post" action="index.php">
            <p><label for="login">Identifiant :<br>
                <input type="text" name="login" id="login">
            </label>
            <p><label for="password">Mot de passe :<br>
                <input type="password" name="password" id="password">
            </label>
            <p><input type="submit" name="connexion" value="Connexion">
        </form>
        <?php
    }
    else { // utilisateur connecté
        ?>
        <h1>Déconnexion</h1>
        <form method="post" action="form.php">
            <p><input type="submit" name="deconnexion" value="Déconnexion">
        </form>
        <?php
    }
    ?>
</body>
</html>
```

Cet exercice porte sur un formulaire de connexion avec session. Une fois connecté, l'utilisateur le reste jusqu'à la fin de la connexion : soit la fermeture du navigateur, soit l'appel de fonction `session_destroy()`.

À ce stade, seul Bill peut se connecter avec les identifiants bill / bill.

**Exercice :**

1. Essayez le formulaire en vous connectant avec les identifiants bill / bill, puis déconnectez-vous. Vous remarquerez que le formulaire de déconnexion doit être soumis 2 fois pour revenir sur le formulaire de connexion. Corrigez cela en codant une redirection vers la page « index.php » après la destruction de session.
2. Lors de la connexion de Bill,
  - a. Ajoutez un paramètre « titre » en session. Le titre de Bill est « Monsieur ».
  - b. Ajoutez un paramètre « admin » en session. Ce paramètre vaut True si l'utilisateur connecté est un admin. Bill est un admin.
3. Ajoutez la possibilité que Bob se connecte avec les identifiants bob / bob. Bob n'est pas admin. Son titre est « Monsieur ».
4. Ajoutez la possibilité que Betty se connecte avec les identifiants betty / betty. Betty est admin. Son titre est « Madame ».
5. Juste sous le titre <h1>Déconnexion</h1>, si l'utilisateur connecté est admin, affichez « Bonjour Monsieur » ou « Bonjour Madame ».

## E. Réafficher les données du formulaire

Pour que le formulaire ne perde pas les données entrées, on va jouer sur :

- L'attribut **value** des champs de saisie

```
<input type="text" name="nom" value="<?= $nom ?>">
```

- L'attribut **checked** des cases à cocher

```
<input type="radio" name="tarif" value="senior" <?= ($tarif=='senior')?'checked':'' ?>> Senior
```

- L'attribut **selected** des listes de sélection
- Le contenu des champs multilignes

## F. Envoi de fichier



Exemple de code PHP, lors de la réception d'un formulaire, testant l'extension et le poids d'un fichier afin de l'écrire sur le serveur.

```
define('MAX_FILE_UPLOAD', 10000000); // octets
/*UPLOAD de fichier */
if (!empty($_FILES['fichier']['name'])) {
    /* Test extension */
    $extensions_permises = array('.pdf', '.doc', '.docx', '.txt', '.odf',
    '.rdf', '.png', '.gif', '.jpg', '.jpeg', '.svg', '.webp');
    $extension = strrchr($_FILES['fichier']['name'], '.');
    if (!in_array($extension, $extensions_permises)) {
        $erreur['fichier'] = 'Extension de fichier interdite : '.$extension;
    }
    /* Test Poids */
    elseif ( filesize($_FILES['fichier']['tmp_name'])>MAX_FILE_UPLOAD ) {
        $erreur['fichier'] = 'Fichier trop lourd :
    '.$_FILES['fichier']['size'].' > limite '.MAX_FILE_UPLOAD;
    }
    /* C'est bon, écriture du fichier sur le serveur */
    else {
        $tempFile = $_FILES['fichier']['tmp_name'];
        $targetFile = $_SERVER["DOCUMENT_ROOT"].'/'.$_FILES['fichier']['name'];
        $targetFile = '/dossier/'.$_FILES['fichier']['name'];
        if ( move_uploaded_file($tempFile,$targetFile) ) {
            $nom_fichier = $_FILES['fichier']['name'];
        }
        else $erreur['fichier'] = "Erreur upload : $tempFile => $targetFile";
    }
}
}
```

Extrait du formulaire :

```
<form method="post" action="..." enctype="multipart/form-data">
...
<p>Fichier : <?= $nom_fichier; ?></p>
<label class="<?php if (!empty($erreur['fichier'])) echo 'error'; ?>"
for="fichier">Changer de fichier ?
    <input type="file" id="fichier" name="fichier">
</label>
<?php
if (!empty($erreur['fichier'])) {
    echo '<small class="error">',$erreur['fichier'],'</small>';
}
?>
...
</form>
```

Cet exemple de code constitue une base pour la réception de fichiers via un formulaire. Dans la pratique, de nombreux tests seront nécessaires : sur les noms de fichiers avec espaces, accents ou caractères spéciaux par exemple ; sur les extensions de fichiers en majuscules ; ou encore sur les noms de fichiers déjà présents sur le serveur (écraser ? changer le nom ?) ...

## 1.9 Sécurité

### A. Introduction

Il est difficile de faire le tour de toutes les failles de sécurité du Web : elles sont nombreuses, dépendent des technologies, logiciels et langages utilisés et évoluent au fil du temps.

Ce chapitre porte exclusivement sur les failles de sécurité liées au développement web, c'est-à-dire les attaques par injection de code.

### B. Les attaques par injection

Les attaques par injection sont des tentatives malicieuses d'entrer du code (dans différents langages) par une des portes d'entrée du site afin que ce code soit interprété.

#### Les trous

Un site web PHP peut comporter des failles de sécurité aux attaques par **injections** de code, aux endroits suivants :

1. Les formulaires (GET ou POST)
2. Les paramètres dans les URL (GET) : `www.exemple.com/index.php?p=...`
3. L'AJAX (GET ou POST)
4. Les cookies (COOKIE)

#### Les langages

Les **attaques par injection** peuvent utiliser différentes technologies : HTML, **SQL**, XPATH, XSS (*cross-site scripting*), LDAP, etc.

Voici un code comportant une énorme faille de sécurité :

```
$result=mysql_query("SELECT * FROM utilisateur WHERE  
nom='".$$_POST["login"]."' and password=  
'".$_POST["password"]."'");
```



Devinez l'effet de cette requête lorsqu'un visiteur entrera ceci dans le formulaire :

```
' OR 1=1 --
```

(Le code `--` met en commentaire la suite de la requête SQL.)

Ou pire encore :

```
' ; DROP TABLE utilisateur ; --
```

## Les parades

1. **Utiliser au minimum les variables potentiellement dangereuses `$_GET` et `$_POST`** : que ce soit dans un *echo*, ou dans une requête SQL, il est toujours dangereux de manipuler ces variables. C'est pourquoi, il est de bon usage de ne les utiliser qu'une seule fois : en début de code, lorsqu'on les sécurise et que l'on transpose leur valeur dans d'autres variables.

2. **Forcer la conversion en entier ou en réel** : si les données transmises sont des nombres, une conversion en nombre empêche l'injection de code.

```
$id = (int) $_GET['id'] ;
```

3. **Hasher les données** : hasher un mot de passe le rend illisible et indécryptable mais a aussi pour effet de le sécuriser puisque le résultat ne contiendra que des caractères sûrs.

```
$password = sha1( $_POST['password'] ) ;
```

4. **Échapper les caractères spéciaux** : avec *htmlspecialchars()* ou *htmlentities()* ou votre propre fonction d'échappement. Méfiez-vous des caractères : `> < + - = ( ) [ ] ' " ; , . { } : / \` car ils sont utiles à la plupart des injections.

```
$nom = htmlspecialchars( $_GET['nom'] ) ;
```

5. **Échapper les balises HTML** : avec *strip\_tags()*. Cela empêchera vos visiteurs d'injecter du code HTML dans les formulaires comme des `<iframe>` ou des `<a>`.

```
$nom = strip_tags( $_POST['commentaire'] ) ;
```

6. **Valider les données avec une liste blanche ou les exclure avec une liste noire** : lorsque vous connaissez toutes les valeurs possibles, vous pouvez valider via une liste blanche ; au contraire, si vous connaissez des valeurs interdites, vous pouvez les exclure via une liste noire.

