



# PHP

Scripts serveur

# Principes de base

## Rappel initiation à la programmation

Structures de base

Opérateurs

Structures de contrôle

Conditionnelles

Itérations

Variables externes

# Structures de base

- **Balises d'ouverture et de fermeture**

`<?php`

`?>`

- **Commentaires**

`/*`

`* Commentaire sur`

`* plusieurs lignes`

`*/`

`//` Commentaire sur une seule ligne

- **Délimiteurs et fin d'instruction**

Guillemets simples (') ou guillemets doubles (")

Point virgule à la fin d'une instruction `// Parse error: syntax error`

- **Instructions de sortie**

<code>echo 'PHP 7';</code>	<code>echo "PHP 7"</code>
<code>print 'PHP 7';</code>	<code>print "PHP 7"</code>
<code>&lt;?= 'PHP 7' ?&gt;</code>	<code>&lt;?= "PHP 7"&gt;</code>

- **Inclusions**

`include` et `include_once`  
`require` et `require_once`

- **Variables – Création (affectation ou assignation)**

`$name = 'John Doe'; // Assignation`  
`$age = 25; // Assignation`  
`$val = $price + 10; // Assignation`

- **Variables - types**

En PHP il ne faut pas pré déclarer une variable, celle-ci est typée lors de son assignation (typage dynamique).  
Il existe 8 types de variables répartis dans trois groupes:

- **Scalaire**

`$a = 'Doe'; // String`

`$b = -10; // Integer`

`$c = 1.21; // Float (décimaux)`

`$d = true; // Boolean`

- **Composées**

`$e = ['PHP', 'JS', 'HTML', 'CSS']; // Array`

`$f = new DateTime(); // Object`

- **Spéciaux**

`$g = fopen("foo", "w"); // Resource`

`$h = null; // Null`

- **Constantes**

```
define('LOGIN', 'johndoe'); // Définition d'une constante
```

- **Les tableaux (array)**

- **Tableaux indexés (indiciels)**

```
$names = ['Servais', 'Marthus', 'Doe']; // Tableau indexé
```

- **Tableaux associatifs**

```
$courses = ['PHP' => 120, 'HTML' => 80, 'CSS' => 80]; // Tableau associatif
```

- **Accès (hors boucle)**

```
echo $names[0]; // Servais
```

```
echo $courses['HTML']; // 80
```

# Fonctions sur les variables

- **ISSET()**

Détermine si une variable est définie et est différente de NULL.

- **EMPTY()**

Détermine si une variable est considérée comme vide. Une variable est considérée comme vide si elle n'existe pas, ou si sa valeur équivaut à FALSE.

- **UNSET()**

Détruit la ou les variables dont le nom a été passé en argument

- **GETTYPE()**

Retourne le type de la variable. vous pouvez aussi utiliser les fonctions **is\_** comme is\_string(), is\_int()...

- **SETTYPE(var, "type")**

Force le type de la variable

- **VAR\_DUMP()**

affiche les informations structurées d'une variable

# Traitements de base

- **Opérateurs arithmétiques**

- $-\$a$  Négation
- $\$a + \$b$  Addition
- $\$a - \$b$  Soustraction
- $\$a * \$b$  Multiplication
- $\$a / \$b$  Division
- $\$a \% \$b$  Modulo (Reste de  $\$a$  divisé par  $\$b$ )
- $\$a ** \$b$  Exponentielle



- **Opérateurs d'affectation**

- **Simple**

- ```
$a = 10;
```

- **Opérateurs combinés**

- En plus du simple opérateur d'affectation, il existe des "opérateurs combinés" pour tous les opérateurs arithmétiques.

- ```
$a = 10;
```

- ```
$b = 5;
```

- ```
$c = 'Hello';
```

- ```
$b += $a; // 15
```

- ```
$c .= 'World !'; // HelloWorld !
```

- **Opérateurs de comparaison**

\$a == \$b	Egal
\$a === \$b	Identique
\$a != \$b	Différent
\$a !== \$b	Différent (si \$a est différent de \$b ou bien s'ils ne sont pas du même type)
\$a < \$b	Plus petit que
\$a > \$b	Plus grand
\$a <= \$b	Inférieur ou égal
\$a >= \$b	Supérieur ou égal

- **Opérateurs logiques**

! \$a	Not (Non)
\$a && \$b	And (Et)
\$a    \$b	Or (Ou)
\$a xor \$b	XOR (si \$a OU \$b est TRUE, mais pas les deux en même temps)

- **Opérateurs d'incrément et décrémentation**

++\$a	Pré-incrémente
\$a++	Post-incrémente
--\$a	Pré-décrémente
\$a--	Post-décrémente

# Structures de contrôle - conditionnelle

- **Instruction IF()**

```
if (condition) {  
    Instructions  
}
```

- **Instruction IF() ELSE**

```
if (condition) {  
    Instructions  
} else {  
    Instructions  
}
```

```
$points = 12;  
  
if($points >= 10) {  
    echo 'Réussite';  
}
```

```
if($points >= 10) {  
    echo 'Réussite';  
} else {  
    echo 'Echec';  
}
```

- **Instruction IF() ELSEIF()**

```
if (condition 1) {  
    Instructions  
} elseif (condition 2) {  
    commandes  
}
```

```
if($points >= 14) {  
    echo 'Bonne moyenne';  
} elseif ($points >= 10) {  
    echo 'Moyenne';  
}
```

- **Instruction IF() ELSEIF() ELSE**

```
if (condition 1) {  
    Instructions  
} elseif (condition 2) {  
    Instructions  
} else {  
    Instructions  
}
```

```
if($points >= 14) {  
    echo 'Bonne moyenne';  
} elseif ($points >= 10) {  
    echo 'Moyenne';  
} else {  
    echo 'Echec';  
}
```

# Structures de contrôle - itération

- **Boucle FOR()**

```
for (initialisation; condition; incrémentation) {  
    Instructions  
}
```

- **Boucle FOREACH()**

// tableau indexé

```
foreach ($array as $value){  
    Instructions  
}
```

// tableau associatif

```
foreach ($array as $key => $value){  
    Instructions  
}
```

```
for ($i = 1; $i <= 10; $i++) {  
    echo $i.'<br>';  
}
```

```
$courses = ['PHP' => 120, 'HTML' => 80, 'CSS' => 80];  
foreach ($courses as $course => $period) {  
    echo $course.' : '.$period.'<br>';  
}
```

# Données externes en PHP

- Vous avez la possibilité de récupérer des données provenant de l'extérieur (du script) en utilisant des variables particulières que l'on nomme **Variables Superglobales**. Il en existe un certain nombre que nous verrons par la suite mais nous allons nous intéresser à deux d'entre elles: `$_GET[]` et `$_POST[]`
- Quand on parle de données externes, cela signifie que le contexte dans lequel les données proviennent **n'appartient pas au PHP**. Dans le cas qui nous préoccupe, il s'agit des données postées par un utilisateur via un formulaire ou des données issues d'une URL.
- La portée (voir fonction) de ces variables **est globale** ce qui signifie que vous pouvez les utiliser dans un contexte de fonction utilisateur.

# Superglobale \$\_POST[]

- Cette variable est utilisée pour récupérer les données transmises via un formulaire. Ce n'est valable que si la méthode de transfert des données du formulaire est de type **post**.
- Le formulaire html est déclaré sur le même fichier que le traitement ce qui vous oblige à tester l'existence de la superglobale: **isset()**. Si vous ne procédez pas préalablement par un test d'existence, php va lever un avertissement (warning).

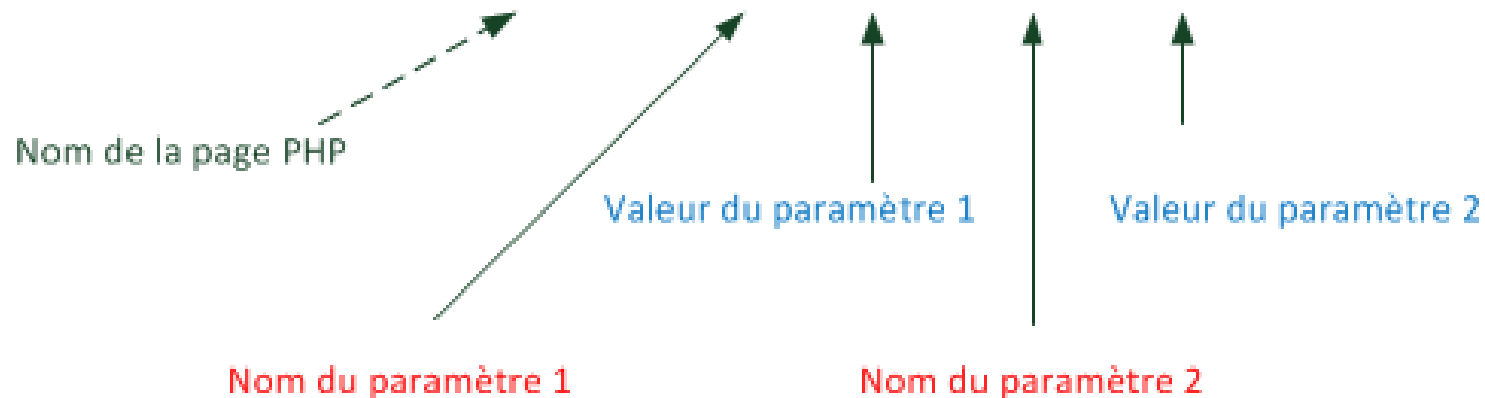
```
<form action="?" method="post">
    <label for="name">Nom:</label>
    <input type="text" id="name" name="name">
    <input type="submit">
</form>
<?php
if(isset($_POST['name'])) {
    echo 'Votre nom est: ' . $_POST['name'];
}
```



# Superglobale \$\_GET[]

- Cette variable est utilisée pour récupérer les données transmises via une URL. Si vous souhaitez transmettre des informations à votre script PHP en passant par l'URL vous allez utiliser dans l'adresse des paramètres qui se présentent sous cette forme:

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



- Le point d'interrogation sépare le nom de la page PHP des paramètres. Ensuite, ces derniers s'enchaînent selon la forme `nom_du_paramètre=valeur`, et sont séparés les uns des autres par le symbole `&`

# Superglobale \$\_GET[]

- Pour récupérer les données de l'url, nous utiliserons en PHP la variable superglobale \$\_GET[].
- Exemple: passage du paramètre en HTML:

```
<a href="?lang=fr">Version française</a>
```

- Exemple: récupération et traitement du paramètre en PHP:

```
if(isset($_GET['lang']) && $_GET['lang'] == 'fr') {  
    // traduire en français  
}
```

# Superglobales \$\_POST[] et \$\_GET[]

- Explication sur le warning:
- Si le test n'est pas réalisé, vous obtenez le message suivant:

**Warning: Undefined array key "lang"**

- La traduction est la suivante: clé "lang" du tableau n'est pas définie.
- En réalité les deux superglobales \$\_GET et \$\_POST sont des tableaux associatifs en PHP. La clé correspond au paramètre et la valeur à la donnée passée.

