

# 1. Sites dynamiques et langage PHP

Une page Web *dynamique* est une page Web dont le contenu HTML est généré automatiquement à l'aide d'un programme. Ceci est nécessaire lorsque le contenu de la page n'est pas connu à l'avance. À titre d'exemple, les pages Web obtenues via un moteur de recherche sont générées automatiquement en fonction des mots-clés saisis dans la barre de recherche.

Afin de créer des sites dynamiques, nous utiliserons dans ce cours le langage PHP. Ce langage est un langage de scripts interprété au niveau du **serveur**, permettant de **générer du code HTML**<sup>1</sup>. Cela modifie alors le traitement d'une requête d'une page Web. Celui-ci se déroule de la manière suivante :

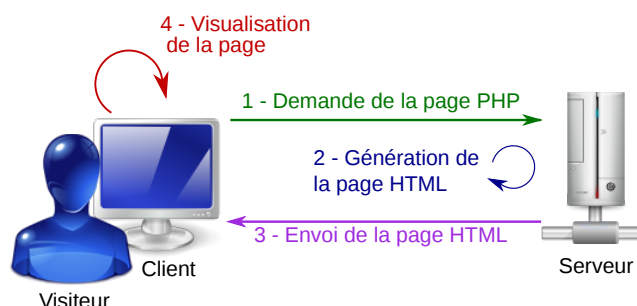


FIGURE 1.1 – Traitement d'une requête en PHP

1. Le client envoie une requête pour demander au serveur d'interpréter un script PHP. Ceci est fait en saisissant une adresse dont le fichier finit par l'extension **.php**. C'est le cas lorsque je saisis par exemple l'adresse **http://www.iutv.univ-paris13.fr/iutv/index.php**.
2. Le serveur interprète le fichier PHP, ce qui a pour résultat de **générer du code HTML**.
3. Le serveur répond en renvoyant le code HTML généré.
4. Le navigateur interprète le code HTML et affiche la page Web.

Souvent, le langage PHP a besoin d'informations pour générer la page HTML. C'est le cas par exemple lorsque la page générée doit afficher la liste des différents messages postés sur un forum. Ces différentes informations (dans cet exemple, les différents messages postés) sont stockées dans une base de données. Lors de l'exécution du script PHP, l'interpréteur PHP va alors automatiquement chercher dans la base de données les informations nécessaires pour générer le code HTML. C'est d'ailleurs le couplage du langage PHP et d'une base de données qui donne toute la force aux sites dynamiques. Ceci sera étudié dans le deuxième chapitre du cours.

## 1.1 Le langage PHP

Le langage PHP est le langage que nous allons utiliser pour générer automatiquement du code HTML. Il existe d'autres langages ayant les mêmes fonctions mais PHP est le plus répandu. Pour que le serveur interprète le langage PHP, il faut bien évidemment qu'un interpréteur PHP soit installé sur le serveur. De plus, il faut impérativement que le fichier contenant du code PHP ait l'extension **.php**.

Le PHP est un langage serveur, c'est-à-dire que le navigateur est incapable d'interpréter le langage PHP ! Il faut donc absolument passer par un serveur afin que celui-ci exécute le code PHP qui générera du code HTML et l'enverra au navigateur.

A l'intérieur d'un fichier `.php`, le code PHP doit être compris entre les balises `<?php` et `?>`. En dehors de ces balises, le seul code autorisé est le langage HTML.



*Le langage PHP ressemble par certains côtés au langage JAVA. Par exemple, toute instruction se termine par le caractère `"`;". De plus, les commentaires sont insérés derrière le symbole `//` (commentaire uniligne) ou entre les symboles `/*` et `*/` (commentaire multiligne).*

### 1.1.1 Inclusion de code

Il est possible d'inclure dans un fichier du code venant d'un autre fichier. Ce code peut être du PHP ou du HTML. Ceci permet de découper son code PHP et HTML en plusieurs fichiers de manière logique.

L'inclusion d'un fichier en PHP se fait grâce à la commande `require()` prenant en paramètre le nom du fichier et son chemin (relatif ou absolu). Par exemple, pour inclure le fichier `menu.php`, contenant le code HTML relatif au menu, dans un autre fichier PHP se trouvant dans le même répertoire sur le serveur, il suffit d'utiliser `<?php require('menu.php'); ?>`.

Si l'on souhaite que le fichier ne soit inclus que s'il n'a pas déjà été inclus, on peut utiliser la fonction `require_once()` plutôt que `require()`.

### 1.1.2 L'instruction echo

L'instruction `echo` est l'instruction la plus utilisée. Elle permet de générer du texte lors de l'interprétation du fichier PHP par le serveur. Le texte qui sera généré est entouré par des guillemets ou des apostrophes. Ainsi, si le fichier PHP contient le code suivant

```
1 <?php echo '<!doctype html>'
2 <html>
3   <head>
4     <title> TITRE OBLIGATOIRE </title>
5     <meta charset="utf-8"/>
6   </head>
7   <body>
8     <p> Hello world </p>
9   </body>
10 </html>';
11 ?>
```

son interprétation générera le code HTML suivant

```
1 <!doctype html>
2 <html>
3   <head>
4     <title> TITRE OBLIGATOIRE </title>
5     <meta charset="utf-8"/>
6   </head>
7   <body>
8     <p> Hello world </p>
9   </body>
10 </html>
```

Le code HTML généré par l'interprétation d'un fichier PHP doit être valide! Autrement dit, il doit satisfaire toutes les règles d'écriture vues l'année dernière. Afin de ne pas surcharger inutilement un fichier PHP d'instructions `echo`, il est possible de mélanger du code HTML et du code PHP. Dans ce cas, tout ce qui n'est pas du code PHP peut être vu comme faisant partie d'une instruction `echo`. Ainsi, le même fichier HTML peut être généré grâce au code PHP

```

1  <!doctype html>
2  <html>
3    <head>
4      <title> <?php echo 'TITRE OBLIGATOIRE'; ?> </title>
5      <meta charset="utf-8"/>
6    </head>
7    <body>
8      <p> <?php echo 'Hello world'; ?> </p>
9    </body>
10   </html>

```

On remarque que le fichier PHP précédent contient deux portions de code PHP, chacune contenant une unique instruction `echo`.

### 1.1.3 Variables

Le nom d'une variable doit commencer par une lettre (majuscule ou minuscule) ou le caractère `"_"`. Il peut ensuite comporter des lettres, des chiffres et le caractère `"_"`. Pour spécifier à l'interpréteur que l'on manipule une variable, on doit toujours mettre le caractère `"$"` avant le nom de la variable, sans espace entre les deux.

Il n'est pas nécessaire de déclarer une variable avant de l'utiliser. De même, les variables ne sont pas *typées*, c'est-à-dire que le type de la variable (nombre entier, nombre réel, chaîne de caractères, booléen, etc) est défini lors de son affectation et peut changer à tout moment. Le code PHP

```

1  <?php
2  $var = 148;
3  $var = "Hello world";
4  $var = 3.14;
5  ?>

```

définit une variable `$var`. Elle est d'abord de type entier et a pour valeur 148. Elle est ensuite de type chaîne de caractères et a pour valeur `"Hello world"`. Elle devient finalement de type flottant avec pour valeur 3.14.

L'interpréteur effectue des **conversions automatiques de type** sur les variables selon les besoins. Il est cependant possible de les faire manuellement. Pour cela, il suffit d'indiquer le type que l'on souhaite (`bool`, `int`, `float`, `string`) entre parenthèses avant la valeur. Par exemple, pour convertir une variable entière `$a` en un nombre réel, on utilise l'instruction `$a = (float) $a;`.