```
$liste_blanche = array('be','de','fr','lu','se') ;
if ( isset($liste_blanche[$_GET['extension']]) ) {
    $extension = $_GET['extension'] ;
}
```

7. **Tester les données avec des motifs** : les motifs (appelés aussi expressions régulières ou *regex*) sont l'outil de test de chaînes de caractères le plus précis qui soit. Comme pour les listes, ils peuvent être utilisés pour valider ou pour exclure des données.

```
$regex_nom = '/[";:!?(){}<>=+]/';
if (!preg_match($regex_nom,$_POST['nom'])) {
    $nom = $_POST['nom'] ;
}
```

8. **Utiliser des requêtes préparées (PDO) ou des procédures stockées** : l'utilisation correcte de requêtes préparées vous protège de toute injection SQL.

Les mesures suivantes sont intéressantes mais seront hélas facilement déjouées par les pirates car elles agissent uniquement du côté du client :

9. Limiter le nombre de caractères dans les champs de formulaires (avec l'attribut *maxlength* par exemple)
10. Marquer des champs avec l'attribut *required* pour forcer l'utilisateur à les compléter
11. Tester les champs de formulaire avec du JavaScript
12. Utiliser les nouveaux champs de formulaire HTML5

### C. Les tests

Une bonne façon de tester un site est d'injecter dans toutes les entrées ce petit code :  
`<script>alert('faille') ;</script>`

Si vous obtenez le message d'alerte vous affichant le mot « faille », vous en avez trouvé une.

Pour pousser le test un peu plus loin, vous pouvez tenter de précéder votre code de différents caractères de fermeture : `> " '`



## D. À propos du hachage

Le hachage est l'application d'un algorithme transformant n'importe quelle chaîne de caractères en un résultat irréversible : une fois la chaîne hachée, il est impossible de la décrypter (ce que permet les algorithmes d'encryptage, à ne pas confondre).

Exemple :

Kilimandjaro : une chaîne

a1fe89572a2cc158e1be82d726aa27ee : sa version hachée avec l'algorithme MD5

Les algorithmes les plus fréquents sont :

- **MD5** : produit un hash de 128 bits ou 32 caractères (chiffres hexadécimaux)
- **SHA1** : produit un hash de 160 bits ou 40 caractères
- **SHA-256** : produit un hash de 256 bits ou 64 caractères
- **SHA-512** : produit un hash de 512 bits ou 128 caractères

```
<?php
$password_md5 = md5($password) ;
$password_sha1 = sha1($password) ;

$password_sha256 = hash('sha256', $password) ;
$password_sha512 = hash('sha512', $password) ;
?>
```

Les algorithmes de hachage sont très nombreux. Le *hacking* de mots de passe fait appel à des machines de plus en plus puissantes, c'est pourquoi les algorithmes de hachage donnent des résultats de plus en plus long et sont surtout volontairement plus lents à exécuter.

Actuellement, les algorithmes MD5 et SHA1 sont considérés comme insuffisants.

Une méthode fréquemment utilisée pour hacker les mots de passe consiste à confronter le mot de passe haché avec une **rainbow table** : un genre de dictionnaire de mots hachés. Ces dictionnaires sont évidemment non exhaustifs mais reprennent suffisamment de combinaisons pour déduire les mots de passe classiques ou des combinaisons de mots connus.

Pour freiner cette **attaque par force brute**, il est de bon usage de **saler** le mot de passe (le concaténer à une chaîne avant de le hacher), ou de bidouiller notre propre algorithme de hachage en partant des algorithmes existants.

Mieux vaut alors concaténer les empreintes que les combiner SHA1(MD5(SHA1('mot de passe')))) car cela augmente le nombre de **collisions** (2 chaînes de caractères qui donnent la même empreinte). En multipliant les collisions, vous augmentez le nombre de chances de trouver votre mot de passe.

## Exercice 9 : Formulaire avec gestion des erreurs du visiteur

## Exemple :

```

<?php
if ( isset($_POST['enregistrement']) ) {
    //un champ obligatoire
    if ( !empty($_POST['nom']) ) $nom = trim($_POST['nom']) ;
    else $erreurs['nom'] = true;

    //un champ obligatoire avec certaines valeurs rejetées
    if ( !empty($_POST['annee']) ) {
        if ( $_POST['annee']>date('Y') or $_POST['annee']<1800 ) {
            $erreurs['annee'] = true;
        }
        else {
            $annee = (int) $_POST['annee'];
        }
    }
    else $erreurs['annee'] = true;

    //une liste blanche pour tester les fruits
    $LBfruits = array('bananes', 'framboises', 'pommes');
    if ( isset($_POST['fruits']) ) {
        //les paramètres multiples arrivent sous forme de tableaux
        foreach ( $_POST['fruits'] as $f ) {
            $fruits[] = $f;
        }
    }

    //s'il n'y a pas d'erreur...
    if (empty($erreurs)) {
        //le formulaire est correctement complété
        $form_correct = True;
    }
}

?><!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Formulaire avec gestion des erreurs</title>
    <style>
        .red { color:red; }
    </style>
</head>
<body>
    <h1>Inscription</h1>
    <?php
    if ( isset($_POST['enregistrement']) ) {
        //Affichage des résultats du formulaire
        if ( !empty($form_correct) ) {
            echo "<p>Bonjour $nom, vous avez choisi : ";
            foreach ( $fruits as $f ) {
                echo $f, ' ';
            }
        }
    }
    ?>

    <form method="post" action="index.php">
        <p <?php if ( isset($erreurs['nom']) ) echo 'class="red"'; ?>>
            <label for="login">Nom :<br>
                <input type="text" name="nom" id="login" <?php if (isset($nom))
echo 'value=', $nom, ' '; ?> >

```

```

</label>
<p <?php if ( isset($erreurs['annee']) ) echo 'class="red"'; ?>>
<label for="annee">Année de naissance :<br>
<input type="number" step="1" name="annee" <?php if (isset($annee))
echo 'value="' . $annee . '"; ?> id="annee">
</label>

<ul>
<li><label for="choix1"><input type="checkbox" name="fruits[]"
value="bananes" id="choix1">Les bananes</label>
<li><label for="choix2"><input type="checkbox" name="fruits[]"
value="framboises" id="choix2">Les framboises</label>
<li><label for="choix3"><input type="checkbox" name="fruits[]"
value="pommes" id="choix3">Les pommes</label>
</ul>

<p><input type="submit" name="enregistrement" value="Enregistrer">
</form>
</body>
</html>

```

### Exercice :

1. À l'aide des outils de développement du navigateur, piratez (modifiez) l'interface HTML avec les outils du navigateur, afin d'envoyer la valeur « patates » par le formulaire. Constatez si la liste blanche fonctionne.
2. Avant d'affecter à `$fruits[]` les noms des fruits choisis, assurez-vous que chaque fruit est un fruit de la liste blanche.
3. Ajoutez un fruit de votre choix au formulaire et à la liste blanche. Testez le résultat.
4. Ajoutez un champ « pseudo ». Celui-ci n'est pas obligatoire : même incomplet, il ne provoque pas d'erreur, mais la valeur complétée par l'utilisateur doit réapparaître.
5. Ajoutez le choix du « titre » sous forme de deux boutons *radios* liés, ramenant la valeur « H » ou « F ». Lors du traitement du formulaire assurez-vous qu'aucune autre valeur n'est permise (y compris une valeur vide). Le bouton coché doit rester coché après envoi si une erreur est signalée dans le formulaire.
6. Ajoutez un champ « nationalité » sous forme de liste de sélection proposant 5 ou 6 pays. Ce champ peut être testé par liste blanche. Une erreur provoque l'affichage en rouge du champ. La valeur sélectionnée doit rester sélectionnée si une erreur est signalée dans le formulaire.
7. Ajoutez un bouton *checkbox* « J'ai lu et j'approuve les conditions d'utilisation. » ramenant la valeur « ok ». Lors du traitement du formulaire, assurez-vous que ce bouton a été coché, autrement c'est une erreur et le texte doit apparaître en rouge.

## 1.10 Regex (expressions régulières)

Les **regex**, appelées aussi **modèles**, **motifs**, **expressions rationnelles** ou **expressions régulières**, permettent d'exprimer avec précision la composition d'une chaîne de caractères.

Leur utilisation permet, par exemple dans le cas du traitement d'un formulaire, de s'assurer que le code postal est composé de 4 à 5 chiffres, ou que l'adresse mail contient un « @ » suivi d'un nom de domaine, suivi d'un « . », etc.

Autre exemple, l'utilisation de motifs permet de rechercher un mot ou une expression dans un texte.

Exemple de motif pour tester la validité d'un email :