Nom du tableau et sa clé	Sa valeur
\$_GET['lang']	Fr
\$_POST['name']	John Doe

# Les tableaux - Array

Rappel

Opérations de base

Trier

Conversion

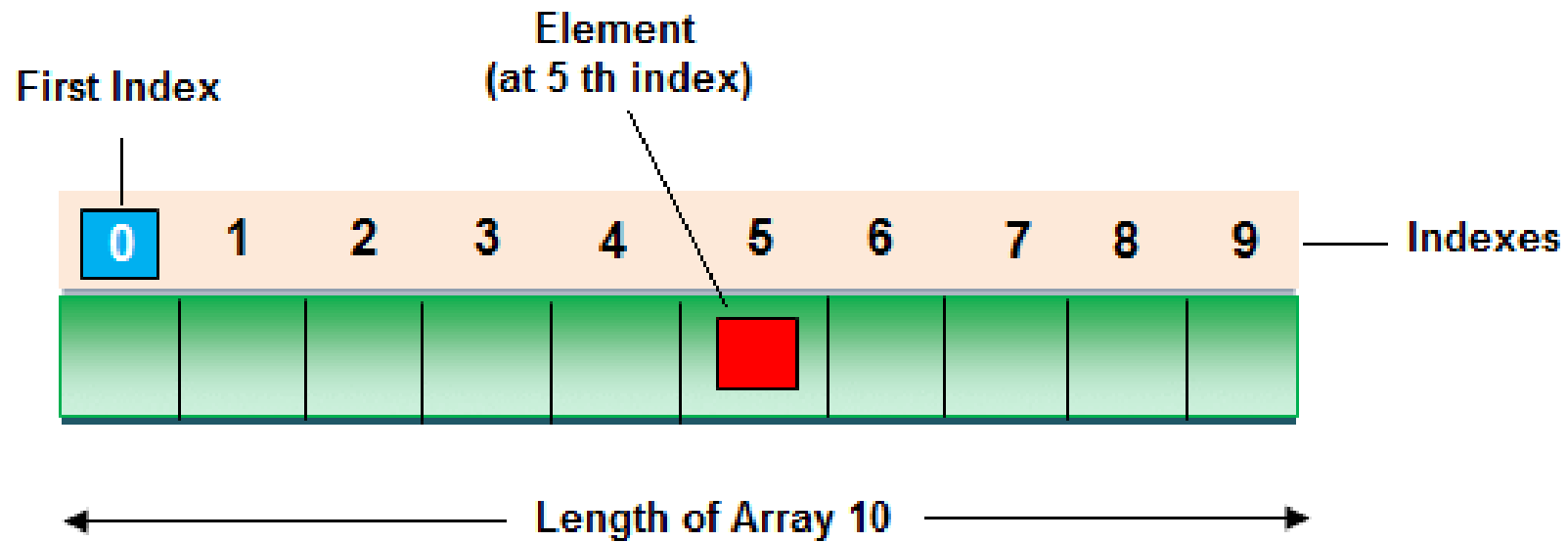
Extraction

Tableaux multidimensionnels

# Rappel

- Contrairement à une variable classique qui ne stocke qu'une seule valeur, un tableau, en revanche, est une variable stockant un ensemble ou une série de valeurs. Un même tableau peut contenir de nombreux éléments, chacun d'eux pouvant être une valeur unique, comme un texte ou un nombre, ou un autre tableau. Un tableau comprenant d'autres tableaux est dit "multidimensionnel".
- Dès lors que des informations sont enregistrées dans un tableau, elles peuvent être soumises à diverses manipulations d'affichage et de traitement (tris, recherches, parcours, calculs, affichages groupés...) L'ensemble des informations enregistrées dans un tableau peut être manipulé comme s'il s'agissait d'une seule entité.

# Tableau unidimensionnel



# Opérations sur les tableaux

- Afficher des informations sur le tableau

**print\_r()**

- Taille d'un tableau

**count()** compte le nombre d'éléments d'un tableau. **sizeof()** est un alias.

- Recherche d'un élément

**in\_array(\$needle, \$haystack)** indique si une valeur appartient à un tableau. Needle est la valeur cherchée et haystack est le tableau. la comparaison est faite en tenant compte de la casse.

- Recherche l'existence d'une clé

**array\_key\_exists(\$key, \$array)** Vérifie si une clé existe dans un tableau.

- Recherche la valeur de la clé correspondante

**array\_search(\$needle, \$haystack)**. Recherche dans un tableau la clé associée à une valeur.

Calcule la somme des valeurs

**array\_sum(\$array)**

## Examples

```
$person = ['name' => 'John Doe', 'city' => 'New-York',  
'job' => 'unknown', 'living' => false];
```

```
var_dump(count($person)); // int(4)
```

```
var_dump(in_array('John Doe', $person)); // bool(true)
```

```
var_dump(array_key_exists('city', $person)); // bool(true)
```

```
var_dump(array_search('John Doe', $person)); // string(4)
```

```
"name"
```



# Opérations sur les tableaux

- Remplir avec des valeurs consécutives
  - range (\$start , \$end [, \$step = 1 ] )** Crée un tableau contenant un intervalle d'éléments.
  - **Start:** première valeur de la séquence.
  - **End:** la séquence se termine lorsque la valeur end est atteinte.
  - **Step:** Si une valeur est donnée au paramètre step, il sera utilisé comme valeur incrémentale entre les éléments de la séquence. step doit être exprimé comme un nombre entier positif. S'il n'est pas spécifié, step vaut par défaut 1.

```
$num = range(1, 5);  
var_dump($num); // array(5) { [0]=> int(1) [1]=> int(2) [2]=> int(3)  
[3]=> int(4) [4]=> int(5) }  
$alpha = range('A', 'G', 2 );  
var_dump($alpha); // array(4) { [0]=> string(1) "A" [1]=> string(1) "C"  
[2]=> string(1) "E" [3]=> string(1) "G" }
```

# Opérations sur les tableaux

- Récupération aléatoire d'éléments

**array\_rand (\$array [, \$num = 1 ] )** Prend une ou plusieurs valeurs, au hasard dans un tableau.

- Array: le tableau d'entrée.
  - Num: spécifie le nombre d'entrées que vous voulez récupérer.
- Lorsque vous ne récupérez qu'une seule entrée, la fonction array\_rand() retourne la clé d'une entrée choisie aléatoirement. Sinon, un tableau de clés d'entrées aléatoires sera retourné.

```
$city = ['New-York', 'Budapest', 'Riga', 'La Havane', 'Berlin'];  
var_dump($city[array_rand($city)]); // string(4) "Riga"
```

# Trier les tableaux

- Tableaux indexés

**sort(\$array)** Les éléments seront triés du plus petit au plus grand.

**rsort(\$array)** Effectue un tri en ordre décroissant (du plus grand au plus petit)

- Tableaux associatifs

**asort(\$array)** Trie le tableau de telle manière que la corrélation entre les index et les valeurs soit conservée. L'usage principal est lors de tri de tableaux associatifs où l'ordre des éléments est important.

**arsort(\$array)** Trie un tableau en ordre inverse et conserve l'association des index.

**ksort(\$array)** Trie un tableau suivant les clés

**krsort(\$array)** Trie un tableau en sens inverse et suivant les clés

# Conversion – explode() et implode()

- Une chaîne en tableau

**explode (\$delimiter , \$string)** Découpe une chaîne en tableau

- Delimiter: le séparateur entre les données (par défaut, une chaîne vide)
- String: la chaîne initiale.

- Un tableau en chaîne

**implode (\$glue, \$pieces )** Rassemble les éléments d'un tableau en une chaîne.

- Glue: par défaut, une chaîne vide.
- Pieces: le tableau de chaînes à rassembler.

```
$hobbies = 'lecture, voyage, sports, histoire';  
$hobbies = explode(',', $hobbies);  
var_dump($hobbies); // array(4) { [0]=> string(7) "lecture" [1]=>  
string(7) " voyage" [2]=> string(7) " sports" [3]=> string(9) " histoire"  
}
```

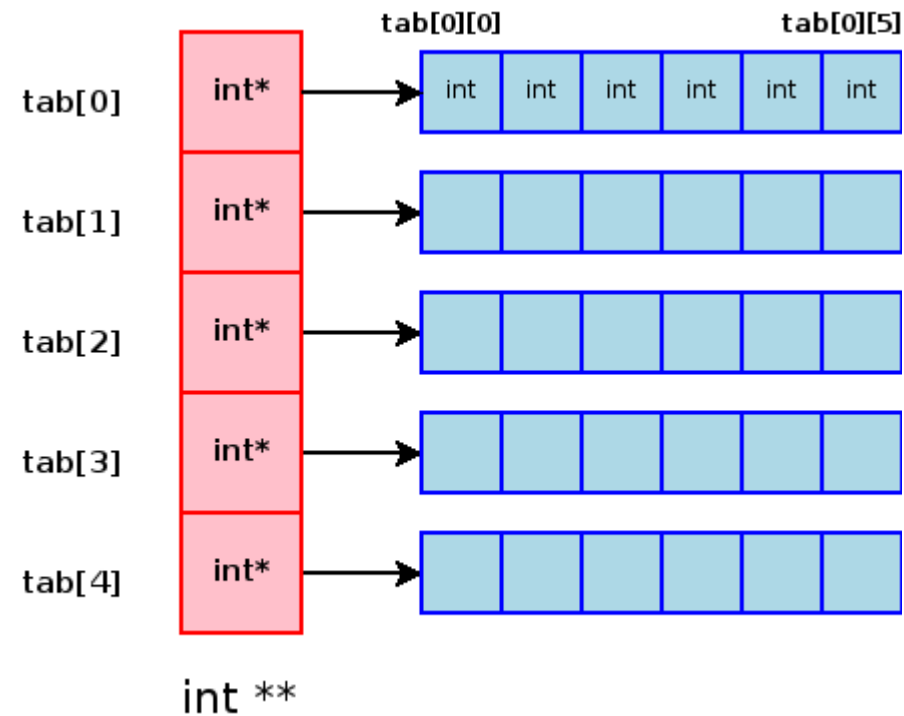
## Extraction dans des variables – list() et extract()

- Il est possible de récupérer le contenu d'un tableau dans des variables en une seule opération. C'est particulièrement intéressant lorsqu'une fonction retourne un tableau.
- Pour un tableau indexé:  
`list(var1, var2, ...) = $tab;`
- Pour un tableau associatif:  
`$tab = ['nom' => 'Doe', 'prenom' => 'John'];`  
`extract($tab); // Crée les variables $nom et $prenom`

# Tableaux multidimensionnels

- Un tableau multidimensionnel est un tableau contenant lui même un ou plusieurs autres tableaux en valeurs. Nous allons pouvoir avoir plusieurs « dimensions » de tableaux multidimensionnels.
  - On appelle tableau à deux dimensions un tableau contenant lui même un ou plusieurs autres tableaux.
  - On appelle tableau à trois dimensions un tableau contenant un ou plusieurs tableaux contenant eux mêmes un ou plusieurs tableaux.
- PHP sait tout à fait travailler avec des tableaux à 2, 3, 4, 5 dimensions et même plus. Cependant, il est conseillé de ne pas stocker trop de tableaux dans d'autres, tout simplement car à partir d'un moment cela devient incompréhensible.

