



ECOLE MAROCAINE DES
SCIENCES DE L'INGENIEUR
Membre de
HONORIS UNITED UNIVERSITIES



Langage de Script PHP

Chapitre 1

Pr. Zainab OUFQIR

Z.Oufkir@emsi.ma



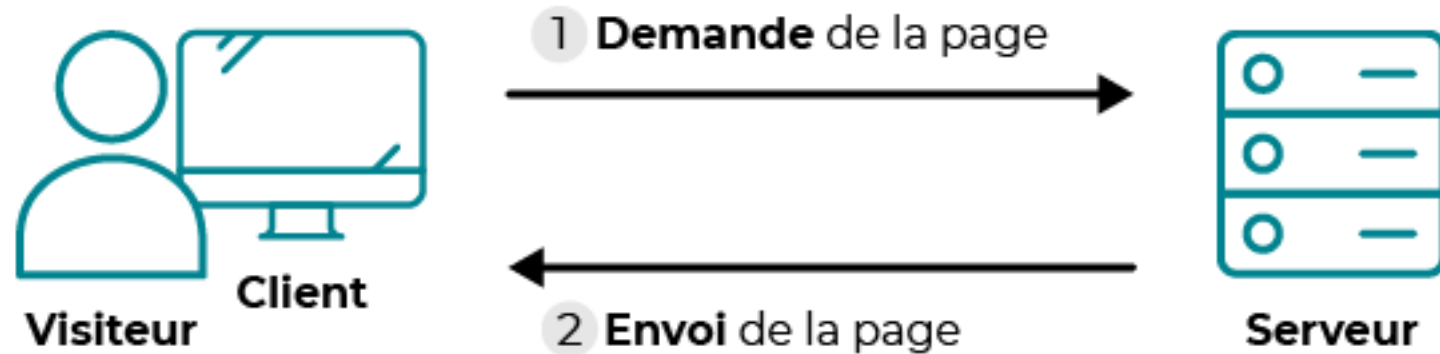
Introduction

- Faites la différence entre site statique et dynamique, On considère qu'il existe deux types de sites web :
 - Les **sites statiques** sont réalisés uniquement à l'aide des langages **HTML**, **CSS** et **JavaScript**. Il fonctionne très bien, mais son contenu ne peut pas être mis à jour automatiquement.
 - Et les **sites dynamiques** utilisent d'autres langages comme **PHP**, **Java**, **Python**. Le contenu de ce type de site est dit « dynamique » parce qu'il peut changer automatiquement.

Consultez un site statique

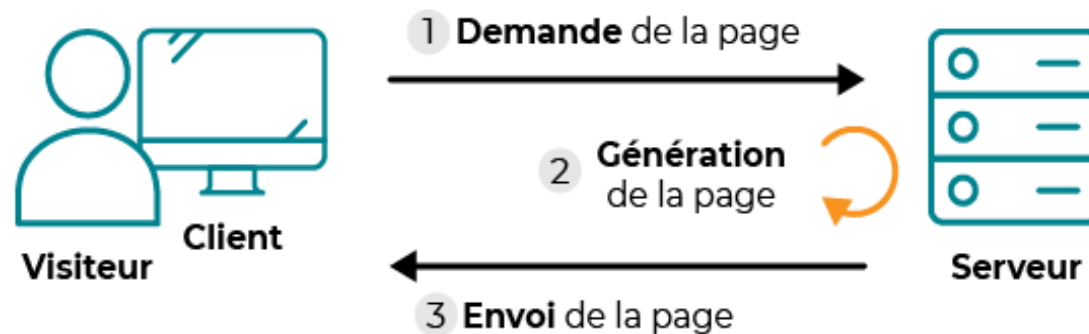
➤ Lorsque vous vous rendez sur site statique, c'est très simple. Cela se passe en deux temps :

1. Le **client** demande au serveur à voir une page web.
2. Le **serveur** lui répond en lui envoyant la page réclamée.



Consultez un site dynamique

- Lorsque vous consultez un site dynamique, la page est **générée** côté **serveur**.
- Il y a une étape supplémentaire, et elle se situe entre les deux étapes de base :
 1. Le **client** demande au serveur à voir une page web.
 2. Le serveur prépare la page spécialement pour le client (il la génère).
 3. Le **serveur** lui répond en lui envoyant la page réclamée.