```
'/^[[[:alpha:]]{1}([[:alnum:]]|[_.-]){1,}@{1}([[:alnum:]]{1,}|(.){1}){1,4}[[[:alpha:]]{2,3}$/'
```

L'écriture des motifs est fastidieuse : un grand soin est requis pour leur écriture car cela conditionne directement la qualité du résultat.

### A. Améliorer la sécurité

L'utilisation des motifs dans le traitement d'un formulaire apporte également davantage de sécurité. Il faut savoir que les formulaires et l'utilisation de certaines variables globales traitées par le navigateur comme \$\_GET ou \$\_POST présente des failles de sécurité par lesquelles des pirates peuvent tenter d'introduire du code malveillant.

Ces codes malveillants peuvent être de plusieurs natures, par exemple du HTML, du SQL ou du JavaScript. Ces codes malveillants utilisent généralement des caractères spéciaux : parenthèses, point-virgule, guillemets, chevrons, *slash* et *backslash*, etc. Les motifs vont nous permettre d'interdire tous ces caractères spéciaux et ainsi de protéger le site.

Remarquez qu'il existe des solutions semblables apportant presque le même niveau de sécurité : les fonctions sur les chaînes de caractères annulant ou remplaçant les caractères spéciaux comme `htmlspecialchars()`.

### B. Utiliser des motifs avec `preg_match()`

Les motifs sont des chaînes de caractères exploitées par des fonctions comme `preg_match()` (qui remplace les fonctions dépréciées `ereg()` et `eregi()` ).

```
if (preg_match("/php/i", "PHP est mon langage préféré !")) {
    echo "Un résultat a été trouvé.";
}
```

Autres fonctions pour les motifs

`preg_replace($motif, $ch1, $ch2)`

Remplace les occurrences de \$motif par \$ch1 dans \$ch2.

`preg_split($motif, $ch1)`

Éclate la chaîne \$ch1 selon les occurrences de \$motif. Le résultat est un tableau.

## C. Types de motifs : POSIX ou PCRE ?

Il existe 2 types de regex disponibles en PHP : les POSIX et les PCRE. Chaque type a sa propre syntaxe.

Les regex de type **POSIX** sont réputées plus faciles à coder mais légèrement plus lentes à l'exécution.

## D. Syntaxe des motifs POSIX

`"/mot/"` : chaîne contenant la chaîne "mot"

`"/^mot/"` : chaîne qui commence par "mot"

`"/mot$/"` : chaîne qui se termine par "mot"

`"/^chaîne$/"` : chaîne qui commence et se termine par "mot" et qui donc ne peut rien contenir d'autre.

`"/mot+/"` : chaîne contenant "mo" suivi d'un ou plusieurs "t" de suite

`"/(mot)+/"` : chaîne contenant une ou plusieurs fois de suite "mot"

`"/(mot)*/"` : chaîne contenant zéro ou plusieurs fois de suite "mot"

`"/(mot)?/"` : chaîne contenant zéro ou une fois "mot"

`"/(mot){3,8}/"` : chaîne contenant "mot" entre 3 et 8 fois de suite

`"/(mot){3,}/"` : chaîne contenant "mot" 3 fois de suite ou plus

`"/(mot){,8}/"` : chaîne contenant "mot" 8 fois de suite ou moins

`"/bob|joe/"` : chaîne contenant "bob" ou "joe"

`"/.{3}/"` : chaîne contenant 3 caractères

`"/[abc]/"` : chaîne contenant un "a", un "b", ou un "c"

`"/[^abc]/"` : chaîne ne contenant ni "a", ni "b", ni "c"

`"/^[abc]/"` : chaîne qui commence par un "a", un "b", ou un "c"

`"/^[^abc]/"` : chaîne qui ne commence pas par un "a", un "b", ou un "c"

`"/[a-z]/"` : chaîne contenant un caractère compris entre "a" et "z"

`"/[a-zA-Z0-9]/"` : chaîne contenant une lettre minuscule, majuscule ou un chiffre

<code>[[:alpha:]]</code>	caractères alphabétiques
<code>[[:upper:]]</code>	caractères majuscules
<code>[[:digit:]]</code>	chiffres
<code>[[:alnum:]]</code>	caractères alphanumériques (chiffres ou lettres)
<code>[[:blank:]]</code>	caractères blancs (espace, tabulation, ...)
<code>[[:space:]]</code>	caractère d'espacement
<code>[[:punct:]]</code>	caractère de ponctuation
<code>[[:print:]]</code>	caractères imprimables (tout caractère à part les codes spéciaux)

## E. Exercice : traduisez les 3 regex suivants en français

`"/^P+/"`

`"/[[:digit:]]{5,6}/"`

`"/^(hobbit){2}$/"`

## 1.11 Fichiers plats

Les fichiers plats sont des fichiers de données non structurés, par exemple des fichiers textes (.txt) ou des fichiers de données tabulaires (.csv). Ils peuvent être utiles pour enregistrer des données simples, non structurées, séparées par des retours à la ligne. Si les données à enregistrer sont plus structurées, on optera pour un fichier XML ou mieux, une base de données relationnelle.

- **Ouverture de fichiers avec `fopen()`**

```
$fichier = fopen('fichiers/commentaires.txt', 'a+');
```

Lors de cette ouverture de fichier, le fichier en entier est transféré dans une variable (appelée `$fichier` dans l'exemple). Le mode d'accès « a+ » permet de créer automatiquement le fichier s'il n'existe pas.

Mode d'accès	Lecture/écriture	Position du pointeur dans le fichier	Création du fichier s'il n'existe pas
<b>r</b>	<b>Lecture seule</b>	<b>Début</b>	<b>Non</b>
<b>r+</b>	<b>Lecture/écriture</b>	<b>Début</b>	<b>Non</b>
<b>w</b>	<b>Écriture seule</b>	<b>Début</b>	<b>Oui</b>
<b>w+</b>	<b>Lecture/écriture</b>	<b>Début</b>	<b>Oui</b>
<b>a</b>	<b>Écriture seule</b>	<b>Fin</b>	<b>Oui</b>
<b>a+</b>	<b>Lecture/écriture</b>	<b>Fin</b>	<b>Oui</b>

- **Tester l'existence d'un fichier**

La fonction `file_exists` permet de tester l'existence d'un fichier avant de l'exploiter.

```
if (file_exists ('notes.txt')) ...
```

- **Lire et afficher dans une page HTML avec `fgets()` et `feof()`**

La fonction `fgets` permet de lire une ligne du fichier. Un curseur retient la dernière ligne lue afin de ramener la ligne suivante au prochain appel.

```
fgets($fichier);
```

Pour lire tout le fichier, il suffit d'une boucle et d'utiliser la fonction `feof()` qui teste si on a atteint la fin du fichier.

```
while(!feof($fichier)) {
    $ligne = fgets($fichier); // récupère une ligne
    echo $ligne; // affiche la ligne
}
```

Plus directe que la méthode précédente, `file_get_contents()` retourne tout le contenu d'un fichier sous forme de chaîne de caractères.

```
$contenu = file_get_contents('notes.txt');
```

- **Enregistrement de nouvelles données dans le fichier avec *fputs()***

Cette fonction comporte deux paramètres : la variable contenant tout le fichier (*\$fichier*) et les nouvelles données à y inscrire (*\$donnees*).

Si lors de l'ouverture du fichier, nous avons spécifié comme mode d'accès « a+ », alors les nouvelles données seront écrites à la fin du fichier.