**R** Contrairement à beaucoup d'autres langages, il n'y a pas de division entière en PHP. Ainsi,  $3/4$  est égal à 0.75. Pour obtenir une valeur entière, il faut forcer la conversion en entier selon `(int)(3/4)`, qui prendra la partie entière inférieure de 0.75, soit 0.

**R** Une variable booléenne peut seulement prendre les valeurs `true` ou `false`.

**!** La fonction `isset($var)` renvoie `true` si la variable `$var` existe, et `false` sinon. La fonction `unset($var)` permet de supprimer la variable `$var`.

L'instruction `echo` permet également d'écrire, dans le fichier HTML généré, la valeur des variables. Le code PHP

```

1  $who = "Yoda";
2  $age = 400;
3  echo '<p>Quand ';
4  echo $age;
5  echo ' ans comme ';
6  echo $who;
7  echo ' tu auras, vieux tu seras !</p>';

```

générera alors

```
1 <p>Quand 400 ans comme Yoda tu auras, vieux tu seras !</p>
```

**R** Dans l'exemple précédent et par la suite, afin de ne pas surcharger les exemples, seule une partie du code PHP est présentée. Mais il est clair qu'il faut rajouter toutes les instructions nécessaires afin que le fichier HTML soit valide. Par exemple, dans l'exemple précédent, il faut ajouter toutes les balises nécessaires telles que les balises `html`, `head`, `body`. Afin de toujours écrire du code PHP correct, il faut impérativement vérifier que le fichier HTML généré est valide à l'aide d'un validateur.

**R** La fonction `var_dump()` est une fonction PHP très utile lors du débogage d'un programme. Elle prend en paramètre une variable et affiche alors sa valeur et son type. Par exemple, le code `$a=3.23; var_dump($a);` affiche `float(3.23)`.

### 1.1.4 Concaténation, apostrophes, guillemets et echo

Il est possible de concaténer des chaînes de caractères pour en former une seule. L'opérateur de concaténation du PHP est le caractère `.` qui permet de concaténer deux chaînes de caractères. La concaténation permet notamment de diminuer le nombre d'instructions `echo`. Par exemple, les quatre instructions `echo` du code précédent peuvent être remplacées par l'unique instruction

```
1 echo '<p>Quand ' . $age . ' ans comme ' . $who .
2 ' tu auras, vieux tu seras !</p>';
```

Si la chaîne de caractères que je souhaite afficher contient un ou plusieurs caractères apostrophe, il faut alors spécifier que ces derniers ne correspondent pas à la fin de la chaîne de caractères que je souhaite afficher mais à des caractères normaux. Autrement dit, l'interpréteur doit voir ces caractères comme des caractères quelconques. On utilise alors le caractère d'échappement `'\'`. Par exemple `'j\'ai 20 ans.'`.

Il est aussi possible de délimiter une chaîne de caractères avec les guillemets plutôt que les apostrophes. L'utilisation est similaire, à la différence qu'il est possible d'insérer des variables dans une chaîne de caractères délimitée par des guillemets. L'interpréteur remplacera alors les variables par leurs valeurs. Ainsi, lors de l'exécution du code PHP

```
1 $season = 'Winter';
2 echo "<p>$season is coming!</p>";
3 echo '<p>$season is coming!</p>';
```

le code HTML généré sera

```
1 <p> Winter is coming!</p><p>$season is coming!</p>
```

et le navigateur affichera alors

```
1 Winter is coming!
2
3 $season is coming!
```

Si l'affichage des valeurs des variables semble plus simple grâce à l'utilisation des guillemets, cette technique comporte quelques bémols. Ceci ne fonctionne pas si la variable est une case d'un tableau (il faut dans ce cas ajouter des accolades) et il n'est pas possible d'afficher de cette manière le résultat d'un calcul.

**R** Par défaut, le code HTML est généré sur une seule ligne. Pour effectuer des retours à la ligne dans le code HTML et le rendre ainsi plus lisible, il faut utiliser le caractère spécial `"\n"` dans une chaîne de caractères délimitée par des guillemets (pour qu'il soit interprété). Attention, il ne faut pas confondre ceci avec la balise `<br />` qui indique au navigateur d'aller à la ligne lors de l'affichage de la page Web.

► Sur ce thème : **Exercice 1**

### 1.1.5 Conditions

Les instructions de contrôle (instructions **if**) s'utilisent de manière similaire en PHP et en C. La syntaxe est quasiment la même, excepté que le langage PHP possède également l'instruction **elseif**. Comme en C, si plusieurs instructions doivent être effectuées pour une valeur du test (vrai ou faux), ces instructions doivent être délimitées par des accolades. Voici un exemple d'utilisation de structure de contrôle.

```

1  if($a < 0)
2      echo '<p> nombre négatif </p>';
3  elseif($a < 10)
4      echo '<p> nombre positif ou nul mais inférieur strictement à 10 </p>';
5  else
6      echo '<p> nombre supérieur ou égal à 10 </p>';

```

Les opérations de comparaisons sont : <, <=, >, >=, ==, !=, ===, !==. Les quatre premiers sont classiques et n'ont donc pas besoin d'explication. Pour tester l'égalité entre deux éléments, il existe deux opérateurs : == et ===. Le premier opérateur renvoie **true** si les valeurs des deux éléments sont identiques. Pour tester les valeurs, une conversion automatique de type est effectuée si cela est nécessaire. Le deuxième opérateur renvoie **true** uniquement si les valeurs et les types des deux éléments sont identiques. Voici un exemple d'utilisation des opérateurs == et ===

```

1  echo '1.4'==1.4;    //Affiche 1 car le test est vrai
2  echo '1.4'===1.4;   //N'affiche rien car le test est faux
3  echo 2===2.0;       //N'affiche rien car le test est faux

```

Les opérateurs != et !== sont respectivement les négations logiques des opérateurs == et ===.

Une condition peut être définie par plusieurs conditions reliées par les opérateurs logiques **and** et **or**. Le non logique est défini par l'opérateur **!**.

### 1.1.6 Boucles

Le langage PHP accepte les instructions **for**, **while** et **do ... while**. Ces boucles s'utilisent de la même manière qu'en langage C. Voici rapidement un exemple de boucles :

```

1  echo '<p>';
2  for($i=0;$i<10;$i++)
3      echo 'itération numéro ' . $i . '<br />';
4
5  $i = 0;
6  while($i < 10) {
7      echo 'itération numéro ' . $i . '<br />';
8      $i++;
9  }
10
11 $i = 0;
12 do {
13     echo 'itération numéro ' . $i . '<br />';
14     $i++;
15 } while ($i < 10);
16 echo '</p>';

```

► Sur ce thème : **Exercice 2**

### 1.1.7 Tableaux

Les tableaux regroupent un ensemble de valeurs qui peuvent être de **types différents**. Chaque valeur est associée à une *clé* qui permet de retrouver l'élément dans le tableau. Cette clé peut être

un entier (par exemple l'indice où se trouve l'élément dans le tableau) ou une chaîne de caractères. Pour accéder à une valeur du tableau, on utilise alors la notation `$variableTableau[clé]` où `clé` correspond à la clé associée à la valeur à laquelle on souhaite accéder. On parle de *tableau associatif*.