PHP Hypertext Preprocessor

- **PHP** (PHP Hypertext Preprocessor) est un langage que **seuls les serveurs comprennent**, et qui permet de rendre un site dynamique. C'est PHP qui « **génère** » la page web.
- Le PHP étant un **langage script**, ce qui signifie que le code est **interprété** et non pas compilé comme le langage C ou C++ .

Outils

- Pour que votre ordinateur puisse lire du PHP, il faut qu'il **se comporte comme un serveur**.
- Il suffit simplement d'installer les mêmes programmes que ceux que l'on trouve sur les serveurs qui délivrent les sites web aux internautes. Ces programmes sont:
 - **Apache**: C'est ce qu'on appelle un serveur web. **Il est chargé de délivrer les pages web aux visiteurs**. Cependant, Apache ne gère que les sites web statiques (il ne peut traiter que des pages HTML). Il faut donc le compléter avec d'autres programmes.
 - **PHP**: C'est un plug-in pour Apache qui le rend capable de **traiter des pages web dynamiques** en PHP. En clair, en combinant Apache et PHP, notre ordinateur sera capable de lire des pages web en PHP.

Outils

- **MySQL**: C'est le logiciel de **gestion de bases de données**. Il permet d'enregistrer des données de manière organisée.
- Nous allons installer des logiciels dans notre machine qui reproduisent le comportement exact d'un serveur.



PHP et HTML

- Pour utiliser PHP, on utilise la balise `<?php ?>`, on peut la placer n'importe où dans le code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ceci est une page de test avec des balises PHP</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h2>Page de test</h2>
    <p>
      Cette page contient du code HTML avec des balises PHP.<br />
      <?php /* Insérer du code PHP ici */ ?>
    </p>
    <?php
      /* Encore du PHP
      Toujours du PHP */
    ?>
  </body>
</html>
```


PHP et HTML

- Le code (HTML) **non contenu** dans `<?php ?>` est **transmis tel quel** au client.
- Le code (PHP) **contenu** dans `<?php ?>` est **exécuté** et le **résultat** transmis au client.

```
1 <?php echo "Ceci est du texte"; ?>
2
3 <!-- Ou bien, avec des parenthèses -->
4 <?php echo("Ceci est du texte"); ?>
```

- Les pages web contenant du PHP ont l'extension **.php**
- On ne peut pas visualiser **un script PHP** avec le protocole **file** : **rien ne s'affiche** ou bien on voit le source du fichier.

PHP et HTML

- On peut aussi demander d'afficher des balises. Par exemple, le code suivant fonctionne :

```
1 <?php echo "Ceci est du <strong>texte</strong>"; ?>
```

- Pour afficher des guillemets, il faut précéder le guillemet d'un antislash \ :

```
1 <?php echo "Cette ligne a été écrite \"uniquement\" en PHP."; ?>
```

Affichage web

- Enregistrez la page avec l'extension **.php** par exemple : **bonjour.php** et la mettre dans dossier **tests**
- Le dossier tests doit se trouver dans **C:\wamp64\www\tests** sous **Windows**.
- Testez la page PHP:
 1. Démarrez **WAMP** si ce n'est pas déjà fait.
 2. Allez à l'adresse **http://localhost/tests**. Une page web s'ouvre, indiquant tous les fichiers qui se trouvent dans le dossier **tests** . Vous devriez avoir le fichier **bonjour.php** .
 3. Cliquez dessus : votre ordinateur **génère** alors le code PHP puis ouvre la page.

Affichage web

- Ce n'est pas plus simple de l'écrire en HTML, finalement ?

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8" />
5     <title>Ma page web</title>
6   </head>
7   <body>
8     <h1>Ma page web</h1>
9     <p>Aujourd'hui nous sommes le <?php echo date('d/m/Y h:i:s'); ?>.</p>
10  </body>
11 </html>
```

La date et l'heure s'affichent **automatiquement** sur la page web !