```
fputs($fichier, $donnees);
```

- **Fermeture du fichier avec *fclose()***

À la fin de l'utilisation du fichier, il faut le refermer. C'est à ce moment que le fichier est mis à jour et que **les modifications prennent effet**.

```
fclose($fichier);
```

## Exercice 10 : Formulaire avec fichiers plats

1. En vous basant sur les exercices précédents, développez un formulaire permettant au visiteur de poster une **appréciation** (liste de sélection avec « mauvais », « moyen », « bon », « très bon ») accompagnée d'un **commentaire** éventuel, d'un **pseudo** et d'un **email**.

Chaque appréciation postée subit les vérifications suivantes :

- Pseudo obligatoire, une *regex* vérifie que le pseudo ne contient que des lettres minuscules ou majuscules, accentuées ou non, des chiffres et les caractères ' - . \_
- Email obligatoire et vérifié par une *regex*
- Appréciation obligatoire et testée par liste blanche
- Commentaire non obligatoire

Tous les champs contenant du texte, à part l'appréciation, sont trimés et vidés d'éventuelles balises HTML.

2. Une fois les traitements acceptés (aucune erreur détectée), les résultats sont enregistrés dans un fichier texte, les uns en dessous des autres. Le fichier texte doit contenir les données comme suit :

```
Joé2000 [joe@chose.be]
Appréciation : très bon
"Bon sang, j'adore poster des commentaires sur ce site ! "
*****
```

PS : dans les fichiers textes, vous obtiendrez des retours à la ligne avec « \n ».

3. Le fichier plat est ensuite ouvert et son contenu est affiché dans la page. Les « \n » peuvent être habilement remplacés par des <br> à l'aide de la fonction **str\_replace()**.

## 1.12 POO : Programmation orientée objet

### A. Introduction POO

La programmation orientée objet (POO) permet au développeur de manipuler des entités regroupant plusieurs variables (propriétés) et fonctions (méthodes). Parmi d'autres avantages, ce concept apporte une meilleure lisibilité du code et tente de se rapprocher d'entités réelles décomposées en une série de paramètres et d'actions.

L'objet est un conteneur de variables (alors appelées propriétés ou attributs) et de fonctions (alors appelées méthodes). Ce conteneur se rapproche d'une entité réelle en rassemblant un certain nombre de propriétés (couleur, marque, poids, position, vitesse, etc.) et de fonctions (accélérer, tourner, klaxonner, etc.).

### B. Classes et objets

En POO, il convient dans un premier temps de définir les **classes** : des modèles énonçant tous les membres (propriétés et méthodes). Dans l'exemple ci-dessous, la *class* Voiture annonce les propriétés « couleur », « vitesse » et « acceleration » ainsi qu'une méthode « accélérer ».

```
<?php
// déclaration d'une classe
class Voiture {
    // déclaration des propriétés (avec ou sans type)
    // valeurs par défaut non obligatoires
    protected string $couleur = 'gris métallisé';
    private float $vitesse = 32.7 ;
    private $acceleration = 0.2 ;

    // déclaration des méthodes
    public function accélérer() {
        $this->vitesse += $this->acceleration ;
    }
}
```

La visibilité des propriétés et des méthodes est définie avec les mots-clés : **public**, **protected**, ou **private** :

- Les éléments déclarés avec **public** sont accessibles partout.
- Les éléments déclarés avec **protected** ont un accès limité à la classe elle-même, ainsi qu'aux classes parentes ou qui en héritent.
- L'accès aux éléments déclarés avec **private** est uniquement réservé à la classe elle-même.

Il est possible de typer les propriétés afin de bloquer leur type. Pour cela nous utiliserons les mots-clés **int**, **float**, **string**, **bool**, etc.

La pseudo-variable **\$this** représente l'objet courant.

Le symbole **->** est utilisé entre un objet et ses membres (propriétés/méthodes).



Le symbole `::` est utilisé pour accéder aux membres statiques et aux constantes.  
Cette classe est ensuite utilisée afin d'**instancier** des **objets**. C'est la commande ***new*** qui crée un nouvel objet en se servant d'une classe déclarée.

```
// instanciation d'un objet depuis la classe Voiture
$maCaisse = new Voiture;
?>
```

Remarquez le bon usage de la casse : le nom de la classe utilise l'écriture *UpperCamelCase* alors que le nom de l'objet (ainsi que les noms de propriétés, variables, méthodes, etc.) utilise l'écriture *lowerCamelCase*.

## C. Héritage

L'héritage en PHP permet de créer une nouvelle classe qui héritera des propriétés et des méthodes d'une classe parent et qui pourra, si on le souhaite, redéfinir certaines propriétés et méthodes.

Pour étendre une classe, il suffit d'utiliser le mot-clé ***extends*** de cette façon :

```
<?php
class Vehicule {
    private $vitesse = 0 ;
}
class Voiture extends Vehicule {
    private $nbRoues = 4 ;
}
```

Dans cet exemple, la classe *Voiture* peut profiter du paramètre *vitesse* de la classe *Vehicule*.

## D. *stdClass*

PHP dispose d'une classe générique ***stdClass*** aux propriétés dynamiques. Cela permet d'éviter la déclaration de la classe.

```
<?php
$newObj = new stdClass();
?>
```

## E. Sérialisation

Un objet PHP ne peut pas être stocké en session à moins d'être sérialisé. La **sérialisation**, ou **linéarisation**, est la transformation d'un objet ou d'un tableau sous forme de chaîne de caractères. Le processus inverse se nomme la **réhydratation**.

La sérialisation d'un objet simplifie aussi sa sauvegarde ou son transport sur le réseau.

PHP dispose de sa propre sérialisation avec les fonctions : ***serialize()*** et ***unserialize()***.

Mais actuellement, le format de sérialisation le plus courant est **JSON**. Sans rentrer dans les détails, PHP propose 2 fonctions simples pour utiliser JSON : ***json\_encode()*** et ***json\_decode()***.

Ainsi, l'exemple suivant :

```
<?php
$data = new stdClass();    //création d'objet
$data->nbSeasons = 5;
$data->nbEpisodes = array(7, 13, 13, 13, 16);
$data->title = "Breaking Bad";
echo json_encode($data);
?>
```

Affichera :

```
{"nbSeasons":5,"nbEpisodes":[7,13,13,13,16],"title":"Breaking Bad"}
```

## F. Espaces de noms (*namespace*)



Déclarer 2 fonctions, 2 constantes ou 2 classes portant le même nom provoque une erreur PHP. Les *namespace* résolvent ce problème en agissant comme des conteneurs de noms et aident à la réutilisabilité du code.

La commande *namespace* suivie d'un nom, est obligatoirement située au début de vos codes PHP, et permet de déclarer un espace de noms (ou *namespace*).

Le fonctionnement des espaces de noms est similaire à celui de dossiers : vous ne pouvez donner le même nom à 2 fichiers que s'ils sont situés dans 2 dossiers différents. Par analogie, les espaces de noms permettent de donner le même nom à plusieurs fonctions (ou constantes ou classes). Ceci peut être utile lorsque l'on combine de nombreux fichiers PHP.

```
<?php
namespace monEspaceDeNoms ;
...
?>
```

Ici, la déclaration de *namespace* implique que toutes les déclarations de fonctions suivantes seront propres à ce *namespace*. Il est même possible d'utiliser des noms de fonctions réservées par PHP comme *header()* ou *strlen()*.

Dans un *namespace*, pour accéder aux fonctions globales malgré leur redéfinition, on précède leur nom d'un caractère « \ ». Dans l'exemple ci-dessous, les 2 premiers *echo* affichent « Hello world ! » alors que le dernier affiche 12.