# Tableau à deux dimensions – tous les deux sont indicés



Array

```
(  
  [0] => Array  
    (  
      [0] => John Doe  
      [1] => New-York  
      [2] => Director  
    )  
  
  [1] => Array  
    (  
      [0] => Dominique Servais  
      [1] => Liège  
      [2] => Teacher  
    )  
  
  [2] => Array  
    (  
      [0] => Jon Snow  
      [1] => The Wall  
      [2] => Lord commandant  
    )  
)
```



# Tableau à deux dimensions – le deuxième est associatif

## Tableau Multidimensionnel

Array

(

[0] => Array

(

[prenom] => Julien

[nom] => Cottet

)

[1] => Array

(

[prenom] => Nicolas

[nom] => Lafaye

)

)

<u>indice</u>	<u>valeur</u>
0	Array
1	Array

<u>indice</u>	<u>valeur</u>
prenom	Julien
nom	Cottet

<u>indice</u>	<u>valeur</u>
prenom	Nicolas
nom	Lafaye

# Création et parcours

```
// DECLARATION
$members = [
    ['John Doe', 'New-York', 'Director'],
    ['Dominique Servais', 'Liège', 'Teacher'],
    ['Jon Snow', 'The Wall', 'Lord commandant']
];
```

```
// PARCOURS
echo '<br>';
foreach ($members as $key) {
    foreach ($key as $key1) {
        echo $key1.' &nbsp;';
    }
    echo '<br>';
}
```

La première boucle foreach va boucler sur les valeurs de notre tableau principal (c'est à dire sur nos sous tableaux) et va contenir une deuxième boucle qui va boucler sur les valeurs de chaque sous tableau.

# Structures de contrôle

Syntaxe alternative

Structures conditionnelles

Structures itératives

# Syntaxe alternative

- PHP propose une autre manière de rassembler des instructions à l'intérieur d'un bloc, pour les structures de contrôle if, while, for, foreach et switch. Dans chaque cas, le principe est de remplacer l'accolade d'ouverture par deux points (:) et l'accolade de fermeture par, respectivement, endif, endwhile, endfor, endforeach, ou endswitch. Cette manière de faire est surtout intéressante dans le cas d'écriture HTML entre les blocs.

- Exemples de syntaxe

```
<?php if ($a == 5): ?>
```

```
A égal 5 // HTML
```

```
<?php endif; ?>
```

```
if ($a == 5):  
    echo "a égale 5";  
elseif ($a == 6):  
    echo "a égale 6";  
else:  
    echo "a ne vaut ni 5 ni 6";  
endif;
```

On sort de PHP

On reste en PHP mais avec la syntaxe alternative

```
if ($login == 'john.doe') {  
    echo '<p>' . $login . ' est identifié comme  
administrateur</p>';  
    echo '<a href="admin.php">Admin</a>';  
} else {  
    echo '<p>' . $login . ' n\'est pas autorisé</p>';  
}
```

```
<?php if ($login == 'john.doe') : ?>  
    <p><?= $login ?> est identifié comme  
administrateur</p>  
    <a href="admin.php">Admin</a>  
<?php else : ?>  
    <p><?= $login ?> n'est pas autorisé</p>  
<?php endif; ?>
```

# Les structures conditionnelles

L'instruction IF (première partie)

L'opérateur ternaire

L'opérateur de nullité

L'instruction switch

# L'opérateur ternaire

- Appelé aussi affectation conditionnelle. Il est utilisé pour remplacer la syntaxe if et else dans certaines conditions et ainsi obtenir un code plus simple et plus court.
- Syntaxe:

**(expr1) ? (expr2) : (expr3)**

**(condition) ? (valeur si vrai) : (valeur si faux)**

Renvoie la valeur de l'expression expr2 si l'expression expr1 est vraie, et l'expression expr3 si l'expression expr1 est fausse.

ou

Le ? vérifie si la condition testée (expr1) est vraie ou fausse. Si elle est vraie, PHP retourne la valeur suivant le ?, si elle est fausse, PHP retourne ce qui se trouve après le :

## Exemple comparatif entre if et ternaire

```
//IF
if ($moyenne >= 10) {
    echo 'Elève admis';
} else {
    echo 'Elève non admis';
}

//TERNAIRE
echo ($moyenne >= 10) ? 'Elève admis' : 'Elève non admis';
```



# L'opérateur de nullité

- Opérateur disponible depuis la version 7 de PHP. Il est la suite logique du ternaire et permet d'affecter une valeur en testant si une variable existe et ne vaut pas "nul". Utile pour tester une variable de formulaire ou un paramètre de lien.
- Syntaxe:

**(expression 1) ?? Expression 2;**

```
//IF
if(!isset($_GET['page'])) {
    $_GET['page'] = 'home';
}
//NULLITE
$_GET['page'] = $_GET['page'] ?? 'home';
```

# L'instruction switch

- L'instruction **Switch** est prévue pour tester l'égalité d'une variable quand plusieurs données doivent être testées. Dans le cas des comparateurs logiques vous devez écrire le booléen **true** dans les parenthèses
- Cette instruction permet de faire plusieurs tests sur la valeur d'une variable, ce qui évite de faire plusieurs `elseif` et simplifie ainsi la lecture du code.
- Les parenthèses qui suivent le mot-clé **switch()** indiquent une expression dont la valeur est testée successivement par chacun des cases. Lorsque la valeur correspond à un case, la suite d'instructions est exécutée jusqu'à la fin du switch ou lors de l'apparition d'un `break`. Si aucune correspondance n'est trouvée, alors le code est exécuté à partir du mot-clé `default`.

## Syntaxe:

```
switch ($var) {  
    case condition 1:  
        instruction;  
        break;  
    case condition 2:  
        instruction;  
        Break;  
    default:  
        instruction;  
}
```

## Exemple avec switch

```
$level = 'secrétariat';  
switch ($level) {  
    case 'directeur':  
        echo 'Administration niveau 1';  
        break;  
    case 'sous-directeur':  
        echo 'Administration niveau 2';  
        break;  
    case 'secrétariat':  
        echo 'Administration niveau 3';  
        break;  
    default:  
        echo 'Accès refusé';  
}
```

# Structures itératives

Boucle FOR() (cours initiation et rappel)

Boucle WHILE()

Boucle DO ... WHILE()

BREAK et CONTINUE

# Boucle WHILE()

- Pour répéter une série d'instructions, la boucle while ou "tant que" en français vous sera utile.
- Syntaxe:

```
while (expression) {  
    instruction;  
    Incrémentation;  
}
```
- Cela signifie que tant que l'expression sera **VRAIE**, l'instruction sera exécutée. Il faut donc s'assurer que l'expression passe obligatoirement à **FAUX** à un moment donné, sinon le script bouclera à l'infini et ne se terminera que lorsque le temps maximal d'exécution d'un script sera atteint.
- La boucle **for** a toutes ses déclarations (initialisation, condition, itération) en haut du corps de la boucle. Inversement, dans la boucle **while**, seules l'initialisation et la condition se trouvent en haut du corps de la boucle et l'itération peut être écrite n'importe où dans le corps de la boucle.

# Boucle WHILE()

Le programme commence par tester si la condition est vraie. La boucle **while** exécute alors le code jusqu'à ce que la condition devienne fausse. Si la condition est fausse dès le départ, le code du bloc ne sera jamais exécuté.

Exemple simple:

```
$i = 1;  
while ($i <= 10) {  
    echo $i++;  
}
```

# Boucle DO ... WHILE()

- Une variante de la boucle while est la boucle do...while. La différence réside dans ce que le test est fait après les instructions. Ainsi, la série d'instructions est obligatoirement effectuée au moins une fois avant d'être testée.

- Syntaxe:

```
do {  
    instruction1;  
    instruction2;  
} while (expression);
```

## Exemple simple

```
$i = 1;  
do {  
    echo $i;  
    $i++;  
}  
while ($i <= 10);
```

# Les instructions BREAK et CONTINUE

- Deux instructions permettent de contrôler les sorties de boucles (for, while...). `break` est une instruction qui fait sortir de la boucle, alors que `continue` passe à l'itération suivante sans `exécuter` les instructions d'après.

- Exemples:

- Boucle habituelle

```
for ($i = 0; $i < 10; $i++) {  
    echo $i; // va afficher 0,2,4,6,8  
    $i++;  
    echo $i; // va afficher 1,3,5,7,9  
}  
//Résultat: 0123456789
```

Cette simple boucle affiche tous les chiffres de 0 à 9, mais affiche deux chiffres à chaque itération.



# Les instructions BREAK et CONTINUE

- Boucle avec instruction break

```
for ($i = 0; $i < 10; $i++) {  
    echo $i;  
    $i++;  
    if ($i > 4) break;  
    echo $i;  
}  
//Résultat: 01234
```

La deuxième boucle, semblable à la première, utilise l'instruction break. À la première itération les chiffres 0 et 1 sont affichés. À la seconde les chiffres 2 et 3 sont affichés. Puis 4 est affiché. Enfin, après l'incréméntation \$i vaut 5 (la condition est vérifiée) et le programme sort définitivement de la boucle.

# Les instructions BREAK et CONTINUE

- Boucle avec l'instruction continue

```
for ($i = 0; $i < 10; $i++) {  
    echo $i;  
    $i++;  
    if ($i > 4) continue;  
    echo $i;  
}  
//Résultat: 0123468
```

À la première itération les chiffres 0 et 1 sont affichés. À la seconde les chiffres 2 et 3 sont affichés. Puis 4 est affiché. Enfin, après l'incrémentation \$i vaut 5 (la condition est vérifiée) et la seconde instruction echo n'est pas exécutée (donc 5 n'est pas affiché). Mais une nouvelle itération est réalisée faisant apparaître le chiffre 6, mais pas 7, puis 8, mais pas 9.