Un tableau se déclare à l'aide de crochets. Les différents couples clé/valeur du tableau sont mis entre les crochets et séparés par des virgules. Chaque couple clé/valeur est noté `clé => valeur`.

```

1 //Création d'un tableau contenant le nombre 21 et la chaîne 'toto'.
2 $tableau = [0 => 21, 1 => 'toto'];
3
4 //Création d'un tableau contenant 4 valeurs : 12, 'rue de la liberté',
5 //'Paris', 75001
6 $adresse = ['numero' => 12, 'rue' => 'rue de la liberté',
7            'ville' => 'Paris', 'cp' => 75001];
8
9 echo '<p>' . $tableau[0] . '<br />' ; //Affiche 21
10 echo $tableau[1] . '<br />' ; //Affiche toto
11 echo $adresse['numero'] . '<br />' ; //Affiche 12
12 echo $adresse['rue'] . '<br />' ; //Affiche rue de la liberté
13 echo $adresse['cp'] . '<br />' ; //Affiche 75001
14 echo $adresse['ville'] . '</p>' ; //Affiche Paris

```

❗ Il n'est pas possible d'accéder aux valeurs d'un tableau à l'aide des indices, à moins que les clés correspondent aux indices comme dans le tableau `$tableau`. Ainsi, dans l'exemple précédent, `$adresse[1]` ne correspond pas à *'rue de la liberté'*, cela ne correspond à rien car il n'y a aucune clé associée à la valeur 1 dans le tableau `$adresse`.

Ⓡ Il est possible de mélanger dans un tableau des clés de type entier et de type chaînes de caractères.

Pour modifier une valeur associée à une clé ou pour ajouter un nouveau couple clé/valeur, il suffit d'affecter à `$variableTableau[clé]` la nouvelle valeur.

```

1 //En reprenant l'exemple précédent
2 $adresse['ville'] = 'Villetaneuse';
3 $adresse['cp'] = 93430;
4 $adresse['pays'] = 'France';
5
6 /* Le tableau adresse contient maintenant les valeurs 12,
7    'rue de la liberté', 93430, 'Villetaneuse', 'France' */

```

Il est possible de laisser l'interpréteur PHP associer lui-même la clé à une valeur du tableau. Pour cela, il suffit de ne pas spécifier la clé. Dans ce cas, l'interpréteur recherche, parmi les clés entières du tableau, l'entier le plus grand et associe comme clé ce nombre plus un, si cette valeur correspond un nombre positif ou nul, ou la valeur 0.

```

1 $t = [35, 20 => 42, 10 => 27];
2 $t[] = -6;
3 $t[] = 'hello world';
4 /* $t contient les valeurs 35, 42, 27, -6, 'hello world'.
5    Les clés associées sont respectivement 0, 20, 10, 21, 22 */

```

Ⓡ Lors du développement ou du débogage, il est possible d'utiliser la fonction `print_r()` plutôt que `var_dump()` pour afficher un tableau de manière plus lisible.

► Sur ce thème : **Exercice 3**

### Parcours de tableaux

Pour parcourir un tableau, c'est-à-dire, pour accéder aux couples clés/valeurs d'un tableau, on utilise la boucle `foreach` selon :


```
1 foreach($tableau as $cle => $valeur ) {  
2     //Bloc d'instructions  
3 }
```

La variable `$tableau` correspond au tableau que l'on veut parcourir. Le bloc d'instructions associé au `foreach` est exécuté pour chaque case du tableau. À chaque itération, les variables `$cle` et `$valeur` contiennent respectivement la clé et la valeur de la case courante. À titre d'exemple, l'exécution du code :

```
1 $tab = ['a' => 18, 'b' => 'hello', 'c'];  
2 foreach($tab as $c => $v)  
3     echo '<p> cle : ' . $c . ', valeur : ' . $v . ' </p>' . "\n" ;
```

produit le résultat suivant :

```
1 <p> cle : a, valeur : 18 </p>  
2 <p> cle : b, valeur : hello </p>  
3 <p> cle : 0, valeur : c </p>
```

 Dans l'exemple PHP précédent, `'c'` est une valeur du tableau et non une clé. Comme aucune clé n'a été spécifiée pour cette valeur, l'interpréteur PHP lui a associé la clé 0.

Si l'on souhaite n'avoir que les valeurs du tableau, il est possible de supprimer la variable contenant la clé dans le `foreach`. L'instruction devient alors :

```
1 foreach($tableau as $valeur)  
2     echo '<p> valeur : ' . $valeur . ' </p>' ;
```

### ► Sur ce thème : Exercice 4

#### Fonctions relatives aux tableaux

Il existe plusieurs fonctions relatives aux tableaux qui sont déjà définies en PHP. Voici les plus importantes d'entre elles.

##### Fonction `count($tab)` :

Cette fonction retourne le nombre de valeurs du tableau `$tab`.

##### Fonction `in_array($val, $tab, $strict=false)` :

Cette fonction renvoie `true` si la valeur `$val` est une valeur du tableau `$tab`, et `false` sinon. Le troisième paramètre (optionnel) indique si le test de `$val` avec les différentes valeurs du tableau prend en compte le type. S'il est à `true`, alors l'opérateur de comparaison est `===` alors que s'il est à `false`, l'opérateur utilisé est `==`.

##### Fonction `array_search($val, $tab, $strict=false)` :

Cette fonction renvoie la clé associée à la valeur `$val` dans le tableau `$tab` si `$val` est une valeur du tableau, et `false` sinon. Le troisième paramètre `$strict` joue le même rôle que pour la fonction `in_array()`.

##### Tester si une clé existe :

Pour tester si `$cle` est une clé du tableau `$tab`, il suffit de tester si la valeur associée à cette clé existe en utilisant `isset($tab[$cle])`.

**Supprimer une valeur :**

La suppression de la valeur `$val` associée à la clé `$cle` dans le tableau `$tab` se fait de la manière suivante : `unset($tab[$cle])`.

**Fonction `array_sum($tab)` :**

Cette fonction renvoie la somme des valeurs d'un tableau.

**Fonction `array_keys($tab)` :**

Cette fonction renvoie un nouveau tableau dont les valeurs sont les clés du tableau `$tab` (les clés de ce nouveau tableau correspondent aux indices).

**Fonctions de tri :**

La fonction `sort($tab)` permet de trier le tableau `$tab` de la valeur la plus petite à la valeur la plus grande. Les clés du tableau correspondent alors aux indices. Si l'on souhaite garder les clés du tableau `$tab`, il faut utiliser la fonction `asort($tab)`. Pour trier le tableau selon les clés, il faut alors utiliser la fonction `ksort($tab)`. Enfin, si l'on souhaite trier selon l'ordre inverse, il faut respectivement utiliser les fonctions `rsort($tab)`, `arsort($tab)` et `krsort($tab)`.

**Fonction `implode($colle, $tab)` :**

Cette fonction retourne une chaîne de caractères constituée des valeurs du tableau collées entre elles avec la chaîne `$colle`. Ainsi, `implode(' : ', ['a', 2, 'c'])` retourne la chaîne de caractères `'a : 2 : c'`.

**Fonction `explode($delimiteur, $ch)` :**

Cette fonction prend en paramètre deux chaînes de caractères. Elle découpe la chaîne `$ch` en morceaux qui sont délimités par la chaîne `$delimiteur`. Elle retourne un tableau dont les valeurs correspondent aux différents morceaux de la chaîne découpée. Ainsi, `explode(' ', 'Il fait beau')` retourne le tableau `['Il', 'fait', 'beau']`.