Les commentaires

➤ Il existe deux types de commentaire :

➤ Les commentaires **monolignes**.

```
1 <?php
2 echo "J'habite en Chine."; // Cette ligne indique où j'habite
3
4 // La ligne suivante indique mon âge
5 echo "J'ai 92 ans.";
6 ?>
```

➤ Les commentaires **multilignes**.

```
1 <?php
2 /* La ligne suivante indique mon âge
3 Si vous ne me croyez pas...
4 ... vous avez raison ;o) */
5 echo "J'ai 92 ans.";
6 ?>
```

Les variables

- Une **variable**, c'est une information **stockée en mémoire temporairement**.
- En **PHP**, la **variable** (l'information) **existe tant que la page est en cours de génération**. Dès que la page PHP est générée, toutes les variables sont supprimées de la mémoire.
- Une variable est toujours constituée de deux éléments :
 - **son nom** : pour pouvoir la reconnaître, vous devez donner un nom à votre variable. Par exemple age;
 - **sa valeur** : c'est l'information qu'elle contient, et qui peut changer. Par exemple : 17 .

Les variables

- Les **variables** sont capables de stocker différents types d'informations. On parle de **types de données**. Voici les principaux types à connaître:
 - **Les chaînes de caractères** (**string**) : c'est le nom informatique qu'on donne au texte.
 - **Les nombres entiers** (**int**) : ce sont les nombres du type 1, 2, 3, 4, etc. On compte aussi parmi eux les entiers relatifs : -1, -2, -3...
 - **Les nombres décimaux** (**float**) : ce sont les nombres à virgule, comme 14,738. Attention, les nombres doivent être écrits avec un point au lieu de la virgule.
 - **Les booléens** (**bool**) : c'est un type très important qui permet de stocker soit **true** soit **false**.
 - **Rien** (**NULL**) : On a parfois besoin de dire qu'une variable ne contient rien. Ce n'est pas vraiment un type de données, mais plutôt l'absence de type.

Les variables

- Affectez une valeur à une variable:

```
1 <?php
2 $userAge = 17;
3 ?>
```

- Avec ce code PHP, on vient en fait de créer une variable :
 - son **nom** est `userAge`;
 - sa **valeur** est `17` .
- On utilise la convention **camelCase** pour nommer les variables. Notez qu'on ne peut pas mettre d'espace dans un nom de variable. On utilise donc une **majuscule** pour "**détacher**" visuellement les mots et les rendre plus lisibles.

Les variables

- Le type **string** (chaîne de caractères)

```
1 <?php
2 $email = "Z.Oufkir@emsi.ma"; // 'Z.Oufkir@emsi.ma'
3 ?>
```

- Le type **int** (nombre entier)

```
1 <?php
2 $userAge = 17;
3 ?>
```

- Le type **float** (nombre décimal)

```
1 <?php
2 $price = 57.3;
3 ?>
```

Les variables

➤ Le type **bool** (booléen)

```
1 <?php
2 $isAuthor = true;
3 $isAdministrator = false;
4 ?>
```

➤ Une variable **vide** avec **NULL**

```
1 <?php
2 $noValue = NULL;
3 ?>
```

Les variables

- Pour afficher le contenu d'une variable, on utilise **echo**

```
1 <?php
2 $email = "Z.Oufkir@emsi.ma";
3 echo $email;
4 ?>
```

- Quand il s'agit d'une variable, **on ne met pas de guillemets autour.**

Les variables

- **Concaténez** avec des **guillemets doubles**: Vous pouvez écrire le nom de la variable au milieu du texte et il sera remplacé par sa valeur.

```
1 <?php
2 $firstname = "Zainab";
3 echo "Bonjour $firstname et bienvenue sur le site !";
4 ?>
```

```
1 <?php
2 $firstname = "Zainab";
3 echo "Bonjour {$firstname} et bienvenue sur le site !";
4 ?>
```

- Ça affiche : **Bonjour Zainab** et bienvenue sur le site !

Les variables

- On ne peut pas concaténer du texte avec des guillemets simples !

```
1 <?php
2 $firstname = 'Zainab';
3 echo 'Bonjour $firstname et bienvenue sur le site !'; // ERREUR !
4 ?>
```

- Ça affiche : Bonjour \$firstname et bienvenue sur le site ! .

Les variables

- **Concaténez** avec des **guillemets simples**: il faut écrire la variable en dehors des guillemets et séparer les éléments les uns des autres à l'aide d'un point.

```
1 <?php
2 $firstname = 'Zainab';
3 echo 'Bonjour ' . $firstname . ' et bienvenue sur le site !'; // OK
4 ?>
```

- Ça affiche : **Bonjour Zainab** et bienvenue sur le site !