```
<?php
namespace monEspaceDeNoms ;

function strlen($x) {
    echo $x;
}

echo strlen('Hello world!');
echo \monEspaceDeNoms\strlen('Hello world!');

echo \strlen('Hello world!');
?>
```

Il est également possible de déclarer des sous-espaces de noms :

```
<?php
namespace monEspaceDeNoms\sousEspace ;
?>
```

Ou de revenir au *namespace* global :

```
<?php
namespace ;
?>
```

En cas de doute, une constante contient le nom du *namespace* actuel :

```
<?php
echo __NAMESPACE__ ;
?>
```

La commande *use* permet de créer des alias d'espaces de noms, ce qui peut s'avérer utile si cet espace de noms est un peu long à écrire (car composé de plusieurs noms par exemple). Les 2 instructions suivantes ont le même effet : renommer l'espace de noms « C » pour la suite du code.

```
<?php
use monEspaceDeNoms\sousEspace as E;
use monEspaceDeNoms\sousEspace ;
?>
```

Mais la commande *use* sert également à importer des classes depuis un *namespace*.

```
<?php
namespace \A\B\C;

class MaClasse {
    public function hello() {
        echo 'Hello world!';
    }
}
?>
```

Dans un autre fichier :

```
<?php
require 'C.php';

use A\B\C\MaClasse as Bonjour;

$a = new Bonjour;
$a->hello();
?>
```

On dit d'une fonction (constante ou classe) qu'elle est **qualifiée** si le *namespace* dans lequel il se trouve est spécifié. Sinon, on dit de cette fonction qu'elle est **non qualifiée**.

## 1.13 PHP et MySQL : exemples

### A. Principales commandes PDO

```
$bd->exec();
```

Exécute une requête DELETE, UPDATE ou INSERT.

```
$bd->query();
```

Exécute une requête SELECT.

```
$req=$bd->prepare('');
```

Prépare une requête SQL et renvoie un objet PDO en résultat.

```
$req->execute();
```

Exécute une requête préparée.

```
$req->bindValue(':x', $x);
```

Associe la valeur de \$x au paramètre SQL entier :x

```
$req->bindParam(':x', $x);
```

Associe la variable \$x au paramètre SQL entier :x

```
$req->setFetchMode(PDO::FETCH_OBJ);
```

Définit le mode de réception des résultats de la requête. Le paramètre non obligatoire « PDO::FETCH\_OBJ » demande un retour des valeurs sous forme d'objets.

```
$req->fetch(PDO::FETCH_OBJ)
```

Récupère la ligne suivante de résultats.

```
$req->rowCount();
```

Renvoie le nombre de lignes du résultat.

```
$req->closeCursor();
```

Ferme le curseur, permettant à la requête d'être à nouveau exécutée.

**PDO** (PHP Data Objects) est une extension PHP (à partir de la version 5.1) pour les accès aux bases de données permettant de préparer les requêtes.

Les requêtes préparées (*prepare()* puis *execute()*) apportent deux avantages importants : elles permettent de meilleures performances lors de requêtes appelées en boucle car seule l'exécution de la requête se fait dans la boucle, et permettent un meilleur niveau de sécurité en bloquant les attaques par injection SQL.

Les commandes *exec()* et *query()* sont réservées aux requêtes sans paramètres et hors des boucles.

## B. Se connecter à la base de données

```
date_default_timezone_set('Europe/Brussels');
$hote='localhost';
$nomBD='projetWeb';
$user='root';
$mdp='';

try {
    $bd=new PDO('mysql:host='.$hote.';dbname='.$nomBD, $user, $mdp);
    $bd->exec("SET NAMES 'utf8'");
}
catch (Exception $e) {
    echo "Erreur de connexion à la BD : $e";
}
```

## C. Charger les noms des catégories depuis la BD

```
$reqCat=$bd->prepare('SELECT * FROM categorie');
$reqCat->execute();
$reqCat->setFetchMode(PDO::FETCH_OBJ);
while ($result=$reqCat->fetch() ) {
    echo '<p>', $result->nom;
}
$reqCat->closeCursor();
```

### Variante non préparée :

```
$reqCat=$bd->query('SELECT * FROM categorie');
$reqCat->setFetchMode(PDO::FETCH_OBJ);
while ($result=$reqCat->fetch() ) {
    echo '<p>', $result->nom;
}
$reqCat->closeCursor();
```

L'étoile « \* » signifie que vous souhaitez récupérer tous les champs.

La commande SQL « **AS** » permet de renommer une colonne ou une table en lui donnant un alias. Cet alias peut alors être utilisé dans la commande et dans les résultats réceptionnés en PHP.

Par exemple :

```
$reqCat=$bd->query('SELECT nom AS cat FROM categorie');
```

Ou encore :

```
$reqCat=$bd->query('SELECT c.nom FROM categorie AS c');
```

## D. Charger un article précis

```
$reqArt=$bd->prepare('SELECT img, nom FROM article
                        WHERE id_article=:id_article');
$reqArt->bindValue(':id_article', $id_article);
$reqArt->execute();
$article=$reqArt->fetch(PDO::FETCH_OBJ);
if (!empty($article)) {
    echo 'nom,'">' ;
}
$reqArt->closeCursor();
```

### Variante 1 :

```
$reqArt=$bd->prepare('SELECT img, nom FROM article
                        WHERE id=:id AND statut=:statut');
$reqArt->execute(array(':id'=>$id, ':statut'=>$statut));
$article=$reqArt->fetch(PDO::FETCH_OBJ);
if (!empty($article)) {
    echo 'nom,'">' ;
}
$reqArt->closeCursor();
```

### Variante 2 :

```
$reqArt=$bd->prepare('SELECT img, nom FROM article
                        WHERE id=? AND statut=?');
$reqArt->execute(array($id,$statut));
$article=$reqArt->fetch(PDO::FETCH_OBJ);
if (!empty($article)) {
    echo 'nom,'">' ;
}
$reqArt->closeCursor();
```

## E. Exécuter une requête en boucle

```
$id_articles = array(2,4,10,17,19,35); // des id d'articles

$reqArt=$bd->prepare('SELECT nom FROM article WHERE
id_article=:id_article');
$reqArt->bindParam(':id_article', $id_article);
$reqArt->setFetchMode(PDO::FETCH_OBJ);

foreach ($id_articles as $a) {
    $id_article = $a;
    $reqArt->execute();
    $article=$reqArt->fetch();
    echo '<p>',$article->nom ;
}

$reqArt->closeCursor();
```

Dans cet exemple, la requête est exécutée dans une boucle. Pour accélérer l'exécution du script, certaines instructions précèdent la boucle : *prepare()*, *bindParam()* et *setFetchMode()*. Le *closeCursor()* lui aussi n'est exécuté qu'une fois : après la boucle.

## F. Fonctions SQL : nombre, moyenne, somme, min et max

```
$reqFilm=$bd->prepare('SELECT AVG(duree) FROM film WHERE
id_realisateur=8');
$reqFilm->execute();
$result=$reqFilm->fetch(PDO::FETCH_OBJ);
$reqFilm->closeCursor();
```

Cet exemple retourne la durée moyenne des films dont le réalisateur a l'ID 8. De la même manière, vous pouvez obtenir :

- La moyenne avec AVG()
- La somme avec SUM()
- Le nombre avec COUNT()
- Le maximum avec MAX()
- Le minimum avec MIN()

## G. GROUP BY

```
$reqFilm=$bd->prepare('SELECT AVG(duree) AS dureeMoy FROM film WHERE
annee>2000 GROUP BY id_realisateur HAVING dureeMoy>90');
$reqFilm->execute();
while ($result=$reqFilm->fetch() ) {
    echo '<p>',$result->dureeMoy;
}
$reqFilm->closeCursor();
```

On récupère ici les durées moyennes de films parus après 2000, regroupées par réalisateur et dont la durée moyenne est supérieure à 90. Cette 2<sup>e</sup> clause (>90) ne peut pas intervenir dans la clause WHERE car porte sur un paramètre calculé avec la fonction AVG.



## H. UPDATE

Attention à ne jamais oublier la clause WHERE dans vos UPDATE, sinon vous modifiez toute la table !