► Sur ce thème : **Exercices 5 et 6**

**1.1.8 Fonctions**

La définition de fonctions se fait de la manière suivante :

```
1 function nom_fonction($param1, $param2, ...) {
2     //Bloc d'instructions
3     return val; //Pour retourner une valeur
4 }
```

Pour appeler une fonction, on écrit son nom suivi des valeurs des paramètres entre parenthèses. Voici un exemple de fonction :

```
1 <?php
2 //Définition de la fonction
3 function moyenne($tab) {
4     return array_sum($tab) / count($tab);
5 }
6
7 //Appels de la fonction
8 $res = moyenne([10,20,14,16]);
9 echo '<p> Moyenne = ' . $res . '</p>';
10
11 $res = moyenne([1,1,1,4]);
12 echo '<p> Moyenne = ' . $res . '</p>';
13 ?>
```

L'exécution du code PHP précédent fournit alors le code HTML suivant :



```
1 <p> Moyenne = 15</p><p> Moyenne = 1.75</p>
```



*Pour retourner plusieurs valeurs, il faut retourner un tableau.*

### Passage des paramètres d'une fonction

Par défaut, les paramètres d'une fonction sont passés par copie. Ceci signifie que la variable passée en paramètre aura la même valeur avant et après l'appel de la fonction (mais sa valeur peut changer durant l'appel). Ceci est également vrai pour les tableaux passés en paramètre.

```
1 function test($nb) {
2     $nb += 5;
3     echo '<p>'. $nb . '</p>';
4 }
5
6 $x = 1;
7 echo '<p>'. $x . '</p>'; //Affiche 1
8 test($x);             //Affiche 6
9 echo '<p>'. $x . '</p>'; //Affiche 1
```

Il est possible de passer les paramètres par référence. Pour cela, il faut ajouter le symbole & devant le paramètre. Ceci permet que les modifications de ce paramètre à l'intérieur de la fonction se répercutent en dehors.

```
1 function test(&$nb) {
2     $nb += 5;
3     echo '<p>'. $nb . '</p>';
4 }
5
6 $x = 1;
7 echo '<p>'. $x . '</p>'; //Affiche 1
8 test($x);             //Affiche 6
9 echo '<p>'. $x . '</p>'; //Affiche 6
```

### Variables locales

Il est possible de créer des variables locales dans la fonction. Ces variables seront détruites à la fin de la fonction. Par ailleurs, seules les variables locales et les paramètres sont accessibles dans les instructions de la fonction. Il n'est donc pas possible d'utiliser une variable définie en dehors de la fonction et qui n'est pas passée en paramètre. Une exception cependant : les variables `$_GET`, `$_POST`, `$_SESSION` et `$_COOKIE`, que l'on verra par la suite, sont accessibles dans la fonction même si elles ne sont pas passées en paramètre.

► Sur ce thème : **Exercice 7**

#### 1.1.9 Objets

Le langage PHP possède une couche objet dont la syntaxe est très proche de celle du langage JAVA. Le langage PHP implémente donc les classes, l'héritage, les interfaces, etc. Nous donnons ici un rapide aperçu sur l'utilisation de cette couche objet. Les personnes souhaitant acquérir une connaissance plus approfondie sur la programmation orientée objet en PHP pourront trouver des cours et tutoriels sur le Web<sup>2</sup>.

La définition d'une classe se fait à l'aide du mot-clé `class` suivi du nom de la classe. Les accolades qui suivent contiennent alors les attributs (précédés du symbole `$`) et méthodes de cette classe. Chaque attribut ou méthode est précédé du mot-clé `public`, `protected` ou `private` suivant qu'il soit public, protégé ou privé.

- R** Un attribut (ou méthode) est public s'il est accessible partout dans le programme. Il est privé (respectivement protégé) s'il est accessible uniquement à l'intérieur de la classe (respectivement à l'intérieur de la classe et des ses descendants).

Pour accéder aux attributs d'un objet, on utilise la syntaxe `$objet->attribut`. À l'intérieur d'une méthode, `$this` fait référence à l'objet sur lequel est appelée la méthode.

Les attributs ou méthodes statiques (appartenant à une classe et non à un objet) sont précédés du mot clé `static`. Pour accéder à ces attributs ou méthodes, il faut faire précéder le nom de la méthode ou de l'attribut par le nom de la classe suivi de `::`. Les constantes dans une classe sont définies à l'aide du mot-clé `const` et sont accessibles de la même manière que les attributs statiques.

Voici un exemple de définition et d'utilisation d'une classe en PHP.

```

1  class Joueur {
2      public $login;
3      private $score;
4      const NOM_JEU = "Fortnite";
5
6      public function __construct($ch,$ent) {
7          $this->score = $ent;
8          $this->login = $ch;
9      }
10
11     public function score() {
12         return $this->score;
13     }
14
15     public function gagne($score) {
16         $this->score += $score;
17     }
18 }
19 echo '<p>Jeu : ' . Joueur::NOM_JEU . '</p>';
20 $j1 = new Joueur('Bugha',21);
21 $j2 = new Joueur('psalm',33);
22 $j1->gagne(38);
23 echo '<p> Scores : ' . $j1->login . ' a ' . $j1->score() . ' points <br/>';
24 echo $j2->login . ' a ' . $j2->score() . ' points. </p>';

```

L'exécution du code précédent affichera :

```

1  Jeu : Fortnite
2
3  Scores : Bugha a 59 points
4  psalm a 33 points.

```

- R** Le constructeur d'une classe est la méthode `__construct()`. La surcharge, qui consiste à définir plusieurs versions d'une même fonction ou méthode avec un nombre différent de paramètres, n'est pas possible en PHP<sup>3</sup>. Par conséquent, il n'y a qu'un constructeur par classe.

- !** Une variable de type objet ne contient pas l'objet mais uniquement un identifiant de l'objet (ce qui permet de retrouver l'objet). Ainsi, si `$a` est une variable de type objet, après `$b = $a;`, `$a` et `$b` correspondent au même objet. Ils ont chacun une copie de l'identifiant qui pointe sur le même objet. De même, si je passe un objet en paramètre d'une fonction par copie, les modifications sur cet objet sont globales.

- R** Une méthode (ou fonction) peut être appelée depuis une variable contenant son nom. Par exemple `$methode = "score"; $j1->$methode();` appellera la méthode `score` de `$j1`.

► Sur ce thème : **Exercice 8**

### 1.1.10 Expressions régulières en PHP

Il est très important de vérifier que les données saisies par l'utilisateur sont correctes et cohérentes. En effet, si l'on demande l'âge de quelqu'un, il faut que la valeur saisie soit un nombre. De même, si vous demandez une adresse e-mail, il faut vous assurer que la saisie contient le symbole "@". Ceci peut être fait à l'aide des expressions régulières.

Il existe deux types d'expressions régulières : POSIX et PCRE. Celles-ci sont similaires, seules quelques règles d'écriture et fonctionnalités diffèrent. Nous décrivons ici les expressions régulières de type PCRE. Les explications données ici sont loin d'être exhaustives. Il est possible d'obtenir plus d'informations sur l'utilisation des expressions régulières sur le Web.

Une *Expression Régulière (ER)* représente toutes les chaînes vérifiant certaines propriétés. Des exemples d'ER sont :

- toutes les chaînes non vides composées uniquement de chiffres,
- toutes les chaînes de caractères contenant le symbole "@" et se terminant par les caractères ".fr" ou ".com",
- toutes les chaînes de caractères commençant par "http://" et se terminant par ".org".

On dit qu'une chaîne de caractères *satisfait* l'expression régulière si elle appartient à l'ensemble des chaînes représentées par l'ER. Ainsi, la chaîne "lacroix@iutv.univ-paris13.fr" satisfait l'ER correspondant au deuxième exemple, contrairement à la chaîne "toto@xyz.ffr".