Les opérations

➤ Les opérations de base :

Symbole	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo

Les opérations

➤ Exemple:

```
1 <?php
2 $number = 2 + 4; // $number prend la valeur 6
3 $number = 5 - 1; // $number prend la valeur 4
4 $number = 3 * 5; // $number prend la valeur 15
5 $number = 10 / 2; // $number prend la valeur 5
6
7 $number = 3 * 5 + 1; // $number prend la valeur 16
8 $number = (1 + 2) * 2; // $number prend la valeur 6
9
10 $number = 10 % 5; // $number prend la valeur 0 car la division tombe juste
11 $number = 10 % 3; // $number prend la valeur 1 car il reste 1
12 ?>
```


Les opérations

➤ L'ordre des opérations:

Order of Operations		
B	Brackets	$10 \times (4 + 2) = 10 \times 6 = 60$
I	Indices	$5 + 2^2 = 5 + 4 = 9$
D	Division	$10 + 6 \div 2 = 10 + 3 = 13$
M	Multiplication	$10 - 4 \times 2 = 10 - 8 = 2$
A	Addition	$10 \times 4 + 7 = 40 + 7 = 47$
S	Subtraction	$10 \div 2 - 3 = 5 - 3 = 2$


Les conditions

➤ les symboles à connaître:

Symbole	Signification
==	Est égal à
>	Est supérieur à
<	Est inférieur à
>=	Est supérieur ou égal à
<=	Est inférieur ou égal à
!=	Est différent de

Utilisez la structure if... else



- Voici ce qu'on doit écrire, dans l'ordre, pour utiliser cette condition:
 - Pour introduire une condition, on utilise le mot **if** qui signifie « **si** », en anglais.
 - On ajoute à la suite entre parenthèses la condition en elle-même.
 - Enfin, on ouvre des accolades à l'intérieur desquelles on placera les instructions à exécuter si la condition est remplie.

```
1 <?php
2 $isEnabled = true; // La condition d'accès
3
4 if ($isEnabled == true) {
5     echo "Vous êtes autorisé(e) à accéder au site ";
6 }
7 ?>
```

Utilisez la structure if... else

➤ il y a deux possibilités:

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

```
1 <?php
2 $isEnabled = true;
3
4 if ($isEnabled == true) {
5     echo "Vous êtes autorisé(e) à accéder au site ";
6 }
7 else {
8     echo "Accès refusé  ";
9 }
10 ?>
```

Utilisez la structure if... else

➤ Dans l'ordre, PHP rencontre les conditions suivantes :

1. Si `$isAllowedToEnter` est égale à « Oui », tu exécutes ces instructions....
2. Sinon, si `$isAllowedToEnter` est égale à « Non », tu exécutes ces autres instructions...
3. Sinon, affiche ce message.

```
1 <?php
2 $isAllowedToEnter = "Oui";
3
4 // SI on a l'autorisation d'entrer
5 if ($isAllowedToEnter == "Oui") {
6     // instructions à exécuter quand on est autorisé à entrer
7 } // SINON SI on n'a pas l'autorisation d'entrer
8 elseif ($isAllowedToEnter == "Non") {
9     // instructions à exécuter quand on n'est pas autorisé à entrer
10 } // SINON (la variable ne contient ni Oui ni Non, on ne peut pas agir)
11 else {
12     echo "Euh, je ne comprends pas ton choix, tu peux me le rappeler s'il te plaît ?";
13 }
14 ?>
```

Les tableaux numérotés

➤ Il s'agit en fait de **variables** « **composées** », que l'on peut imaginer sous la forme de tableaux.

Regardez par exemple celui-ci, contenu de la variable **\$recipes** :

Clé	Valeur
0	Cassoulet
1	Couscous
2	Escalope milanaise
3	Salade César
4	Bo bun
...	...

➤ **Attention !** Un tableau numéroté commence toujours à la case n° 0 !