```
$modifBD = $bd->prepare('UPDATE bande_dessinee SET auteur=:auteur,
titre=:titre WHERE id=:id');
$modifBD->bindValue(':id',$id);
$modifBD->bindValue(':auteur', $_POST['auteur']);
$modifBD->bindValue(':titre', $_POST['titre']);
$modifBD->execute();
$modifBD->closeCursor();
```

Dans cet exemple, certains paramètres viennent directement de la variable \$\_POST. C'est plutôt risqué, mais pas interdit dans la mesure où PDO bloque tout de même l'effet des injections SQL.

## I. INSERT

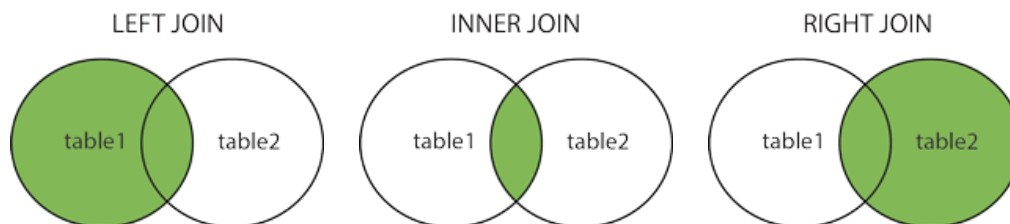
```
$reqNewBD = $bd->prepare('INSERT INTO bande_dessinee (titre, auteur)
VALUES (:titre, :auteur)');
$reqNewBD->bindValue(':titre', $titre);
$reqNewBD->bindValue(':auteur', $auteur);
$reqNewBD->execute();
$reqNewBD->closeCursor();
```

## J. JOIN

Lors d'une lecture (SELECT), si les données à lire proviennent de plusieurs tables, il y a lieu d'utiliser des jointures pour les réunir.

Attention, lorsqu'une requête porte sur plusieurs tables, mieux vaut écrire les noms des champs sous leur forme complète : « table.champ » (au cas où le nom serait présent dans plusieurs tables).

Voici 3 jointures possibles selon les résultats souhaités :



MySQL ne propose pas de jointure FULL JOIN ramenant tout.

Un exemple de requête SQL où « films » est « left table » et « réalisateurs » est « right table » :

```
SELECT films.titre, réalisateurs.nom
FROM films
INNER JOIN réalisateurs ON réalisateurs.id_film=film.id_film
WHERE films.duree<=120
```

Dans cet exemple, la jointure INNER JOIN ramène les titres de films et leurs noms de réalisateurs, mais pas les films sans réalisateurs, ni les réalisateurs sans films. La condition de jointure est ici une correspondance de champs « id\_film ».

Quelle jointure choisir si l'on souhaite obtenir tous les films avec ou sans réalisateurs (mais pas les réalisateurs sans films) ?

Il est possible de joindre une table à elle-même : on parle alors de « *self join* » même si cette commande n'existe pas dans MySQL. Cette jointure spécifique intervient par exemple lorsqu'une table « employes » comprend un champ « id\_patron » pointant vers l'ID d'un autre employé de cette même table.