# Les fonctions utilisateur

Introduction

Déclaration et appel

Valeurs par défaut

Valeur de retour

Visibilité des variables

L'opérateur de jonction

Typage

Fonction anonyme

# Introduction

- Il existe deux types de fonctions en PHP : les fonctions dites natives, que vous pouvez employer sans faire appel à des bibliothèques (`var_dump()`, `isset()`, etc.) et les fonctions dites utilisateurs, que vous déclarer personnellement ou qui sont définies dans une bibliothèque tierce.
- Une fonction utilisateur peut être résumée à un sous-programme que l'on appelle depuis le programme principal. Contrairement à une instruction simple, une fonction est un ensemble d'instructions qui peuvent être parfois très complexes. On regroupe donc toutes ces instructions en une fonction que l'on pourrait donc assimiler à un sous-programme. Ce groupe d'instructions est lancé à chaque appel de la fonction par le programme principal. Il sera par la suite possible d'exécuter ce bloc en une commande (via le nom de la fonction), au lieu de recopier plusieurs fois le même code.

# Avantages

- Permet de réaliser des traitements spécifiques ou annexes en dehors du script principal.
- Permet une meilleure maintenabilité du code source. Vos fonctions regroupent de tâches bien précises et sont plus facilement accessibles pour une modification ou une correction ultérieure.
- Réutilisation des fonctions ultérieurement. Au fur et à mesure du développement de vos projets vous vous constituez des outils que vous allez réemployer pour les projets à venir. Principe de la boîte à outils et réduction du temps de développement.
- Respect de la convention de programmation "DRY" don't repeat yourself.

# Déclaration de la fonction

- La définition d'une fonction s'appelle déclaration et peut se faire n'importe où dans le code grâce au mot-clé `function`. Il est cependant préférable de regrouper ses fonctions dans des fichiers dédiés et séparés.

- Syntaxe:

```
function nomFonction()  
{  
    // Liste d'instructions;  
}
```

```
function nomFonction ($paramètre1, $paramètre2, ...)  
{  
    // Liste d'instructions ;  
}
```

- Les paramètres représentent sous la forme de variables les données qui seront nécessaires pour l'exécution de la fonction. Il peut y en avoir un, plusieurs, ou même aucun (dans ce cas, on laisse les parenthèses vides). Ils permettent d'utiliser la fonction dans différents contextes.

# Appel de la fonction

- Pour exécuter une fonction, il suffit de l'appeler dans votre script en utilisant son nom et en lui passant les arguments éventuels. L'appel de la fonction peut être réalisé avant la déclaration mais c'est déconseillé.
- Syntaxe:  
`nomFonction (argument1, argument2);`
- Exemple sans paramètre:

## *Déclaration*

```
function premiereFonction()  
{  
    echo "J'aime PHP !";  
}
```

## *Appel*

```
premiereFonction(); // Affiche "J'aime PHP !"
```

# Les paramètres et les arguments de la fonction

- La confusion règne entre le terme paramètres et arguments. Les paramètres désignent les variables dans la signature de la fonction et les arguments représentent les données que le passent à la fonction lors de son utilisation.
- Pour accomplir la tâche pour laquelle elles ont été conçues, la plupart des fonctions requièrent qu'un ou plusieurs arguments leurs soient fournis. Les arguments permettent de passer des données à une fonction (ce sont les valeurs effectives). Les arguments peuvent être des valeurs, de simples variables, mais aussi des tableaux ou des objets.
- Exemple:

```
function display($name)
{
    echo '<p>Content de vous revoir '.$name.'</p>';
}
display('John Doe');
```



- La même fonction avec une instruction de traitement

```
function display($name)
{
    $name = strtoupper($name);
    echo '<p>Content de vous revoir '.$name.'</p>';
}
display('John Doe'); //Content de vous revoir JOHN DOE
```

- Dans cet exemple, on ajoute une fonctionnalité supplémentaire (couche métier) qui traite ou manipule les données qui seront affichées par la suite.
- Un seul paramètre est demandé dans cette fonction. Si la fonction en nécessite plusieurs, on les séparent par une virgule. Des paramètres optionnels peuvent également figurer dans les parenthèses (voir dia suivante)

# Paramètres par défaut ou facultatif

- Un ou plusieurs paramètres peuvent être facultatifs si on leur donne une valeur par défaut. Ils doivent être placés après les paramètres obligatoires.

```
function tva($price, $rate = 21)
{
    echo '<p>Montant TVA: ' . ($price * $rate / 100) . '</p>';
}
```

```
tva(1000); // Montant TVA: 210
tva(1000, 6); // Montant TVA: 60
```

- Par défaut, le taux de tva est de 21% mais peut varier s'il est passé comme argument dans la fonction.

# Arguments nommés

- PHP 8.0.0 introduit les arguments nommés comme extension aux paramètres positionnels existants. Les arguments nommés permettent de passer les données à une fonction en s'appuyant sur le nom du paramètre, au lieu de la position du paramètre.
- Les arguments nommés sont passés en préfixant la valeur avec le nom du paramètre suivi d'un deux-points. Utiliser des mots-clés réservés comme nom de paramètre est autorisé. Le nom du paramètre doit être un identifiant.

```
function toPay(int $price, int $reduce) : int {  
    return $price - $reduce;  
}  
echo toPay(price: 100, reduce: 20); // 80  
echo '<br>';  
echo toPay(reduce: 20, price: 100); // 80
```

# Les arguments nommés

## Les avantages:

L'ordre dans lequel les arguments sont passés à la fonction n'importe plus.

Ecrire un code plus compréhensible parce que la signification des paramètres est auto documentées.

Possibilité d'utiliser les mots réservés (array, ...). Cependant je ne le recommande pas.

# Valeur de retour: return

L'utilisation du mot-clé **return** permet de renvoyer au script une valeur au lieu de l'afficher par l'intermédiaire de la fonction. C'est alors dans le code du script principal que se décidera l'affichage ou l'utilisation de ce résultat.

```
function tva($price, $rate = 21)
{
    $tva = $price * $rate / 100;
    return $tva;
}
$tva = tva(1000);
echo '<b>'.$tva.'</b>';
```

Maintenant que le script récupère la valeur sous la forme d'une variable, vous pouvez la manipuler comme bon vous semble. Sans passer par une variable intermédiaire vous devrez utiliser l'instruction echo suivie du nom de la fonction.

- Une autre manière de faire est de renvoyer directement le résultat du calcul sans passer par une variable.

```
function tva1($price, $rate = 21)
{
    return $price * $rate / 100;
}
$tva = tva1(1000);
echo $tva;
```

- Si le calcul est complexe ou nécessite plusieurs étapes, utilisez la première technique pour des raisons de lisibilité et de conventions.

- Le mot-clé return interrompt l'exécution d'une fonction. L'exécution d'une fonction prend fin, soit parce que toutes ses instructions ont été exécutées, soit parce que le mot-clé return a été atteint.
- Normalement, le mot-clé return s'emploie pour sortir d'une fonction, au milieu de celle-ci, lorsqu'une certaine condition a été vérifiée.

```
function hello($name, $text = "")  
{  
    if(empty($text)){  
        return false;  
    }  
    echo $text.' '. $name;  
}
```

```
hello('John Doe'); // ne renvoie rien
```

# Retourner plusieurs valeurs

- Si vous souhaitez qu'une fonction retourne plusieurs valeurs (données ou variables), le plus simple est d'utiliser un tableau lors du return.

```
function nom_fonction()
{
    ....
    return [ 'contenu1', $variable2, $variable3 ];
    // On retourne les valeurs voulues dans un tableau
}
$retour = nom_fonction();
echo $retour[0], $retour[1], $retour[2];
```

- Une boucle de type foreach() peut également être utilisée.



# Nombre de paramètres variables

- Certaines fonctions natives acceptent un nombre indéfini d'arguments. Avoir ce comportement avec des valeurs par défaut nécessiterait de définir énormément de valeurs par défaut et de gérer autant de conditions dans le code. Même ainsi, le fonctionnement ne serait pas parfait, car vous ne pourriez que définir un grand nombre de paramètres, pas un nombre indéfini.
- Il est possible de travailler avec un nombre réellement indéfini d'arguments à l'aide **l'opérateur de jonction ...** (trois points successifs). Il a été introduit à partir de la version 5.6 de PHP.

```
function students(...$name)
{
    // code de la fonction
}
```

```
students(argument1, argument2, ...);
```

# Exemple avec des paramètres variables

## Exemple

```
function teachers(...$name)
{
    echo '<ul>';
    foreach($name as $key) {
        echo '<li>'.$key.'</li>';
    }
    echo '</ul>';
}

teachers('John', 'Dominique', 'Patrick', 'Jane');
```

Cette syntaxe va créer un tableau avec nos différents arguments. Il faut utiliser une boucle foreach pour parcourir ce tableau.

Notez ici que nous aurions bien pu utiliser un tableau comme argument.

# Visibilité des variables

- Les variables n'ont pas toutes la même **visibilité** dans un script. Par exemple, les variables déclarées dans une fonction ne seront utilisables que dans celle-ci. Inutile donc (par défaut) d'essayer de les appeler en dehors. De la même façon, les variables déclarées dans votre script ne seront pas accessibles dans une fonction. Les deux espaces sont complètement indépendants.
- Une variable a donc **une portée** plus ou moins grande selon l'endroit où elle est définie. Il existe plusieurs niveaux de définition de variable (seulement deux sont évoqués ici):
  - Le niveau **global** permet à une variable d'être visible dans la fonction et à l'extérieur de la fonction.
  - Le niveau **local**, utilisé par défaut, permet de définir une variable qui ne sera visible que dans la fonction en cours.

```
$a = 1; // portée globale

function test()
{
    $b = 2; // portée locale
}
```

```
function level()  
{  
    $level = 'Administrateur';  
    return $level;  
}  
  
level();  
echo $level; // Undefined variable: level
```

Variable déclarée  
dans la fonction

```
$status = 'Administrateur';  
function level()  
{  
    echo 'Votre niveau: ' . $status;  
}  
  
level(); // Error
```

Variable déclarée  
dans le script

```
function level()  
{  
    global $level;  
    $level = 'Administrateur';  
    return $level;  
}  
  
level();  
echo $level; // Administrateur
```

```
$status = 'Administrateur';  
function level()  
{  
    global $status;  
    echo 'Votre niveau: ' . $status;  
}  
  
level(); // Votre niveau: Administrateur
```

Pour utiliser une variable en dehors de son contexte de déclaration, utilisez le mot clé **global**. Pour modifier la portée sur plusieurs variables séparez les par une virgule:

**global \$a, \$b;**

Toutefois, cette pratique est déconseillée. Elle oblige à chaque utilisation de la fonction, la déclaration dans le script de la variable.

# Typage des paramètres

- Pour l'instant, nous n'avons pas encore abordé la notion de typage pour les **arguments** (paramètres) et les **retours** (return) de nos fonctions. A partir de la version 7 de PHP, il est possible (facultatif) de typer ces types de données. Ce procédé s'appelle une **déclaration de type**.
- Certains développeurs critiquent PHP car il est considéré comme un langage faiblement typé. En réalité, l'interpréteur PHP définit en interne le type des variables et les transtypages nécessaires en fonction des besoins et de vos instructions. On parle alors de **typage souple** ou **dynamique**.