Les tableaux numérotés

- Pour créer un tableau **numéroté** en PHP, on liste ses valeurs entre crochets **[]**.
- Cet exemple vous montre comment créer le tableau **\$recipes** :

```
1 <?php
2
3 $recipes = ['Cassoulet', 'Couscous', 'Escalope Milanaise', 'Salade César',,];
4
5 // La fonction array permet aussi de créer un tableau
6 $recipes = array('Cassoulet', 'Couscous', 'Escalope Milanaise');
7 ?>
```

- L'ordre a beaucoup d'importance :
 - Le premier élément (« **Cassoulet**») aura le n° **0**.
 - Ensuite **Couscous** le n° **1**.
 - Etc.

Les tableaux numérotés

- On peut aussi créer manuellement le tableau, case par case :

```
1 <?php
2 $recipies[0] = 'Cassoulet';
3 $recipies[1] = 'Couscous';
4 $recipies[2] = 'Escalope Milanaise';
5 ?>
```

- On peut ne pas écrire le numéro de la case que nous allons créer, c'est possible de laisser PHP le sélectionner automatiquement en laissant les crochets vides :

```
1 <?php
2 $recipies[] = 'Cassoulet'; // Créera $recipies[0]
3 $recipies[] = 'Couscous'; // Créera $recipies[1]
4 $recipies[] = 'Escalope Milanaise'; // Créera $recipies[2]
5 ?>
```


Les tableaux numérotés

- Pour afficher un élément, il faut **donner sa position entre crochets** après **\$recipes**.

```
1 <?php
2 echo $recipes[1]; // Cela affichera : Couscous
3 ?>
```

- Cela revient à dire à PHP : « Affiche-moi le contenu de la case n° 1 de \$recipes »

Les tableaux numérotés

- Pour afficher tous le tableau, nous utilisons `print_r` (utilisée principalement pour le débogage).

```
1 <?php
2 print_r($recipes);
3 ?>
```

- Cela affiche le tableau sous cette forme:

Array ([0] => Cassoulet [1] => Couscous [2] => Escalope Milanaise [3] => Salade César)

Les tableaux associatifs

- Les tableaux **associatifs** fonctionnent sur le même principe, sauf qu'au lieu de numéroter les cases, on va les **étiqueter en leur donnant à chacune un nom différent**.

```
1 <?php
2 // Une bien meilleure façon de stocker une recette !
3 $recipe = [
4     'title' => 'Cassoulet',
5     'recipe' => 'Etape 1 : des flageolets, Etape 2 : ...',
6     'author' => 'john.doe@example.com',
7     'enabled' => true,
8 ];
9
10 ?>
```

- On écrit une flèche (=>) pour dire « **associé à** ».

Les tableaux associatifs

➤ Par exemple, on dit que la propriété « **title** » du tableau `$recipe` a pour valeur « **Cassoulet** ».

Clé	Valeur
title	Cassoulet
recipe	Étape 1 : des flageolets, Étape 2 : ...
author	john.doe@exemple.com
enabled	true

Les tableaux associatifs

➤ Il est aussi possible de créer le tableau case par case, comme ceci :

```
1 <?php
2 $recipe['title'] = 'Cassoulet';
3 $recipe['recipe'] = 'Etape 1 : des flageolets, Etape 2 : ...';
4 $recipe['author'] = 'john.doe@exemple.com';
5 $recipe['enable'] = true;
6 ?>
```

Les tableaux associatifs

- Pour afficher un élément, il suffit d'indiquer le nom de cet élément entre crochets, ainsi qu'entre guillemets ou apostrophes, puisque l'étiquette du tableau associatif est un texte:

```
1 <?php
2 echo $recipe['title']; // Cela affichera : Cassoulet
3 ?>
```

- Pour afficher tous le tableau, nous utilisons `print_r` .

```
1 <?php
2 print_r($recipe);
3 ?>
```

- Cela affiche le tableau sous cette forme:

Array ([title] => Cassoulet [recipe] => Etape 1 : des flageolets, Etape 2 : ... [author] => john.doe@exemple.com [enabled] => 1)

Les tableaux multidimensionnels

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

```
1 <?php
2
3 /**
4  * Déclaration du tableau des recettes
5  * Chaque élément du tableau est un tableau numéroté (une recette)
6  */
7 $recipes = [
8     ['Cassoulet', 'mickael.andrieu@exemple.com', true],
9     ['Couscous', 'mickael.andrieu@exemple.com', false],
10 ];
11
```