```
SELECT employes.nom, patrons.nom AS patron_nom
FROM employes
LEFT JOIN employes AS patrons
ON patrons.id_employe=employes.id_patron
```

Remarquez l'utilisation de AS pour renommer des tables ou des champs afin d'éviter les doublons.

## Exercice 12 : Walking Pets

Exercice PHP + MySQL



### Fichiers à disposition sur Moodle

- Un dossier contenant une série d'icônes
- Un fichier « walkingPets.sql » contenant une BD à compléter



### 1<sup>re</sup> étape

1. Importez la BD dans phpMyAdmin. Il se peut que l'importation directe ne fonctionne pas et que vous deviez créer une BD « walkingPets » et y importer la BD.
2. Créez une table « categories » contenant les champs suivants :
  - a. « id\_cat » : l'id de catégorie (clé primaire, auto-incrémentée, *unsigned*)
  - b. « name » : le nom
  - c. « icon » : le nom de l'icône correspondant (dog.png pour le chien, bunny.png pour le lapin, etc.)
  - d. « max\_walks » : le nombre de promenade maximum par catégorie d'animaux (2 pour les chiens, 1 pour les chats, 0 pour les autres)
3. Insérez dans cette table, les données suivantes (attention à bien respecter les id) :

id_cat	name	icon	max_walks
1	chien	dog.png	2
2	chat	cat.png	1
3	rongeur	hamster.png	0
4	oiseau	parrot.png	0
5	lapin	bunny.png	0
6	poisson	fish.png	0
7	reptile	turtle.png	0

## 2<sup>e</sup> étape

4. Créez un dossier « walkingPets » dans le dossier « www » et placez-y le dossier d'icônes.
5. Créez un fichier « style.css », un fichier « index.php », un fichier « \_bd.php » et un fichier « \_header.php ».
6. Codez « \_bd.php » afin de vous connecter à la BD « walkingPets ».
7. Dans « index.php », incluez le fichier « \_bd.php » au début du code, liez le fichier « style.css » et incluez « \_header.php » au début du body.
8. Dans « include\_header.php », codez ceci :



Attention, la 2<sup>e</sup> ligne (reprenant les noms de catégories d'animaux) est générée depuis les contenus en BD. À vous de coder une requête SQL ramenant les noms de catégories et générant exactement le résultat ci-dessus.

## 3<sup>e</sup> étape

9. Complétez « index.php » afin d'afficher les noms des animaux (depuis la table « pets » de la BD), leur nombre de promenades (« walks ») et leur nombre de repas (« meals ») ainsi que l'icône de leur catégorie.



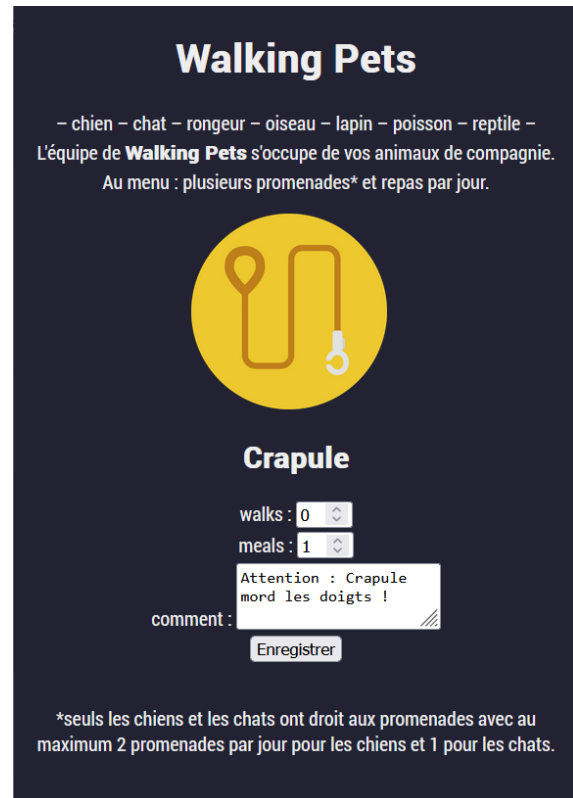
Pour cela, vous aurez besoin d'une requête utilisant une jointure SQL. Choisissez une jointure, sachant que certains *pets* ont 0 en *id\_cat* et qu'aucune catégorie n'a d'*id\_cat* valant 0...

4<sup>e</sup> étape

10. Pour chaque animal, testez si l'icône est vide et affichez l'icône de la grosse papatte si c'est le cas (cf. ci-dessous : Brownie, Doumi et Roro).
11. Testez également si « *comment* » n'est pas vide et affichez « – remarque – » si c'est le cas (cf. ci-dessous : Brownie, Crapule, Doumi, Dynamite et Roro). Codez un attribut HTML *title* sur le mot « remarque » afin d'afficher la remarque au survol.
12. Pour finir, testez si le nombre de promenades demandées est supérieur au nombre de promenade maximum « *max\_walks* » spécifié dans la table *categories*. Si c'est le cas, barrez le nombre de promenade demandé et affichez le nombre de promenade max (cf. ci-dessus : c'est le cas entre autres de Bouldegome et Caramel).
13. Un peu de CSS sur le tout pour faire beau 😊

## 5<sup>e</sup> étape

14. Chaque *pet* est cliquable et envoie vers une page « `modif_pet.php` » avec l'id du *pet* en paramètre GET. Créez cette page PHP et récupérez et sécurisez ce paramètre.
15. Tout en haut de votre script (avant le `<!DOCTYPE html>`), exécutez une requête SQL ramenant toutes les données du *pet* dont vous avez récupéré l'id. Vérifiez avec la méthode `->rowCount()` si le nombre de résultats reçus est de 0. Dans ce cas, renvoyez le visiteur vers la page d'index avec la commande `header()`. Effectuez 2 tests : l'un avec un *id\_pet* existant, l'autre avec un *id\_pet* inexistant.
16. Affichez un formulaire permettant de modifier les valeurs de *walks*, *meals* et *comment*. En HTML, faites-en sorte que *walks* et *meals* soient des champs *number* avec une valeur minimum de 0. Le *comment* quant à lui sera représenté sous forme de *textarea*.
17. Codez ensuite la réception de ce formulaire entre la réception du paramètre GET et le `<!DOCTYPE html>`. À l'aide d'une requête UPDATE, mettez les données de l'animal à jour dans votre BD. Puis, redirigez le visiteur vers la page d'index.



## 6<sup>e</sup> étape

18. Démarrez une session dans les 2 pages PHP. Chaque fois qu'un *pet* est modifié sur la page « `modif_pet.php` », ajoutez la valeur de son id dans un tableau stocké dans la variable de session. Ce qui donne plus ou moins :  
`$_SESSION['pets'][] = ... ;`
19. Puis, modifiez la page « `index.php` » : lors de l'affichage des *pets*, vérifiez pour chacun si l'id est dans le tableau avec la fonction `in_array()` et affichez le texte du *pet* en jaune si c'est le cas. De cette façon, tous les animaux modifiés récemment s'affichent en jaune.

## 1.14 Coder comme des professionnels : PSR et MVC

### A. Coder comme des professionnels

Un code fonctionnel, n'est pas forcément professionnel.

L'apprentissage des bases du langage, vous permet de produire des codes fonctionnels, mais pas forcément professionnels. Ce chapitre vous présente des pistes pour aller plus loin en abordant les habitudes des développeurs professionnels et certaines architectures de code.

Un code « pro » est souvent :

- **Modulaire** : découpé en nombreux fichiers indépendants et ayant chacun un rôle unique.
- **Documenté** : proprement au-dessus de chaque classe et méthode.
- **En anglais** : les commentaires, les noms de variables, de fonctions, etc.
- Codé dans le respect des **normes de formatage**, le plus souvent **PSR-12**.

Tout ceci devrait permettre à vos codes d'être réutilisables, évolutifs et de pouvoir y travailler à plusieurs.

### B. Les PSR

Les PSR (*PHP Standard Recommendation*) sont des standards de programmation propres au PHP, des bonnes pratiques qui ont pour but d'harmoniser les habitudes de développement.

Il existe une quinzaine de PSR, concernant différents domaines :

- Les classes de chargement automatique : PSR-4
- Les interfaces : PSR-3, PSR-6, PSR-11
- HTTP : PSR-7, PSR-15
- Habitudes et syntaxe de développement : **PSR-1**, **PSR-12**

La **PSR-1** suggère des recommandations :

- Les seules balises PHP permises sont : `<?php ... ?>` et `<?= ... ?>`
- Le code PHP doit utiliser uniquement UTF-8
- Les noms de méthodes sont écrits en *camelCase* : objet->changeCouleur()
- Etc.

La **PSR-12** « Guide de style de codage étendu » qui remplace la PSR-2 est certainement l'une des plus connues :

- Le code doit respecter **PSR-1**
- Les fichiers ne contenant que du PHP ne doivent pas se terminer par `?>`
- Les types doivent utiliser leurs syntaxes raccourcies : **int** au lieu de **integer**, etc.
- Les lignes de code ne devraient pas dépasser 80 caractères.
- Tous les mots-clés et types PHP doivent être en minuscules.
- Les accolades de fermeture ne doivent pas être suivies d'un commentaire ou d'une déclaration sur la même ligne.
- Etc.

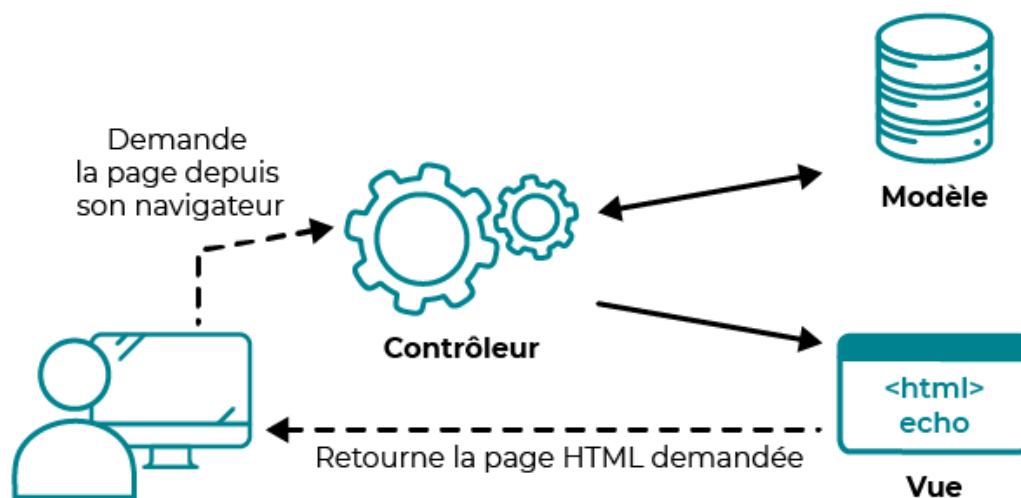
Il existe des outils comme PHP-CS-Fixer afin de corriger vos codes vis-à-vis des recommandations PSR-12.

En conclusion, les PSR permettent d'harmoniser le travail des développeurs, mais sont parfois aussi décriées par les communautés de développeurs qui y voient trop de contraintes. Se conformer à PSR-12 permet de mieux s'intégrer dans une équipe de développeurs.

### C. L'architecture Modèle-Vue-Contrôleur

Modèle-Vue-Contrôleur ou MVC est un patron de conception (*design pattern*) impliquant une répartition du code des pages web en 3 parties :

- Le **modèle** traite les données (dossier « src ») : on y retrouve la connexion à la BD et les accès, ainsi que les traitements PHP nécessaires avant affichage.
- La **vue** affiche les informations (dossier « templates ») : on y retrouve de l'HTML et un peu de PHP à des fins d'affichage (des boucles, des conditions, etc.).
- Le **contrôleur** fait le lien entre les deux (dossier « controllers »), mais aussi avec l'utilisateur : on y retrouve la réception des requêtes de l'utilisateur, la demande au modèle et l'envoi du résultat du modèle à la vue.





## D. Exemple MVC

**Modèle** : des fonctions de connexion à la BD, de lecture des données de la BD, etc.

```
<?php
// Connection to the database
function connectDB() {
    try {
        $database = new
PDO('mysql:host=localhost;dbname=mvc_demo;charset=utf8', 'root', '');
        return $database;
    }
    catch(Exception $e) {
        die('Erreur : '.$e->getMessage());
    }
}
// Retrieve all the questions
function getQuestions() {
    $database = connectDB();
    $statement = $database->query(
        'SELECT question, answer FROM questions ORDER BY order ASC'
    );
    $questions = [];
    while ($row = $statement->fetch()) {
        $questions[] = [
            'question' => $row['question'],
            'answer' => $row['answer']
        ];
    }
    return $questions;
}
```

**Vue** : de l'HTML et un peu de PHP pour générer la page et ses contenus

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>FAQ</title>
    <link href="style.css" rel="stylesheet">
</head>

<body>
    <h1>Foire aux Questions</h1>
    <dl>
        <?php
        foreach ($questions as $question) {
            ?>
            <dt><?= $question['question'] ?></dt>
            <dd><?= $question['answer'] ?></dd>
            <?php
            }
            ?>
        </dl>
    </body>
</html>
```

**Contrôleur** : chargement du modèle, appel des traitements nécessaires, chargement de la vue.

```
<?php
require 'src/faq.php';

$posts = getQuestions();

require 'templates/faq.php';
```

Remarquez que les fichiers ne comportant que du PHP ne se terminent jamais par « ?> ». Cela afin d'éviter que des caractères blancs HTML soient générés avant la vue.