Les expressions régulières sont écrites sous forme de chaînes de caractères, en respectant plusieurs règles. Tout d'abord, une ER est délimitée au début et à la fin par le symbole "#"4. Voici plusieurs exemples d'ER :

- "#php#"
- "#p.\*p#"
- "#^[0-9]+\$#"
- "#@.\*\.(fr|com)\$#"
- "#^http://.\*\.org\$#"

Le premier exemple d'expression régulière correspond à toutes les chaînes contenant la suite de caractères "php". Le deuxième correspond à toutes les chaînes contenant au moins deux lettres "p". Les chaînes "php" et "aerlkjperp" satisfont donc la deuxième ER. Les trois derniers exemples correspondent à la traduction des trois exemples donnés en toutes lettres ci-dessus.

L'écriture des ER utilise des *caractères spéciaux* qui sont des caractères ayant une signification particulière. Ceux-ci sont présentés dans le tableau 1.2.

Caractère spécial	Signification
.	représente un caractère quelconque
?	ce qui précède doit apparaître 0 ou 1 fois
*	ce qui précède peut apparaître 0, 1 ou plusieurs fois
+	ce qui précède doit apparaître au moins une fois
^	ce qui suit correspond au début de la chaîne
\$	ce qui précède correspond à la fin de la chaîne
[ ]	définit un ensemble de caractères possibles
( )	définit une séquence de caractères
\	le caractère spécial suivant doit être considéré comme un caractère normal
	opérateur OU logique
{n}	ce qui précède doit apparaître exactement n fois
{n,m}	ce qui précède doit apparaître entre n et m fois

FIGURE 1.2 – Tableau des caractères spéciaux et de leurs significations

Les caractères spéciaux et le caractère `"#"` apparaissant dans l'ER comme un caractère quelconque doivent alors être échappés, c'est-à-dire précédés du symbole `"\"`. Ainsi, l'expression régulière `"#faire\?$#"` représente l'ensemble des chaînes se terminant par la suite de caractères `"faire?"`, alors que l'ER `"#faire?##"` correspond à l'ensemble des chaînes se terminant par `"fair"` ou `"faire"`. Il y a une exception lorsque le caractère spécial est entre crochets ; dans ce cas, il ne faut pas utiliser le symbole `"\"`. Ainsi, l'ER `"#faire[.?$#"` correspond à l'ensemble des chaînes terminant par `"faire."` ou `"faire?"`. Cependant, les caractères `"#"` et `"]"` doivent quand même être échappés à l'intérieur des crochets.

Le tiret peut être utilisé à l'intérieur des crochets afin de spécifier un intervalle. Ainsi, les ER `"#^[abcdef]#"` et `"#^[a-f]#"` sont équivalentes et représentent l'ensemble des chaînes commençant par l'une des six premières lettres de l'alphabet. Si l'on souhaite qu'un des caractères possibles entre les crochets soit un tiret, il faut que celui-ci soit placé en début ou fin. L'ER `"#^[af-]#"` correspond à l'ensemble des chaînes commençant par la lettre `"a"`, la lettre `"f"` ou le tiret.



*On peut utiliser dans une expression régulière les symboles `\d` pour désigner un chiffre entre 0 et 9 et `\w` pour désigner un chiffre, une lettre ou le symbole underscore. Ainsi `"#\d+$#"` représente l'ensemble des chaînes de caractères composées uniquement de chiffres (au moins un).*

Les expressions régulières peuvent être utilisées en PHP pour vérifier si une chaîne de caractères satisfait une expression régulière, ou pour remplacer dans une chaîne de caractères toute partie satisfaisant une ER par une autre chaîne. Ceci se fait à l'aide des deux fonctions PHP suivantes.

**Fonction `preg_match(er, ch)` :** Cette fonction retourne `true` si la chaîne `ch` satisfait l'expression régulière `er`, et `false` sinon. L'instruction `preg_match('#p.p#', 'vive le php');` retourne donc `true`<sup>5</sup>.

**Fonction `preg_replace(er, remplacement, ch)` :** Cette fonction remplace tous les motifs de l'expression régulière `er` trouvés dans la chaîne de caractères `ch` par la chaîne de caractères `remplacement`. La fonction retourne la nouvelle chaîne de caractères ainsi obtenue. L'instruction `echo preg_replace('#p.p#', 'PHP', 'les papillons papillonnent');` affichera alors les PHPillons PHPillonnet.

► Sur ce thème : **Exercice 9**

## 1.2 Transmission d'informations et formulaires

Il est possible de transmettre des informations à un script PHP pour modifier son comportement. Ces informations peuvent être par exemple le nom de l'utilisateur ou les réponses d'un formulaire. Ces informations vont pouvoir modifier le déroulement du script. Elles vont par exemple permettre de créer des pages personnalisées, de sécuriser certaines pages par mot de passe, ou d'afficher les résultats d'un questionnaire.

### 1.2.1 Ajout d'arguments dans l'url

La méthode la plus simple pour transmettre des informations à un script PHP consiste à les mettre directement dans l'url. Le script récupérera ces informations dans un tableau spécifique nommé `$_GET`. Dans l'url, il faut alors spécifier un ou plusieurs couples de clés/valeurs selon `clé=valeur`, les couples étant séparés par le symbole `&`. Pour différencier l'adresse du fichier des couples clés/valeurs, il faut ajouter le symbole `?` avant le premier couple. Chaque couple clé/valeur est appelé *paramètre*, la clé est appelée *nom du paramètre* et la valeur *valeur du paramètre*. Attention, il est obligatoire de préciser une clé pour chaque valeur. Voici un exemple d'url avec transmission d'informations : `http://www.iutv.univ-paris13.fr/iutv/index.php?nom=Jean&age=32&etudiant=non`. Dans cet exemple, nous avons passé trois paramètres, à savoir `nom/Jean`, `age/32` et `etudiant/non`. Le script `index.php` peut alors utiliser ces informations grâce au tableau `$_GET`. Les valeurs de `$_GET` sont les valeurs des paramètres, la clé associée à chaque valeur étant

le nom du paramètre. Le script peut par exemple afficher les différents couples clés/valeurs dans une liste non ordonnée selon :

```
1 echo "<ul>\n";
2 foreach($_GET as $cle => $val)
3     echo "<li> " . $cle . " : " . $val . " </li>\n";
4 echo "</ul>\n";
```

Avec l'url précédente, le code HTML généré lors de l'exécution du code PHP précédent serait :

```
1 <ul>
2 <li> nom : Jean </li>
3 <li> age : 32 </li>
4 <li> etudiant : non </li>
5 </ul>
```



*Il faut **TOUJOURS VÉRIFIER** qu'un paramètre existe avant de l'utiliser. En effet, un script peut être appelé par n'importe qui et n'importe comment, notamment sans paramètre. Pour tester l'existence d'un paramètre de nom `nom`, on peut utiliser `isset($_GET['nom'])`.*



*Il faut **TOUJOURS VÉRIFIER** les valeurs des paramètres. Comme ce sont des chaînes de caractères, on peut tester qu'elles vérifient certaines conditions (correspondent à un nombre, une adresse mail, ne sont pas constituées uniquement d'escapes, etc) à l'aide des expressions régulières.*

► Sur ce thème : **Exercice 10**

## 1.2.2 Retour à l'HTML : les formulaires

Si le passage d'arguments dans l'url fonctionne correctement, ce procédé n'est pas confortable pour l'utilisateur. Dans le cas d'une inscription sur un site, par exemple, la saisie de toutes les informations dans l'url devient extrêmement fastidieuse à cause du nombre de paramètres. De plus, l'utilisateur ne connaît pas les noms des paramètres que vous souhaitez qu'il saisisse (par exemple : `nom` et pas `name`, etc). Un moyen beaucoup plus simple consiste alors à utiliser les *formulaires*. Les formulaires sont des balises HTML qui permettent à l'utilisateur de saisir uniquement les valeurs des paramètres, les noms étant définis directement dans le formulaire. La transmission dans l'url est faite automatiquement lors de la soumission du formulaire. L'utilisation des formulaires est très courante. Les formulaires sont notamment utilisés lors de la création d'un compte (par exemple un compte facebook), lors de l'identification sur un site (comme celui de l'iut), lors de la saisie d'une recherche sur un moteur de recherche, etc. Nous présentons ici différentes balises HTML permettant la création d'un formulaire.