# Typage des paramètres

- Une approche souple du typage se révèle très pratique dans de nombreux cas mais une approche stricte permet notamment de:
  - **Faciliter** la détection d'erreurs dans le code.
  - **Restreindre** la quantité de code nécessaire lors de la gestion et la validation des données.
  - **Permettre** facilement l'emploi de bibliothèques tierces dans votre projet (vous savez quels types sont autorisés lors de l'appel des fonctions).

# Typage des paramètres

```
function identity (string $name, int $age)
{
    ...
}
```

- Le type des paramètres est indiqué devant les variables dans la signature de la fonction.
- Les types disponibles sont: int, bool, float, string, array, mixed et object.
- Si vous utilisez une IDE comme PhpStorm, on vous indique le type souhaité lors de l'appel de la fonction. La fonction `var_dump()` vous permettra de vérifier le type des variables.



```
function add($a, $b)
{
    echo $a + $b;
}

add('10cm', 5); // Warning: A non-numeric value encountered
```

Lors de l'appel de la fonction un simple warning est affiché et la fonction retourne la valeur 15.

```
function add(int $a, int $b)
{
    echo $a + $b;
}

add('10cm', 5); // Uncaught TypeError: Argument 1 passed to
add() must be of the type integer, string given.
```

Dans ce cas, une erreur est levée et spécifiée. Aucune valeur de retour ne sera affichée.

# Typage des valeurs de retour

- Il est également possible de spécifier le type de retour d'une fonction. Cela permet de garantir lors de l'appel de la fonction, que l'on obtiendra toujours une donnée de ce type.

```
function add (int $a, int $b) : int  
{  
    ...  
}
```

- Indiquez le type de retour précédé du symbole : après les parenthèses de la fonction.

# Typage stricte

- Dans les exemples précédents et même en spécifiant une déclaration de type, PHP essaie si c'est possible de convertir les données pour pouvoir les manipuler avec le bon type (transtypage interne).

- Exemple:

```
function add(int $a, int $b)
{
    echo $a + $b;
}
add(10, 5); // 15
add('10', 5); // 15
add(10.35, 5); // 15
```

- Dans le cas où la conversion n'est pas possible, PHP lèvera une erreur ou un warning.

# Mise en place d'une approche stricte

- Dans ce mode, PHP ne réalise pas de conversion automatique des valeurs. Un type invalide provoque une *TypeError* même si la valeur aurait pu être convertie. Le typage strict est accessible depuis PHP7.

```
declare(strict_types =1 );
```

```
function add(int $a, int $b)  
{  
    echo $a + $b;  
}
```

```
add(10, 5); // 15
```

```
add('10', 5); // TypeError
```

```
add(10.35, 5); // TypeError
```

- Pour activer le typage strict, l'expression `declare` est utilisée avec la déclaration `strict_types=1`

```
declare(strict_types=1);
```

- Cette déclaration doit être obligatoirement positionnée en premier dans le script.
- La seule exception à cette règle est qu'un entier (integer) peut être passé à une fonction attendant un nombre flottant (float).
- Pour déclarer le type de simples variables (hors fonction) écrivez le `nom de la variable = (type) valeur`.

```
$name = (int) '15';
```

```
var_dump($name); // int(15)
```

Ou les fonctions `settype()` vues en initiation à la programmation.

# Traitements de chaînes

Informations sur une chaîne

Protection et échappement

Conversions et formatages

# Informations sur une chaîne

## 1) Taille d'une chaîne - Strlen (\$string)

Renvoie la taille d'une chaîne de caractères (le nombre de signes, espaces et caractères blancs compris).

```
$str = 'abcdef';  
echo strlen($str); // 6
```

## 2) Lister les mots d'une chaîne - str\_word\_count (\$string)

Compte le nombre de mots utilisés dans une chaîne.

```
$str = 'Pas de commentaire' ;  
echo str_word_count($text); // 3
```

# Informations sur une chaîne

## 3) Position d'une sous-chaîne – strpos()

Retourne la position de la première occurrence dans une chaîne. La valeur FALSE est renvoyée si la chaîne n'est pas trouvée.

```
strpos ($string, $sous-chaîne [, $offset = 0 ] )
```

```
$mail = 'John.doe@gmail.com';  
echo strpos($mail, '@'); // 8
```

```
$mail = '@john.doe@gmail.com';  
echo strpos($mail, '@'); // 0
```

```
echo strpos($mail, '@', 1); // 9
```



# Protection et échappement

## 4) Conversion des changements de lignes - nl2br(\$string)

- En HTML, le caractère de fin de ligne est un caractère dit caractère blanc, ayant la même signification qu'un espace. Un texte classique de plusieurs paragraphes sera rendu comme un seul paragraphe une fois inséré dans du HTML.
- La balise <br> est le moyen, en HTML, de forcer un passage à la ligne. Pour permettre à un texte de s'afficher dans une page HTML avec des fins de lignes là où c'était prévu, il faudra utiliser la fonction nl2br qui convertit chaque caractère de fin de ligne en <br>.

```
$text = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas at odio id quam vestibulum blandit.  
  
Cras lorem mi, tristique non faucibus id, dapibus et tortor. Integer pharetra fermentum ligula, et accumsan nibh varius at. Curabitur elementum semper elit.';  
  
echo $text; // pas de retour à la ligne  
echo '<hr>';  
echo nl2br($text); // gère le retour à la ligne
```

# Protection et échappement

## 5) Suppression des balises HTML et PHP d'une chaîne – strip\_tags()

Cette fonction est très pratique, car elle vous permet de limiter les balises HTML dans une variable envoyée par un utilisateur. Sur un forum, par exemple, vous pourriez n'autoriser que le gras et l'italique (<b> et <i>).

`strip_tags ($str [, string $allowable_tags ] )`

```
$_POST['comment'] = 'Votre <em>site</em> contient des <b><del>erreurs</del></b>';  
  
echo $_POST['comment']; // Toutes les balises  
echo '<br>';  
echo strip_tags($_POST['comment']); // Aucune balises  
echo '<br>';  
echo strip_tags($_POST['comment'],'<em><b>'); // em et b autorisées  
echo '<br>';
```

# Manipulations des chaînes

## 6) Recherche d'une sous-chaîne – strstr()

La fonction `strstr()` fonctionne de manière similaire à `strpos()` mais retourne le reste de la chaîne à partir de la position repérée, au lieu de la position elle-même. Elle a ses équivalents `stristr()` pour l'analyse sans prise en compte de la casse, et `strrchr()` pour l'analyse de droite à gauche.

`strstr ($haystack , $needle, $before_needle = FALSE)`

```
echo strstr($mail, '@'); // @gmail.com
echo '<hr>';
echo strstr($mail, '@', true); // gmail.com
```

# Manipulations des chaînes

## 7) Récupérer une sous-chaîne à partir d'une position – substr()

Il est possible de récupérer une sous-partie d'une chaîne de caractères si on en connaît la position de départ. La fonction `substr()` prend en paramètres une chaîne de caractères référence, une position de départ et une longueur. Cette fonction renvoie les caractères à partir de la position initiale jusqu'à atteindre la longueur définie.

`substr ( string $string , int $start [, int $length ] )`

```
$text = 'PHP 8.2 dernière version';  
echo 'Vous utilisez PHP '.substr($text, 4, 1); // Vous utilisez PHP 8
```

# Manipulations des chaînes

## 8) Remplacer une chaîne – str\_replace()

La fonction `str_replace()` permet de remplacer un motif dans une chaîne. Le premier argument est la sous-chaîne à rechercher, le deuxième argument est la chaîne de remplacement, et le troisième est la chaîne de référence où faire le remplacement.

`str_replace ($search , $replace , $subject)`

```
$mail = 'john.doe@hotmail.com';  
echo str_replace('hotmail.com', 'gmail.com', $mail); // john.doe@gmail.com
```

# Manipulations des chaînes

9) Supprimer les espaces (ou d'autres caractères) en début et fin de chaîne – trim()

trim (\$str, ['characters'])

```
$name = '-   John   Doe   -';  
echo trim($name); // - John Doe -  
echo '<br>';  
echo trim($name, '-'); // John Doe  
echo '<br>';  
echo rtrim($name, '-'); // - John Doe
```

Voir aussi ltrim() et rtrim()

# Manipulations des chaînes

## Changement de casse

- Le changement de casse est la transformation de caractères minuscules en majuscules ou inversement. Ces fonctions sont en général appelées juste avant l’affichage pour normaliser la présentation de données. On peut aussi s’en servir avant d’insérer des informations dans une base de données pour normaliser les enregistrements.

### 10) Renvoie une chaîne en minuscules

`strtolower ($string)`

### 11) Renvoie une chaîne en majuscules

`strtoupper ($string)`

### 12) Met le premier caractère en minuscule

`lcfirst (string $str)`

# Manipulations des chaînes

13) Met le premier caractère en majuscule

`ucfirst ($str)`

14) Met en majuscule la première lettre de tous les mots

`ucwords ($str)`



# Traitements des dates

# Introduction

- Depuis toujours, les fonctions de date utilisent la notion de timestamp Unix pour le calcul et la conversion. Il s'agit du nombre de secondes écoulées depuis le 1er Janvier 1970 à 0h00. Cette date correspond au début de l'ère linux. En effet, les systèmes Unix enregistrent la date et l'heure courantes sous la forme d'un entier sur 32 bits.
- Sur la plupart des systèmes, il est toutefois possible d'accéder aux dates entre 1901 et 1970 en utilisant un entier signé négatif pour le nombre de secondes. La limite dans les deux cas est 2038.
- Inconvénients:
  - Données numériques qui ne correspondent pas directement à une date
  - Timestamp trop long avec le temps
  - Ne tient pas compte des fuseaux horaires
  - Ne tient pas compte de l'heure d'été...

# Pratique procédurale

- Vous allez encore retrouvé ou devoir manipuler des dates avec l'ancienne procédure et ses fonctions. Mais actuellement la tendance est d'employer une classe native en PHP du nom de `DateTime()`.
- Anciennes fonctions
  - `time()`: retourne l'heure courante, mesurée en secondes depuis le début de l'époque UNIX, (1er janvier 1970 00:00:00 GMT).
  - `date(format, timestamp)`: formate une date/heure locale
  - `strtotime(time,now)`: Transforme un texte en timestamp
  - `date_diff()`: calcule la différence entre deux dates
  - `mktime()`: Retourne le timestamp UNIX d'une date
  - ...

# Pratique objet

- L'approche à base de timestamps pour manipuler les dates a ses limites et a tendance à laisser la place à une seconde approche, plus récente, la classe DateTime().