Le passage d'un code « classique » à un patron de conception comme MVC, s'appelle la **factorisation** du code.

Cet exemple est volontairement simple. On pourrait aller plus loin en ajoutant un routeur chargé d'aiguiller les requêtes vers les bons contrôleurs.

# Récapitulatifs HTML et CSS

## Récapitulatif HTML

### Code de base de tout document HTML

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8">
    <title>...</title>
</head>
<body>
    ...
</body>
</html>
```

### Méta-balises

```
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="As des as du FBI, Bill Baroud est un espion
solide, un homme moulé dans de l'acier, ...">
<meta name="keywords" content="FBI, espion, Bill Baroud, héros">
<meta name="author" lang="fr" content="Bill Baroud">
<meta name="robots" content="index, follow">
```

### Éléments neutres (ni sémantique, ni mise en forme par défaut)

```
<span>Ligne neutre</span>
<div>Bloc neutre</div>
```

### Blocs sémantiques

```
<header>En-tête</header>
<footer>Pied de page</footer>
<main>Bloc principal</main>
<nav>Navigation</nav>
<article>Sujet à part entière</article>
<section>Section d'une page ou d'un article</section>
<aside>Compléments d'informations</aside>
```

### Titres et éléments de texte

```
<h1>Titre de la page (unique)</h1>
<h2>Titre de deuxième niveau</h2>
<h3>Titre de troisième niveau</h3>
<h4>Titre de quatrième niveau</h4>
<p>Paragraphe</p>
<strong>élément de texte important</strong>
<blockquote>Citation (bloc)</blockquote>
<cite>Citation (ligne)</cite>
<address>Citation</address>
<acronym title="Federal Bureau of Investigation">FBI</acronym>
```

### Liens

```
<a href="dossier/page.html">Lien interne</a>
<a href="dossier/page.html#ancree">Lien interne ancré</a>
<a href="http://www.heh.be" target="_blank" rel="noopener">Lien externe</a>
```

### Images

```
<figure>
  
  <figcaption>Description</figcaption>
</figure>
```

### Listes : non numérotée (<ul>), numérotée (<ol>), d'associations (<dl>)

```
<ul>                                <ol>                                <dl>
  <li>Elem1</li>                    <li>Elem1</li>                    <dt>Expression</dt>
  <li>Elem2</li>                    <li>Elem2</li>                    <dd>Association</dd>
</ul>                                </ol>                                </dl>
```

### Tableaux

```
<table>
  <tr>
    <td rowspan="2">Cellule avec fusion verticale</td>
    <th colspan="2">Cellule d'en-tête avec fusion horizontale</th>
  </tr>
  <tr>
    <td>Cellule</td>
    <td>Cellule</td>
  </tr>
</table>
```

### Formulaires

```
<form action="form.html" method="post" autocomplete="none">
  <fieldset>
    <legend>identité</legend>
    <label for="prenom">Un champ texte obligatoire : <input type="text"
name="prenom" id="prenom" size="32" required></label>
    <label for="nom">Idem avec auto-complétion : <input type="text" name="nom"
id="nom" size="32" list="noms"></label>
    <datalist id="noms">
      <option value="Danes">
      <option value="Lewis">
      <option value="Patinkin">
    </datalist>
    <label for="avis">Un champ texte multiligne :
      <textarea cols="50" rows="5" name="avis" id="avis"></textarea> </label>
    <label>Password : <input type="password" name="password"
size="32"></label>
    <label>Un nombre : <input type="number" name="enfants" value="18"></label>
    <label>Un email : <input type="email" name="email" size="32"></label>
    <label>Une date: <input type="date" name="naissance"></label>
    <label><input type="checkbox" name="options[]" value="1">Option 1</label>
    <label><input type="checkbox" name="options[]" value="2">Option 2</label>
    <label><input type="radio" name="choix" checked>Bouton radio coché</label>
    <label><input type="radio" name="choix">Bouton radio</label>
    <select name="pays" id="pays">
      <option value="B" selected>Belgique</option>
      <option value="F">France</option>
    </select>
    <label><input type="range" name="val" min="0" max="20" value="8"></label>
    <input type="submit" value="Valider" name="submitted">
  </fieldset>
</form>
```

## Retours à la ligne et séparations horizontales

`<br>` retour à la ligne  
`<wbr>` permission de césure au sein d'un long mot  
`<hr>` barre de séparation horizontale

## Barres de progression et de mesure

`<meter min="0" max="100" value="52" low="50" high="60">52%</meter>`  
`<progress max="100" value="52">52%</progress>`

## Récapitulatif des sélecteurs CSS

Sélecteurs	Syntaxe	Descriptions
Sélecteur de type	<code>x { }</code>	Toutes les balises <code>x</code>
Sélecteur d'ID	<code>#nom { }</code>	Toutes les balises dont l'ID est <code>nom</code>
Sélecteur d'ID et de type	<code>x#nom { }</code>	Toutes les balises de type <code>x</code> dont l'ID est <code>nom</code>
Sélecteur de classe	<code>.nom { }</code>	Toutes les balises dont la classe est <code>nom</code>
Sélecteur de classe et de type	<code>x.nom { }</code>	Toutes les balises <code>x</code> dont la classe est <code>nom</code>
Sélecteur multiple	<code>x, y { }</code>	Toutes les balises <code>x</code> et <code>y</code>
Sélecteur descendant	<code>x y { }</code>	Toutes les balises <code>y</code> contenue dans une balise <code>x</code> (descendants)
Sélecteur d'enfant	<code>x&gt;y { }</code>	Toutes les balises <code>y</code> directement contenues dans une balise <code>x</code> (enfants)
Sélecteur universel	<code>x * y { }</code>	Toutes les balises <code>y</code> contenues dans une balise quelconque, elle-même contenue dans une balise <code>x</code> (descendants non enfants)
Sélecteur de frères adjacents	<code>x + y { }</code>	Toutes les balises <code>y</code> s'ouvrant après la fermeture d'une balise <code>x</code>
Sélecteurs de pseudo-classes	<code>x:z { }</code>	Toutes les balises <code>x</code> dans un état <code>z</code> (par exemple <code>:hover</code> , <code>:focus</code> , <code>:active</code> , <code>:checked</code> , etc.)

## Responsive

Réserver du CSS à certaines tailles de fenêtre

```
@media (max-width:760px) { }
```

Appliquer du CSS ou pas selon la taille de l'ancêtre doté de `container-type: inline-size;`

```
@container (max-width:640px) { }
```

## Sources

BRILLANT Alexandre, XML cours et exercices, éditions Eyrolles, 2007  
ENGELS Jean, PHP5 cours et exercices, éditions Eyrolles, 2004  
PARISOT Thomas, Réussir son Blog professionnel, éditions Eyrolles, 2009  
WYKE-SMITH Charles, Coder pour le Web, CampusPress, 2007

Alsacrérations : <http://css.alsacreations.com/>  
Comment ça marche : <http://www.commentcamarche.net/contents/>  
Développez.com : <http://www.developpez.com/>  
OpenClassrooms : <https://openclassrooms.com/fr/>  
PHP.net : <http://www.php.net/>  
Validator W3C : <http://validator.w3.org/>

### Ouvrages conseillés aux étudiants :



**PHP et MySQL** : HEURTEL Olivier, Maîtrisez le développement d'un site web dynamique et interactif (4e édition), éditions Eyrolles, 2019