**Balise `form` :** Cette balise permet de définir un formulaire. Toute balise définissant un champ de saisie de formulaire doit être comprise entre des balises `<form>` et `</form>`. La balise `form` doit contenir l'attribut `action` qui indique le nom de la page PHP qui est appelée (et à qui les informations saisies dans le formulaire sont envoyées) lorsque l'utilisateur soumet le formulaire.

**Balise `input` :** Cette balise auto-fermante permet de créer des champs de saisie de différents types. Elle doit contenir l'attribut `name` indiquant le nom du paramètre. De plus, elle possède l'attribut `type` qui permet de définir le type du champ. Voici les principales valeurs que peut prendre l'attribut `type` :

- `text` : indique que le champ est de type texte. La valeur du paramètre sera égale au texte saisi par l'utilisateur.
- `password` : joue le même rôle que `text` mais le texte saisi par l'utilisateur est caché. Ceci est utilisé lors de la saisie de mots de passe.
- `hidden` : indique que le champ ne sera pas affiché ("`hidden`" signifie "caché" en anglais). Ce type permet de transmettre des informations invisibles pour l'utilisateur.

- **radio** : indique que le champ est une case à cocher. Ce type de bouton est à utiliser si l'on souhaite que l'utilisateur coche exactement un seul bouton pour une question. C'est le cas pour la civilité : l'utilisateur doit cocher Homme ou Femme. Dans ce cas, on crée deux balises **input** de type **radio** ayant le même nom de paramètre (défini par l'attribut **name**). Il faut également spécifier pour chaque balise la valeur du paramètre si l'utilisateur coche cette case. Ceci se fait grâce à l'attribut **value**. Par exemple, on peut avoir :

```

1 <p>
2 <input type="radio" name="civilite" value="h" /> Homme <br />
3 <input type="radio" name="civilite" value="f" /> Femme
4 </p>

```

Suivant la case cochée, le paramètre sera soit **civilite=h**, soit **civilite=f**.

Afin que le bouton soit cliquable même si l'on clique sur le texte à côté, il faut encadrer la balise **input** et le texte associé par les balises **<label>** et **</label>**.

- **checkbox** : indique que le champ est une case à cocher. Contrairement au bouton radio, la case **checkbox** permet de cocher autant de cases que l'on souhaite. Dans le cas où l'utilisateur peut cocher plusieurs cases (par exemple : quels langages de programmation utilisez-vous pour le Web ?), il existe deux possibilités de programmer ceci. À chaque case à cocher est associé un nom de paramètre (**name**) différent. De plus, aucune valeur (**value**) n'est définie. Dans ce cas, si une case est cochée, par exemple celle dont le nom de paramètre est **"php"**, le tableau **\$\_GET** contiendra le couple **'php' => 'on'**. Autrement, il est possible de définir le même nom de paramètre (**name**) à chacune des cases à cocher. L'attribut **value** est alors utilisé pour indiquer quelle valeur sera renvoyée si la case est cochée. Comme l'utilisateur peut cocher plusieurs cases, il peut y avoir plusieurs valeurs pour un même nom de paramètre. Dans ce cas, il faut spécifier que la valeur associée au nom est un tableau. On indique ceci en HTML en ajoutant les crochets après le nom du paramètre dans la valeur de l'attribut **name**. Par exemple, on peut avoir :

```

1 <p>
2 <input type="checkbox" name="langage[]" value="HTML" /> HTML <br />
3 <input type="checkbox" name="langage[]" value="css" /> CSS <br />
4 <input type="checkbox" name="langage[]" value="php" /> PHP <br />
5 <input type="checkbox" name="langage[]" value="javascript" /> Javascript
6 </p>

```

Si l'utilisateur sélectionne les cases associées à HTML, CSS et PHP, la valeur associée à la clé **langage** dans le tableau **\$\_GET** sera **['HTML', 'css', 'php']**.

- **submit** : permet de soumettre le formulaire. On peut ajouter un attribut **value** dont la valeur correspondra au texte affiché sur le bouton (exemple : **value="Envoi du formulaire"**). On peut ajouter un attribut **name** pour ajouter un paramètre lors de la soumission du formulaire avec ce bouton **submit**. Sa valeur correspond à l'attribut **value**. Ceci est utile s'il y a plusieurs boutons **submit** dans un même formulaire.

**Balise select** : Cette balise permet de créer un menu déroulant pour que l'utilisateur saisisse une réponse parmi un ensemble prédéfini. La balise **select** utilise l'attribut **name** qui donne le nom du paramètre. Chaque valeur possible du paramètre est définie grâce à la balise **option**. Si l'attribut **value** est dans la balise ouvrante **option**, il indique alors la valeur du paramètre si la réponse associée est choisie. Dans le cas contraire, la valeur renvoyée est celle comprise entre **<option>** et **</option>**. Voici un exemple d'utilisation de la balise **select** :

```

1 <select name="pays">
2   <option value="en"> England </option>
3   <option value="fr"> France </option>
4   <option value="it"> Italia </option>
5 </select>

```

Si l'utilisateur choisit la réponse **Italia** lors de la soumission du formulaire, le paramètre associé sera `pays=it`.

**Balise `textarea` :** Cette balise permet la saisie d'un texte plus long que la balise `input` de type `text`. Les attributs obligatoires sont : `name` correspondant au nom du paramètre (la valeur du paramètre est le texte saisi par l'utilisateur), `cols` et `rows` qui donnent respectivement le nombre de colonnes et de lignes utilisées pour l'affichage de la zone de saisie du texte.

Voici maintenant un exemple de formulaire :

```

1 <form action="traitementFormulaire.php">
2 <p> Nom : <input name="nom" type="text" /> <br />
3 Prénom : <input name="prenom" type="text" /> <br />
4 Civilité : <br />
5 <label> <input name="civilite" type="radio" value="h"/> Homme </label> <br />
6 <label> <input name="civilite" type="radio" value="f"/> Femme </label> <br />
7 <input type="submit" /> </p>
8 </form>

```

Voici maintenant le code PHP du fichier `traitementFormulaire.php` permettant d'utiliser les informations reçues lors de la soumission du formulaire précédent pour "souhaiter la bienvenue" à l'utilisateur :

```

1 if ( isset($_GET['nom']) and isset($_GET['prenom'])
2     and isset($_GET['civilite']) and preg_match("#^\w+$#", $_GET['nom'])
3     and preg_match("#^\w+$#", $_GET['prenom'])
4     and ($_GET['civilite'] == 'h' || $_GET['civilite'] == 'f')) {
5     if($_GET['civilite'] == 'h')
6         echo '<p>Hello Mr ' ;
7     else
8         echo '<p>Hello Mrs ' ;
9     echo $_GET['prenom'] . ' ' . $_GET['nom'] . ' !</p>';
10 }
11 else

```

Si l'utilisateur saisit "John" pour le prénom, "Doe" pour le nom, et coche la case associée à Homme, la soumission du formulaire générera, avec le code PHP précédent, le code HTML :

```

1 <p>Hello Mr John Doe!</p>

```



*On vérifie que les paramètres `nom`, `prenom` et `civilite` existent. On vérifie également que les valeurs des deux premiers sont composées uniquement de lettres, chiffres et underscores et que la valeur de `civilite` est "h" ou "f".*

► Sur ce thème : **Exercice 11**

### 1.2.3 Utilisation de `$_POST`

Lors de la soumission du formulaire avec la méthode précédente, les paramètres se trouvent dans l'URL. Ceci n'est pas forcément très lisible et il peut être préférable de cacher ces éléments. Dans ce cas, il suffit d'utiliser la méthode `post` dans la déclaration du formulaire, par exemple : `<form action="traitementDonnees.php" method="post">`. Les informations saisies dans le formulaire n'apparaissent alors plus dans l'URL. Elles sont cependant récupérables dans le script appelé lors de la soumission du formulaire (ici `traitementDonnees.php`) dans le tableau `$_POST`. Elles n'apparaissent par contre plus dans le tableau `$_GET`.