Les tableaux multidimensionnels

➤ Pour afficher tous le tableau, nous utilisons `print_r` .

```
1 <?php
2 print_r($recipes);
3 ?>
```

```
Array
(
    [0] => Array
        (
            [0] => Cassoulet
            [1] => mickael.andrieu@exemple.com
            [2] => 1
        )
    [1] => Array
        (
            [0] => Couscous
            [1] => mickael.andrieu@exemple.com
            [2] =>
        )
)
```


Les boucles

- Lorsqu'un **tableau** a été créé, on a souvent besoin de le **parcourir** pour savoir ce qu'il contient. Nous allons voir deux moyens d'explorer un tableau :
 - La boucle **for**
 - La boucle **foreach**

Les boucles

- Quand on écrit `$recipes[$lines]`, la variable `$lines` est d'abord remplacée par sa valeur.
- Si `$lines` vaut `1`, cela signifie qu'on cherche ce que contient `$recipes[1][0]`, c'est-à-dire :
`Couscous`.

```
1 <?php
2
3 /**
4  * Déclaration du tableau des recettes
5  * Chaque élément du tableau est un tableau numéroté (une recette)
6  */
7 $recipes = [
8     ['Cassoulet','mickael.andrieu@exemple.com',true],
9     ['Couscous','mickael.andrieu@exemple.com',false],
10 ];
11
12 for ($lines = 0; $lines <= 1; $lines++) {
13     echo $recipes[$lines][0];
14 }
```

Les boucles

- **foreach** passe en revue chaque ligne du tableau. Lors de chaque passage, elle met la valeur de cette ligne dans **une variable temporaire** (par exemple **\$element**).

```
1 <?php
2
3 // Déclaration du tableau des recettes
4 $recipes = [
5     ['Cassoulet', 'mickael.andrieu@exemple.com', true, ],
6     ['Couscous', 'mickael.andrieu@exemple.com', false, ],
7 ];
8
9 foreach ($recipes as $recipe) {
10     echo $recipe[0]; // Affichera Cassoulet, puis Couscous
11 }
```

Les boucles

➤ **foreach** permet aussi de parcourir les tableaux associatifs.

```
1 <?php
2 $recipe = [
3     'title' => 'Cassoulet',
4     'recipe' => 'Etape 1 : des flageolets, Etape 2 : ...',
5     'author' => 'mickael.andrieu@exemple.com',
6     'enabled' => true,
7 ];
8
9 foreach ($recipe as $value) {
10     echo $value;
11 }
12
13 /**
14  * AFFICHE
15  * CassouletEtape 1 : des flageolets, Etape 2 : ...mickael.andrieu@exemple.com1
16  */
```

Les boucles

```
1 <?php
2
3 $recipes = [
4     [
5         'title' => 'Cassoulet',
6         'recipe' => '',
7         'author' => 'mickael.andrieu@exemple.com',
8         'is_enabled' => true,
9     ],
10    [
11        'title' => 'Couscous',
12        'recipe' => '',
13        'author' => 'mickael.andrieu@exemple.com',
14        'is_enabled' => false,
15    ],
16 ];
17
18 foreach($recipes as $recipe) {
19     echo $recipe['title'] . ' contribué(e) par : ' . $recipe['author'];
20 }
```

Cassoulet contribué(e) par : mickael.andrieu@exemple.com

Couscous contribué(e) par : mickael.andrieu@exemple.com

Les boucles

- Toutefois, avec cet exemple, on ne récupère que **la valeur**. Or, on peut aussi récupérer la **clé** de l'élément. On doit dans ce cas écrire **foreach**, comme ceci :

```
1 <?php foreach($recipe as $property => $propertyValue) ?>
```

- À chaque tour de boucle, on disposera non pas d'une, mais de deux variables :
1. **\$property** qui contiendra **la clé** de l'élément en cours d'analyse (« **title** », « **author** », etc.).
 2. **\$propertyValue** qui contiendra **la valeur** de l'élément en cours (« **Cassoulet** », « **laurene.castor@example.com** », etc.).

Les boucles

➤ Exemple

```
1 <?php
2 $recipe = [
3     'title' => 'Salade Romaine',
4     'recipe' => 'Etape 1 : Lavez la salade ; Etape 2 : euh ...',
5     'author' => 'laurene.castor@exemple.com',
6 ];
7
8 foreach($recipe as $property => $propertyValue)
9 {
10     echo '[' . $property . '] vaut ' . $propertyValue;
11 }
```

[title] vaut Salade Romaine

[recipe] vaut Etape 1 : Lavez la salade ; Etape 2 : euh ...