- **Style procédural**

```
$date = date_create();  
echo date_format($date, 'Y-m-d');
```

- **Style orienté objet**

```
$date = new DateTime();  
echo $date->format('Y-m-d ');
```

# Classe DateTime

- Pour manipuler une date à l'aide de la classe, vous devez créer un objet qui vous permettra de représenter la date. Ce processus se nomme instantiation. Pour ce faire, on utilise la syntaxe d'affectation avec le mot clé **new**. `$date` est un objet représentant une instance de la classe.

**`$date = new DateTime();`** // création de l'objet

- Une fois qu'un objet a été instancié, on peut commencer à utiliser ses attributs (variables et constantes) et ses méthodes (fonctions). On accède à un attribut ou une méthode grâce à l'opérateur `->`

**`$date->format('d/m/Y');`** // formatage de la date en cours (jj/mm/aaaa)

- Ici, on invoque la méthode `format()` sur l'objet `$date`. Cette méthode permet de formater une date pour obtenir une chaîne de caractères la représentant.

- L'instanciation sans paramètres permet d'obtenir un objet correspondant à l'instant présent. Il est également possible de spécifier un paramètre qui fixera la date et l'heure de l'objet créé.

```
$date = new DateTime('2018-01-01');
```

```
$date = new DateTime('12:20:40');
```

```
$date = new DateTime('next monday');
```

```
$date = new DateTime('2018-01-01 12:00:00 Europe/Paris');
```

# Formater une date

- Comme nous l'avons vu en guise d'introduction, la méthode `format()` permet de formater une date, pour obtenir une chaîne la représentant en totalité ou en partie.
- Quelques exemples:  
    `$date = new DateTime();`  
    `echo $date->format('d/m/Y'); // jj/mm/aaaa`  
    `echo $date->format('H:m'); // hh:mm`  
    `echo $date->format('W'); // Numéro de la semaine dans l'année`  
    `echo $date->format('Z'); // Jour de l'année`

# Modifier une date

- Pour modifier une date (ajouter ou soustraire) vous disposez de deux méthodes:

- **modify(\$string)**

```
$dt = new DateTime();  
$dt->modify('+1 month');  
$dt->modify('-1 day');
```

- **Add(\$string) ou sub(\$string)**

```
$dt2 = new DateTime();  
$dt2->add(new DateInterval('P1D'));  
$dt2->sub(new DateInterval('P4M'));
```

Dans ce cas, vous devez utiliser une classe supplémentaire:

**DateInterval()**. Les formats sont les mêmes que la fonction procédurale `date()`.



# Calculer une différence entre deux dates

- **Méthode: diff(datetime)**

datetime correspond à la date à comparer

```
$date1 = new DateTime('2010-10-11');
```

```
$date2 = new DateTime('2018-10-13');
```

```
$interval = $date1->diff($date2);
```

```
echo $interval->format('%y ans %m mois et %d jours'); // 8 ans 0 mois et 2 jours
```

# Traitement des valeurs numériques

# Fonctions mathématiques (base)

`round (nombre [, précision])`

La fonction `round()` retourne la valeur arrondie du nombre passé en paramètre à l'entier le plus proche. Si vous définissez une précision via le second paramètre, la fonction arrondira alors à cette précision.

`ceil (nombre)`

La fonction `ceil()` retourne l'entier supérieur du nombre passé en paramètre. Utiliser `ceil()` sur un entier ne sert à rien.

`floor (nombre)`

La fonction `floor()` retourne l'entier inférieur du nombre passé en paramètre.

`mt_rand ([minimum,maximum])`

Génère une valeur aléatoire comprise entre un minimum et un maximum donné.

# Fonctions de transformation

`number_format ($number [, $decimals = 0 ])`

Formate un nombre pour l'affichage:

number: Le nombre à formater.

decimals: Définit le nombre de décimales.

dec\_point: Définit le séparateur pour le point décimal.

thousands\_sep: Définit le séparateur des milliers.

```
$number = 1234.56;
```

```
echo number_format($number); // 1,235
```

```
// Notation française
```

```
echo number_format($number, 2, ',', ' '); // 1 234,56
```

# Transfert de fichiers vers le serveur

Formulaire

Récupération des informations sur le fichier

Transfert du fichier

# Formulaire

- L'upload de fichiers sur un serveur passe évidemment dans un premier temps par un formulaire.

```
<!-- Le type d'encodage des données, enctype, doit être spécifié -->
<form enctype="multipart/form-data" action="?" method="post">
  <label for="userfile">Votre fichier: </label>
  <input id="userfile" name="userfile" type="file">
  <input type="submit" value="Envoyer votre fichier">
</form>
```

# Récupération des informations sur le fichier

- Les fichiers envoyés avec un formulaire se traitent différemment des autres données. La différence la plus visible est l'utilisation de la **superglobale** `$_FILES[]`. C'est avec cette dernière que vous pourrez accéder aux fichiers téléchargés.
- Il s'agit d'un tableau associatif à deux niveaux. Le premier paramètre doit être le nom du champ de formulaire (fichier dans notre exemple), le second paramètre concerne plusieurs informations sur le fichier transmis.

# La superglobale \$\_FILES[]

- `$_FILES["file"]["name"]` : contient le nom qu'avait le fichier sur le poste client.
- `$_FILES["file"]["type"]` : contient le type MIME initial du fichier et permet un Contrôle
- `$_FILES["file"]["size"]` : donne la taille réelle en octet du fichier transféré.
- `$_FILES["file"]["tmp_name"]` : donne le nom temporaire que le serveur attribue automatiquement au fichier en l'enregistrant dans le répertoire tampon.
- `$_FILES["file"]["error"]` : donne le code d'erreur éventuel associé au fichier téléchargé.



## Exemple pour récupérer les informations du fichier

```
<?php
if(isset($_FILES['userfile'])) {
    echo 'Nom d\'origine: ' . $_FILES['userfile']['name'];
    echo '<br>';
    echo 'Taille du fichier: ' . $_FILES['userfile']['size'];
    echo '<br>';
    echo 'Type de fichier: ' . $_FILES['userfile']['type'];
    echo '<br>';
    echo 'Adresse tmp (serveur): ' . $_FILES['userfile']['tmp_name'];
    echo '<br>';
    echo 'Code erreur ' . $_FILES['userfile']['error'];
}
```

# Transfert du fichier

- PHP crée un fichier temporaire sur le serveur pour chaque fichier envoyé. Ce fichier étant détruit automatiquement à la fin de l'exécution du script, vous avez généralement besoin de le sauvegarder à un emplacement définitif. Cet enregistrement peut se faire par l'intermédiaire de la fonction `move_uploaded_file()`.
- PHP déplace alors le fichier depuis son emplacement temporaire sur le serveur vers la destination finale que vous lui aurez allouée. Cette fonction prend deux paramètres : le nom du fichier téléchargé (temporaire) et l'adresse de destination avec le nom définitif.

```
move_uploaded_file($_FILES['userfile']['tmp_name'], 'uploads/' .  
$_FILES['userfile']['name']);
```

## Exemple: formulaire et transfert

```
<form enctype="multipart/form-data" action="?" method="post">
  <label for="userfile">Votre fichier: </label>
  <input id="userfile" name="userfile" type="file">
  <input type="submit" value="Envoyer votre fichier">
</form>
```

```
<?php
if(isset($_FILES['userfile'])) {
    move_uploaded_file($_FILES['userfile']['tmp_name'],
'uploads/' . $_FILES['userfile']['name']);
}
```

# Amélioration du script

- Le script précédent est élémentaire et doit être amélioré pour des raisons de sécurité.
- Testez la taille du fichier à l'aide `$_FILE['file']['size']`
- Testez ou autorisez certaines extensions de fichier notamment à l'aide d'une liste blanche (tableau)
- Renommez le nom du fichier pour éviter les doublons
- Encadrez la fonction `move_uploaded_file()` d'une conditionnelle pour avertir du succès ou non du transfert
- Utilisez le code erreur pour indiquer le type d'erreur lors du transfert
- ...

## Exemple avec conditionnelle

```
if (move_uploaded_file($_FILES['userfile']['tmp_name'], 'uploads/'  
    . $_FILES['userfile']['name'])) {  
    echo '<p>Transfert réussi</p>';  
} else {  
    echo '<p>Erreur lors du transfert</p>';  
}
```

# Les cookies

Introduction

Création

Lecture

Modification

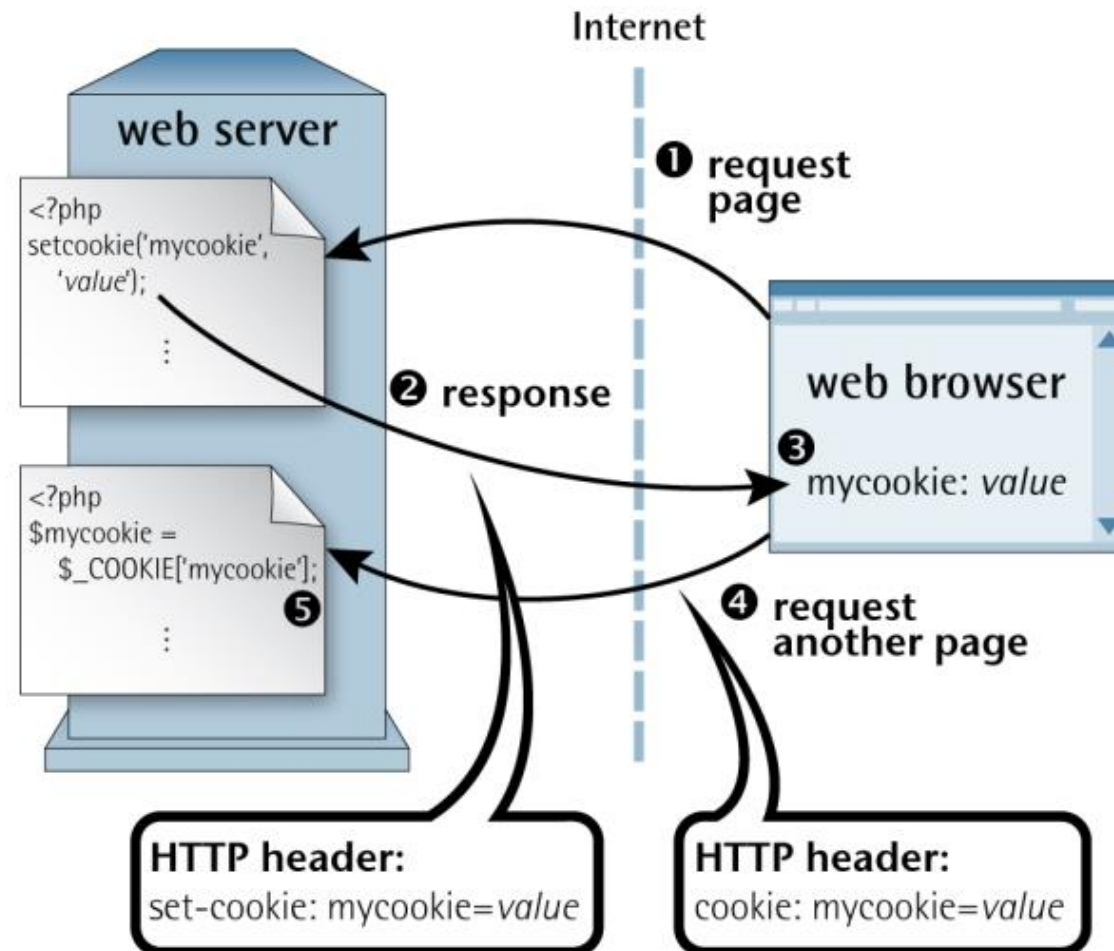
Destruction

Limitation

# Introduction

- Les cookies sont de petits fichiers au format texte enregistrés côté client et qui permettent de retenir des informations sur un utilisateur. vous pouvez enregistrer des données qui seront associées à un visiteur particulier. Les utilisations les plus fréquentes des cookies sont :
  - se souvenir du nom ou d'un login d'un utilisateur pour lui éviter de le ressaisir lors de sa prochaine authentification ;
  - se souvenir des informations saisies dans un formulaire pour éviter de les redemander ou pour pré-remplir le formulaire la prochaine fois ;
  - identifier chaque utilisateur de façon unique lors de ses visites à des fins statistiques.
- Un site donné ne peut écrire que 20 cookies sur un même poste client. Chacun d'eux ne doit pas dépasser 4 Ko, ce qui empêche le stockage d'information de taille importante.