### 1.2.4 Les cookies

Il est possible de stocker des informations sur le **client** dans des *cookies*. Comme l'utilisateur peut voir et modifier ses cookies, on ne peut pas stocker dedans des informations sensibles (par exemple, des informations indiquant si l'utilisateur est connecté ou quels sont ses droits sur le site). L'intérêt des cookies est qu'ils peuvent être stockés autant de temps que nécessaire. C'est au moment de la création du cookie que l'on spécifie sa date d'expiration.

Comme il est possible de créer plusieurs cookies pour un même utilisateur, on stockera une unique information dans un cookie. La création d'un cookie se fait à l'aide de la fonction

```
1 <?php
2 setcookie(nom, valeur, date_expiration, null, null, false, true);
3 ?>
```

qui doit être appelée **avant tout envoi de code HTML**, c'est-à-dire, avant tout code HTML et avant toute instruction `echo`. Si cette fonction peut paraître très compliquée à cause du nombre de paramètres, elle s'utilise très facilement. Les deux premiers paramètres sont des chaînes de caractères contenant respectivement le nom et la valeur du cookie. La date d'expiration est donnée par la fonction PHP `time()` plus la durée en secondes du temps de conservation du cookie. Les autres valeurs de paramètres sont toujours identiques.

Ainsi, si je souhaite créer un cookie contenant le pseudo d'un utilisateur, disons `toto`, et que je souhaite que ce cookie soit automatiquement supprimé au bout d'une semaine, je dois alors appeler la fonction `setcookie()` de la manière suivante :

```
1 setcookie('pseudo', 'toto', time() + 7 * 24 * 3600, null, null, false, true);
```

Pour modifier la valeur d'un cookie, il suffit de recréer le cookie, l'ancien étant automatiquement supprimé.

Les informations contenues dans les cookies peuvent être récupérées à l'aide du tableau `$_COOKIE`. Si je souhaite récupérer et afficher la valeur du cookie précédent, je dois alors écrire

```
1 echo '<p> Bienvenue ' . $_COOKIE['pseudo'] . '</p>';
```



*Comme les cookies présents sur le client sont chargés avant toute instruction PHP, un cookie créé avec la fonction `setcookie()` ne sera pas visible (i.e., n'apparaîtra pas dans le tableau `$_COOKIE`) avant le prochain chargement de page.*

### 1.2.5 Les sessions

Les sessions permettent de stocker des informations sur le **serveur** "le temps de la connexion de l'utilisateur sur le site". Plus précisément, si les sessions sont activées, lorsqu'un utilisateur se connecte, une variable globale `$_SESSION` (tableau) est créée sur le **serveur** pour cet utilisateur si elle n'existait pas déjà. De cette manière, il y a une variable `$_SESSION` par utilisateur. Les données stockées dans la variable `$_SESSION` sont accessibles pour tous les scripts (activant les sessions) que l'utilisateur exécute sur le site.

Les informations stockées dans la variable `$_SESSION` sont automatiquement détruites lorsque l'utilisateur n'accède plus au serveur pendant un certain temps. Cette durée dépend de la configuration du serveur. Elle est de l'ordre d'une demi-heure<sup>6</sup>.

Pour activer les sessions, il faut ajouter sur la **première ligne** des fichiers PHP où vous souhaitez utiliser la variable `$_SESSION` l'instruction :

```
1 <?php session_start(); ?>
```

Voici un exemple (incomplet) d'utilisation de la variable `$_SESSION`. Lorsque l'utilisateur se connecte à un site via un formulaire, si la connexion réussit, le script appelé lors de la soumission du formulaire peut alors stocker les informations dans la variable `$_SESSION` :



```
1 //Code PHP exécuté lors de la soumission du formulaire
2 //On teste si l'utilisateur a correctement saisi son nom et son prénom
3 if( isset($_POST['nom']) and isset($_POST['prenom'])
4     and preg_match("#^\w+$#", $_POST['prenom'])
5     and preg_match("#^\w+$#", $_POST['nom']) {
6     echo '<p> Connexion réussie </p>';
7
8     $_SESSION['nom'] = $_POST['nom'];
9     $_SESSION['prenom'] = $_POST['prenom'];
10    $_SESSION['connecte'] = true;
11 }
```

Les informations stockées dans la variable `$_SESSION` sont relativement sûres. (Le risque zéro n'existe pas!) En effet, comme ces informations sont stockées sur le serveur, il est difficile de pirater ces informations si le site est correctement configuré.

► Sur ce thème : **Exercice 12**

## Références

Vous pouvez trouver de nombreux cours et tutoriels sur le Web. Voici quelques liens intéressants :

- le site PHP (manuel) (<https://php.net/manual/fr/>),
- le site PHP facile (<http://www.lephpfacile.com/cours/>).

## Notes de cours

1. Ce cours nécessite de connaître les langages HTML et CSS vus l'année dernière. Pour réviser ces deux langages, vous pouvez notamment lire le cours HTML/CSS se trouvant sur la page <http://lipn.univ-paris13.fr/~lacroix/>.
2. <http://fr.openclassrooms.com/informatique/cours/programmez-en-orientee-objet-en-php> par exemple
3. Pendant l'exécution d'une fonction/méthode, on peut par contre connaître le nombre de paramètres passés lors de son appel.
4. Il est possible de choisir un autre symbole délimiteur.
5. La fonction `preg_match` peut également prendre un tableau en troisième paramètre pour récupérer les différentes parties de la chaîne satisfaisant les différents éléments (définis par des parenthèses) de l'expression régulière.
6. Il est possible de détruire dans le code une session sans attendre, comme cela est fait lorsque l'utilisateur se déconnecte du site de l'IUT par exemple.

# TD1: PHP

## Exercice 1 Variables et conversion

**Question 1.1 :** Qu'affiche le code suivant ?

```
1  var_dump(1 + 2)    ; echo '<br/>';
2  var_dump('1' + '2'); echo '<br/>';
3  var_dump(1 . 2)    ; echo '<br/>';
4  var_dump('1' . '2'); echo '<br/>';
5  var_dump("3 fruits" + '1 légume' . 'donnent'); echo '<br/>';
6  var_dump("3 fruits" . '1 légume' + 'donnent'); echo '<br/>';
7  var_dump(1/2 + true - "-2"); echo '<br/>';
```

**Question 1.2 :** Considérons les variables `$voitures = 3`; `$motos = 6`; . Afficher, à l'aide d'une seule instruction PHP, le paragraphe : *J'ai 3 voitures et 6 motos, j'ai donc 9 véhicules.* Les nombres 3, 6 et 9 dépendent des valeurs des variables données précédemment. Le faire une fois en utilisant uniquement les apostrophes et une fois en utilisant uniquement les guillemets.

## Exercice 2 Boucles

Écrire la table de multiplication de 7 (de 1 à 10). Cette table sera présentée d'abord sous forme d'une liste non ordonnée puis sous la forme d'une table HTML.

## Exercice 3 Tableaux

**Question 3.1 :** Soit le tableau `$t = ['a'=>'z', 10=>20, 'toto'=>'titi', 'php']`. Quelles sont les clés du tableau ? Quelles sont les valeurs ?

**Question 3.2 :** Définir une variable de type tableau. Ce tableau devra contenir tous les types de données PHP possibles pour les clés et pour les valeurs.

**Question 3.3 :** Ajouter à ce tableau deux valeurs `'jour'` et `'nuit'`. La clé associée à la première valeur sera `'bon'`. La deuxième clé sera donnée par l'interpréteur PHP. Quelle clé donnera-t-il ?

## Exercice 4 Parcours de tableaux

Le tableau suivant donne les noms des étudiants et leur note en PHP.

```
1  $tabE1 = [
2      'Frédérique' => 12,
3      'Silvia' => 3,
4      'Julien' => 19,
5      'Lionel' => 12,
6      'Mario' => 12,
7      'Andrea' => 9,
8      'Gérard' => 3
9  ];
```

**Question 4.1 :** Afficher sous forme d'une liste ordonnée les noms des différents étudiants.

**Question 4.2 :** Afficher sous forme d'une table HTML les noms et les notes des étudiants.

**Question 4.3 :** Créer un autre tableau `$tabE2` dans lequel les notes sont maintenant les clés.

**Question 4.4 :** Refaire les deux premières questions avec le tableau `$tabE2`.

## Exercice 5 Fonctions sur les tableaux I

Considérons le tableau suivant.

```
1 $anneeScolaire = [  
2   "Rentrée" => [2,9,2019],  
3   "Vacances Toussaint" => [  
4       "début" => [19,10,2019],  
5       "fin" => [4,11,2019]  
6   ],  
7   "Vacances Noël" => [  
8       "début" => [21,12,2019],  
9       "fin" => [6,1,2020]  
10  ],  
11  "Vacances d'hiver" => [  
12      "début" => [8,2,2020],  
13      "fin" => [24,2,2020]  
14  ],  
15  "Vacances de printemps" => [  
16      "début" => [4,4,2020],  
17      "fin" => [20,4,2020]  
18  ],  
19  "Fin des cours" => [4,7,2020]  
20 ];
```

Écrire le code permettant d'obtenir l'affichage suivant.

```
1 - Rentrée : 2/9/2019  
2 - Vacances Toussaint : du 19/10/2019 au 4/11/2019  
3 - Vacances Noël : du 21/12/2019 au 6/1/2020  
4 - Vacances d'hiver : du 8/2/2020 au 24/2/2020  
5 - Vacances de printemps : du 4/4/2020 au 20/4/2020  
6 - Fin des cours : 4/7/2020
```



On utilisera uniquement deux boucles `foreach` imbriquées ainsi que les fonctions `count` et `implode`.

## Exercice 6 Fonctions sur les tableaux II

**Question 6.1 :** Qu'affiche le code suivant ?

```
1 $tab = [ 2 => 'test', 0 => 'nombre' ];  
2  
3 if (in_array(2, $tab))  
4     echo '<p> in_array : 2 est dans le tableau </p>';  
5  
6 if(in_array('nombre', $tab))  
7     echo '<p> in_array : nombre est dans le tableau </p>';
```

**Question 6.2 :** Dans le `if`, peut-on remplacer `in_array` par `array_search` ?

## Exercice 7 Fonctions

Considérons le tableau suivant :

```

1 $joueurs = [
2   ['nom' => 'Mehwish', 'score' => 150],
3   ['nom' => 'Laurent', 'score' => 120],
4   ['nom' => 'Ines', 'score' => 98],
5   ['nom' => 'Sondes', 'score' => 153],
6   ['nom' => 'Davide', 'score' => 118]
7 ];

```

Écrire une fonction `meilleur_joueur()` prenant en paramètre un tableau ayant la même structure que le tableau `$joueurs` et retournant un tableau contenant le nom et le score du meilleur joueur.

## Exercice 8 Objets

Définir une classe `Compte` représentant un compte bancaire. Cette classe contiendra deux attributs privés, `montant` et `interet` (annuel), un constructeur prenant en paramètre un montant initial et un intérêt, une méthode `get_montant` retournant le montant du compte, ainsi qu'une méthode `un_an` modifiant le montant du compte au bout d'un an en fonction de l'intérêt.

Créer deux comptes, un d'un montant de 200 euros à 20% et l'autre d'un montant de 1000 euros à 2%. Afficher le montant de chaque compte au bout de 10 ans.

## Exercice 9 Expressions régulières

**Question 9.1 :** Donner l'expression régulière correspondant à un nombre entier. Il est possible d'avoir un nombre négatif.

**Question 9.2 :** Donner l'expression régulière correspondant à un nombre décimal. On utilisera le symbole `.` comme séparateur décimal. Pour simplifier, on supposera qu'il y a forcément un chiffre après le séparateur décimal si celui-ci est donné. Les valeurs 10.0, 3.14, -42, 0.001, .001 doivent satisfaire l'ER mais 10. non.

**Question 9.3 :** Donner l'expression régulière correspondant aux dates au format JJ/MM/AAAA. Il est possible de ne spécifier qu'un chiffre pour le jour et le mois.

**Question 9.4 :** Parmi les différents ensembles de chaînes de caractères, lequel contient uniquement des chaînes satisfaisant l'expression régulière `'#[bon]?j+(our)?#'` ?

- ☐ 'bonjour', 'jour', 'j'
- ☐ 'bjour', 'jour', 'jr'
- ☐ 'our', 'jour', 'j'

**Question 9.5 :** Parmi les différents ensembles de chaînes de caractères, lequel contient uniquement des chaînes satisfaisant l'expression régulière `'#^a.b(ba{3}|c)$#'` ?

- ☐ 'a.bc', 'abc', 'anbc'
- ☐ 'a.bc', 'aabbaaa', 'anbc'
- ☐ 'aabc', 'aabbababa', 'anbc'

## Exercice 10 Passage de paramètres dans l'url

**Question 10.1 :** Créer un script `nombre.php` qui teste s'il existe un paramètre de nom `nombre`. Si c'est le cas, alors le script teste si sa valeur est un nombre ou pas et affiche cette information.

**Question 10.2 :** Donner l'url appelant le script `nombre.php` en lui passant le nombre 25. On suppose que `nombre.php` se trouve sur le serveur `serveurWeb` à la racine.

## Exercice 11 Formulaire

Créer un formulaire avec un champ de type texte (pour saisir un nombre) et un bouton `submit`. La soumission du formulaire appellera le fichier `nombre.php` défini dans l'exercice précédent.

## Exercice 12 Sessions et cookies

On suppose que le formulaire précédent est dans le fichier `nombre.php` défini dans l'exercice 10.

**Question 12.1 :** Utiliser les sessions pour faire la multiplication des nombres saisis dans le formulaire.

**Question 12.2 :** Pourquoi est-ce important de vérifier que le nombre saisi correspond à un nombre ? Que se passerait-il si aucune vérification n'était faite et que l'utilisateur soumettait le formulaire sans avoir saisi de nombre ?

**Question 12.3 :** Modifier le code de manière à pouvoir réinitialiser le produit des nombres en appuyant sur un autre bouton de type `submit`.

**Question 12.4 :** À l'aide des cookies, modifier le script pour qu'à chaque fois que l'on réinitialise un produit, la valeur de ce produit (s'il est différent de 1) soit stocké dans un cookie. Afficher à la fin du script la valeur du dernier produit calculé avant réinitialisation si cette information est disponible.