# Fonctionnement





# Création d'un cookie

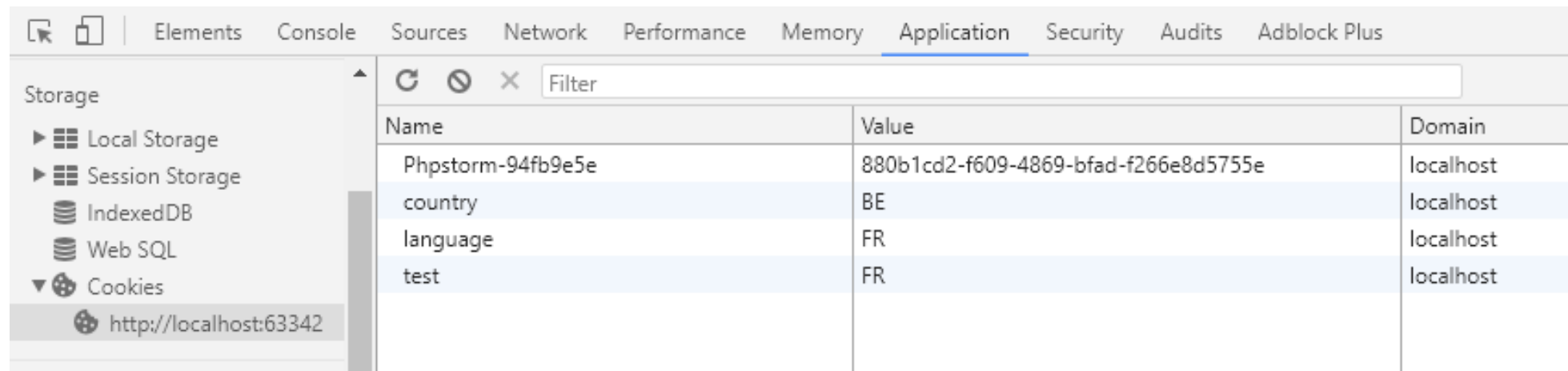
- Pour écrire un cookie, comme pour envoyer des en-têtes au moyen de la fonction `header()`, il est impératif qu'aucun contenu HTML n'ait été envoyé au poste client avant l'écriture du cookie.
- Pour écrire un cookie, vous utilisez la fonction `setcookie()`, dont la syntaxe allégée est la suivante : **`Setcookie(name, value, expire, ...)`**
- **name:** est une chaîne définissant le nom du cookie. Ce nom obéissant aux mêmes règles de nommage que les variables, il faut éviter les caractères spéciaux. Ce nom sert à identifier le cookie pour les opérations de lecture de leur contenu.
- **value:** contient la valeur associée au cookie. Il s'agit d'une chaîne de caractères, même pour un nombre.
- **expire:** datefin est un entier qui permet de stocker un timestamp UNIX exprimé en seconde définissant la date à partir de laquelle le cookie n'est plus utilisable. Si ce paramètre est omis, le cookie n'est valable que pendant le temps de connexion du visiteur sur le site. Pour définir cette date, vous utilisez le plus souvent la fonction `time()`, qui donne le timestamp en cours, auquel vous ajoutez la durée désirée par un nombre de seconde.

# Exemple

```
<?php
// cookie expirant à la fermeture de la session (navigateur)
setcookie('language', 'FR');
// cookie expirant dans une heure
setcookie('country', 'BE', time()+3600);
?>
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Cookies</title>
</head>
<body>
<h3>Deux cookies ont été envoyés</h3>
</body>
</html>
```

# Retrouver et lire les cookies sur son PC

- Affichez le "devtools" dans votre navigateur (F12)
- Activez l'onglet "Application"
- Dans le volet gauche, recherchez la catégorie "Storage" puis "Cookies"
- Tous les cookies seront affichés dans le tableau (nom, valeur, domaine d'envoi, chemin, expiration,...)



The screenshot shows the Chrome DevTools interface with the 'Application' tab selected. In the left sidebar, the 'Storage' section is expanded, and 'Cookies' is selected. The main panel displays a table of cookies for the domain 'http://localhost:63342'.

Name	Value	Domain
Phpstorm-94fb9e5e	880b1cd2-f609-4869-bfad-f266e8d5755e	localhost
country	BE	localhost
language	FR	localhost
test	FR	localhost

# Lire un cookie en PHP

- Si vous avez testé l'exemple précédent, la prochaine fois que le navigateur chargera une page sur votre site, il renverra deux cookies dont les noms sont language et country . Il nous reste donc à savoir relire cette information. Encore une fois, tout est déjà fait dans PHP et vous pouvez accéder à tous les cookies envoyés, grâce au tableau `$_COOKIE[]`. Il s'agit d'une des variables superglobales.
- Code:

```
// On vérifie si le cookie a été reçu
if(isset($_COOKIE['language'])) {
    // On lit la valeur du cookie et on l'affiche
    echo 'Langage: ' . $_COOKIE['language'];
}
```

# Modifier et effacer un cookie

- Pour modifier un cookie, il vous suffit de refaire appel à la fonction `setcookie()` avec le nom du cookie à modifier et sa nouvelle valeur. Il remplacera le précédent de même nom.
- Pour faire disparaître un cookie, vous devez définir une date de validité antérieure à la date actuelle en conservant la valeur utilisée lors de sa définition.
- Code:
  - `// suppression d'un cookie`
  - `setcookie('ISO', 'BE', time()-3600);`
- Rend le cookie inaccessible mais seulement lors du rechargement de la page qui contient ce code.

# Autres paramètres avec le cookie

- **Path:** Le chemin sur le serveur sur lequel le cookie sera disponible. Si la valeur est '/', le cookie sera disponible sur l'ensemble du domaine domain. Si la valeur est '/foo/', le cookie sera uniquement disponible dans le répertoire /foo/ ainsi que tous ses sous-répertoires comme /foo/bar/ dans le domaine domain. La valeur par défaut est le répertoire courant où le cookie a été défini.
- **Domain:** Le domaine pour lequel le cookie est disponible. Le fait de définir le domaine à 'www.example.com' rendra le cookie disponible pour le sous-domaine www mais aussi pour les sous-domaines supérieurs (ex: 'sub.www.example.com').
- **Secure:** Indique si le cookie doit uniquement être transmis à travers une connexion sécurisée HTTPS depuis le client. Lorsqu'il est positionné à TRUE, le cookie ne sera envoyé que si la connexion est sécurisée. Côté serveur, c'est au développeur d'envoyer ce genre de cookie uniquement sur les connexions sécurisées (par exemple, en utilisant la variable `$_SERVER["HTTPS"]`).
- **Httponly:** Lorsque ce paramètre vaut TRUE, le cookie ne sera accessible que par le protocole HTTP. Cela signifie que le cookie ne sera pas accessible via des langages de scripts, comme Javascript. Il a été accepté que cette configuration permet de limiter les attaques via XSS.

# Plusieurs valeurs dans un cookie

- Jusqu'à présent, nous n'avons stocké que des nombres ou des chaînes de caractères dans les cookies. Si vous souhaitez stocker autre chose, par exemple un tableau, il faudra transformer vos données en une chaîne de caractères avant d'envoyer le cookie. De même, une fois votre cookie récupéré à la prochaine page, il sera nécessaire de faire la transformation inverse. Cette transformation, qui s'appelle sérialisation, est gérée par les fonctions PHP `serialize()` et `unserialize()`. La première sert à transformer une variable en une chaîne de caractères, la seconde fait l'opération contraire.

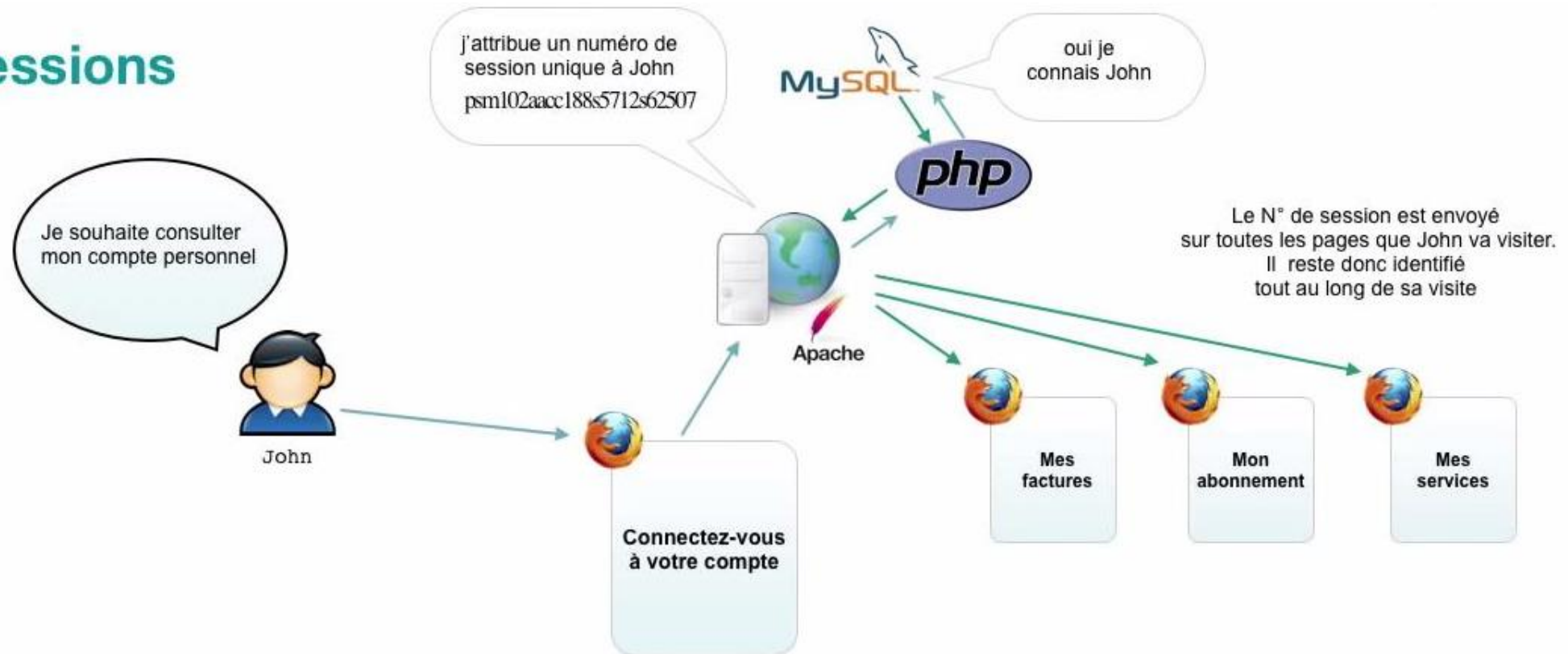
# Les sessions

Introduction



# Principe

## Les sessions



# Introduction

- Au chapitre précédent, nous avons vu comment stocker certaines informations sur le client grâce aux cookies. Ceux-ci ont toutefois deux défauts : leur taille est limitée et le visiteur peut les modifier ou les refuser à loisir. Ce n'est donc pas un bon emplacement pour des données sensibles comme des informations d'authentification ou de contrôle.
- PHP envoie au navigateur un identifiant de session et stocke des données sur le client dans un fichier correspondant à l'identifiant (côté serveur). Quand le navigateur refait une requête sur une de nos pages, l'identifiant est automatiquement renvoyé. PHP ouvre alors le fichier correspondant pour récupérer tout ce qui y avait été sauvegardé.
- La durée de vie des sessions est limitée à la session du navigateur mais peut également être limitée dans le temps par le développeur ou par l'internaute lui-même. **Les sessions sont valables sur tous les scripts** (pages).

- Les sessions sont adaptées à la sauvegarde de données confidentielles ou importantes. On peut citer quelques exemples courants de leur mise en application :
  - authentifier un visiteur ou un administrateur;
  - garder des informations sur un utilisateur tout au long de sa présence dans votre application;
  - gérer les messages d'erreurs liés aux formulaires ou autres fonctionnalités;
  - gérer le panier d'achat d'un internaute sur votre site commercial;
  - mettre en place des formulaires en plusieurs parties et donc retenir les informations fournies dans les pages précédentes;

# Création et utilisation d'une session

La session commence avec un appel à **session\_start()** ; son rôle est d'initialiser la gestion. les sessions utilisent parfois les cookies en interne. La conséquence est que l'appel à **session\_start()** doit respecter les mêmes règles que la fonction **setcookie()** : être placé en haut du script, avant toute sortie vers la page web. L'initialisation doit se faire dans tous les scripts utilisant les sessions, pas uniquement le premier.

Pour démarrer une session en PHP, on va utiliser la fonction `session_start()`. Cette fonction va se charger de vérifier si une session a déjà été démarrée en recherchant la présence d'un identifiant de session et, si ce n'est pas le cas, va démarrer une nouvelle session et générer un identifiant de session unique pour un utilisateur.

Pour lire, modifier, supprimer ou créer des informations dans la session, il vous suffit de lire, modifier, supprimer ou créer des entrées dans le tableau `$_SESSION[]`.

# Exemple

```
// Initialisation de la session
session_start();
// Écrire 'Dominique' dans la variable de session 'log'
$_SESSION['log'] = 'Dominique';
// Tester la présence de la variable 'log' dans la session
if(isset($_SESSION['log'])) {
    echo 'Votre login: ' . $_SESSION['log'];
} else {
    echo 'Non identifié';
}
```

# Suppression d'une session

- Habituellement, il n'est pas nécessaire de supprimer une session puisque PHP l'efface de lui-même au bout d'un certain temps. Si toutefois vous voulez détruire explicitement le fichier de données, vous pouvez utiliser la fonction **session\_destroy()**. Il vous faudra tout de même faire un appel à **session\_start()** quelques lignes plus haut dans le même script.
- Cette fonction ne fait qu'effacer le fichier de données, elle n'efface pas les variables présentes dans `$_SESSION[]`. Pour supprimer les données, utilisez la fonction **unset(\$\_SESSION['nomvariable'])**.

```
// Destruction session
session_start();
// traitements divers
unset($_SESSION['log']);
session_destroy();
```

# JSON et PHP

1. Introduction
2. Syntaxe et exemples
3. `Json_encode()`
4. `Json_decode()`
5. Gestion de fichiers

# Introduction

- JavaScript Object Notation (JSON) est un format d'échange de données dérivé de la notation des objets du langage JavaScript. Il permet de représenter et de stocker de l'information structurée et est indépendant de tout langage.
- Le format JSON est léger et moins verbeux que le XML. Il est parfait pour les informations de configuration et l'échange de données au travers de plusieurs serveurs (WebService).



# Syntaxe - JSON se base sur deux structures

## 1. Un objet

- C'est un ensemble (collection) de membres représentés par un nom et une valeur (un enregistrement, une structure, un dictionnaire, une table de hachage, une liste typée ou un tableau associatif).
- Un objet commence et se termine par une accolade, chaque nom est suivi de deux points et les couples (nom/valeur) sont séparés par une virgule.
  - Objet: { membre, membre, ... }
  - Membre: "nom" : "valeur "
- Exemples: { "name" : "Doe" } ou { "host" : "localhost", "dbname" : "web" }

# Syntaxe - JSON se base sur deux structures

## 2. Un tableau

- C'est un ensemble de valeurs ordonnées (une liste ou une suite) commençant et se terminant par des crochets et séparées par une virgule. Les valeurs peuvent être:
  - Une chaîne de caractères entre guillemets
  - Un nombre
  - Un objet
  - Un tableau
- Syntaxe: [ valeur, valeur, ... ]
- Exemples: ["PHP", "HTML", "JS", ...] ou
- [ { "name": "Doe", "firstName": "John"}, { "name": "Ford", "firstName": "Henry"} ]

## Exemple simple d'un fichier Json

```
[
  {
    "_id": "593932c4d778e2c6838ac2af",
    "gender": "female",
    "company": "INSOURCE",
    "email": "ollieparsons@insource.com",
    "phone": "+1 (956) 403-2264",
    "address": "820 Bouck Court, Noxen, Mississippi, 6891"
  },
  {
    "_id": "593932c4378625b14c601335",
    "gender": "female",
    "company": "PHARMEX",
    "email": "ollieparsons@pharmex.com",
    "phone": "+1 (869) 500-3906",
    "address": "857 Crosby Avenue, Cuylerville, Oklahoma, 1680"
  }
]
```

## Exemple avancé d'un fichier Json

```
[
  {
    "_id": "593933dcf68b3bb9fc83cdbd",
    "name": "Boyd Davenport",
    "gender": "male",
    "company": "MUSAPHICS",
    "longitude": -178.254489,
    "friends": [
      {
        "id": 0,
        "name": "Carolina Nieves"
      },
      {
        "id": 1,
        "name": "Norton Branch"
      },
      {
        "id": 2,
        "name": "Lindsay Ellison"
      }
    ]
  }
]
```

# Créer un fichier Json à partir de données en PHP

1. Les données se trouvent dans un tableau associatif (\$array)
2. La fonction **json\_encode()** permet d'encoder des données en PHP pour obtenir une représentation Json
3. La fonction **fopen()** ouvre un fichier. Le premier paramètre est le fichier à ouvrir et le second, le mode qui spécifie le type d'accès (w+).
4. La fonction **fwrite()** écrit dans le fichier spécifié le contenu.
5. La fonction **fclose()** ferme le fichier

```
$array = ['prenom'=>'Alexandre',  
'nom'=>'Chevalier',  
'profession'=>'Acteur'];  
$json = json_encode($array);  
//Création et ouverture du fichier  
$fileName = 'files/01-encode.json';  
$file = fopen($fileName , 'w+');  
// Ecriture dans le fichier  
fwrite($file, $json);  
// Fermeture du fichier  
fclose($file);  
var_dump($json);
```

## Lire un fichier Json en php

1. Pour des raisons de clarté, les données Json sont directement écrites dans la variable en PHP.
2. La fonction `json_decode()` décode une chaîne Json. Ici le décodage se fait dans une variable en PHP.
3. Affichage du nom du site (attribut de l'objet)

```
// Données en Json dans une variable PHP
$json = '{
    "idSite": "1",
    "nomSite": "Facebook",
    "urlSite": "http:\\\\www.facebook.com",
    "evalSite": "3",
    "catId": "1",
    "idCat": "1",
    "categorie": "Réseaux sociaux"
}';

$jsonDecode = json_decode($json);
echo $jsonDecode->nomSite;
```

# Création d'un fichier à partir d'une DB

```
$dbh = new
PDO('mysql:host=localhost;dbname=websites;charset=utf8',
'root', '');
$sql = "SELECT * FROM websites JOIN categories ON idCat =
catId";
$result = $dbh->query($sql);
while ($data = $result->fetch(PDO::FETCH_OBJ)) {
    $rows[] = $data;
}
$contentu_json = json_encode($rows, JSON_PRETTY_PRINT);
$nom_du_fichier = 'files/02-encode-db.json';
$fichier = fopen($nom_du_fichier, 'w+');
fwrite($fichier, $contentu_json);
fclose($fichier);
```

# Lecture du fichier Json en PHP

```
$json = file_get_contents("files/02-encode-db.json");  
$parsed_json = json_decode($json);  
  
echo '<h4>Tous les sites web</h4>';  
  
$i = 0;  
  
foreach ($parsed_json as $value) {  
    echo '<p>'. $parsed_json[$i]->{'nomSite'}.' - '. $parsed_json[$i]-  
>{'categorie'}.'</p>';  
    $i++;  
}
```



# Rappel des fonctions utilisées (json)

`json_encode ( $value [, $options] )`

\$value = la valeur à encoder

\$options = masque composé des constantes (php.net)

`json_decode ($json [, $assoc [, $depth [, int $options = 0 ]]] )`

\$json = la chaîne json à décoder

\$assoc = lorsque ce paramètre vaut TRUE, l'objet retourné sera converti en un tableau associatif

\$depth = définit la profondeur de récursion

\$options = un masque d'options JSON decode