[author] vaut laurene.castor@exemple.com

Les boucles

- La boucle **while** peut se traduire par « **tant que** ».
- Ici, on demande à PHP **TANT QUE** **\$isValid** est **vrai**, exécuter ces instructions.

```
1 <?php
2 $lines = 1;
3
4 while ($lines <= 100) {
5     echo 'Je ne dois pas regarder les mouches voler quand j\'apprends le PHP.<br>';
6     $lines++; // $lines = $lines + 1
7 }
8 ?>
```


Les boucles

➤ Ce qui affiche... un grand nombre de lignes :

[illegible]

Les boucles

- La boucle pose la condition **TANT QUE** `$lines` est **inférieur ou égal à 100**.
- Dans cette boucle, il y a deux instructions :
 1. **echo** permet d'afficher du texte en PHP. À noter qu'il y a une balise HTML **
** à la fin : cela permet d'aller à la ligne.
 2. **`$lines++`**; est une façon plus courte d'ajouter 1 à la variable. On appelle cela l'incrément.

Les fonctions

➤ Une **fonction** est une **série d'instructions** qui effectue des actions et qui **retourne une valeur**.

```
1 <?php
2
3 function isAllowed( $email )
4 {
5     if ( $email == 'z.oufkir@emsi.ma' ) {
6         return true ;
7     } else {
8         return false ;
9     }
10
11 }
```

Les fonctions

➤ Pour créer une fonction:

1. Vous devez taper **function** (en français, ça veut dire « fonction »).
2. Ensuite, donnez-lui **un nom**. Par exemple, celle-ci s'appelle **isAllowed** .
3. Nous avons mis entre parenthèses une variable **\$email**. C'est le **paramètre** dont a besoin la fonction pour travailler.
4. Ensuite, vous repérez **des accolades**. Elles permettent de marquer les limites de la fonction. Cette dernière commence dès qu'il y a une **{** et se termine lorsqu'il y a une **}** .
Entre les deux, il y a son contenu.

Les fonctions

➤ Pour appeler une fonction:

1. On écrit le nom de la fonction à appeler.
2. On donne en **entrée** à la fonction un **paramètre** sur lequel elle va faire le test.
3. Et la fonction nous retourne en **sortie** le résultat : **false** .

```
1 <?php
2
3 $email = "test@emsi.ma"
4
5 isAllowed($email); // retourne le booléen false
```

Les fonctions

- PHP propose des centaines de **fonctions prêtes à l'emploi**. Sur le site officiel, la documentation PHP les répertorie toutes, [classées par catégorie](#).
- Voici un petit aperçu de ce que peuvent vous permettre de faire des fonctions PHP :
 - **array_key_exists** pour vérifier si une clé existe dans le tableau.
 - **in_array** pour vérifier si une valeur existe dans le tableau
 - **array_search** pour récupérer la clé d'une valeur dans le tableau.
 - **str_replace** pour rechercher et remplacer des mots dans une variable.
 - **strlen** pour calculer la longueur d'une chaîne de caractères ;

Les fonctions

➤ Exemple:

```
1 <?php
2 $recipe = 'Etape 1 : des flageolets ! Etape 2 : de la saucisse toulousaine';
3 $length = strlen($recipe);
4
5
6 echo 'La phrase ci-dessous comporte ' . $length . ' caractères : ' . '<br>' . $recipe;
```

La phrase ci-dessous comporte 63 caractères :
Etape 1 : des flageolets ! Etape 2 : de la saucisse toulousaine

Les fonctions

➤ Exemple:

```
1 <?php
2 $users = [
3     'Mathieu Nebra',
4     'Mickaël Andrieu',
5     'Laurène Castor',
6 ];
7
8 if (in_array('Mathieu Nebra', $users))
9 {
10     echo 'Mathieu fait bien partie des utilisateurs enregistrés !';
11 }
12
13 if (in_array('Arlette Chabot', $users))
14 {
15     echo 'Arlette fait bien partie des utilisateurs enregistrés !';
16 }
```

Mathieu fait bien partie des utilisateurs enregistrés !

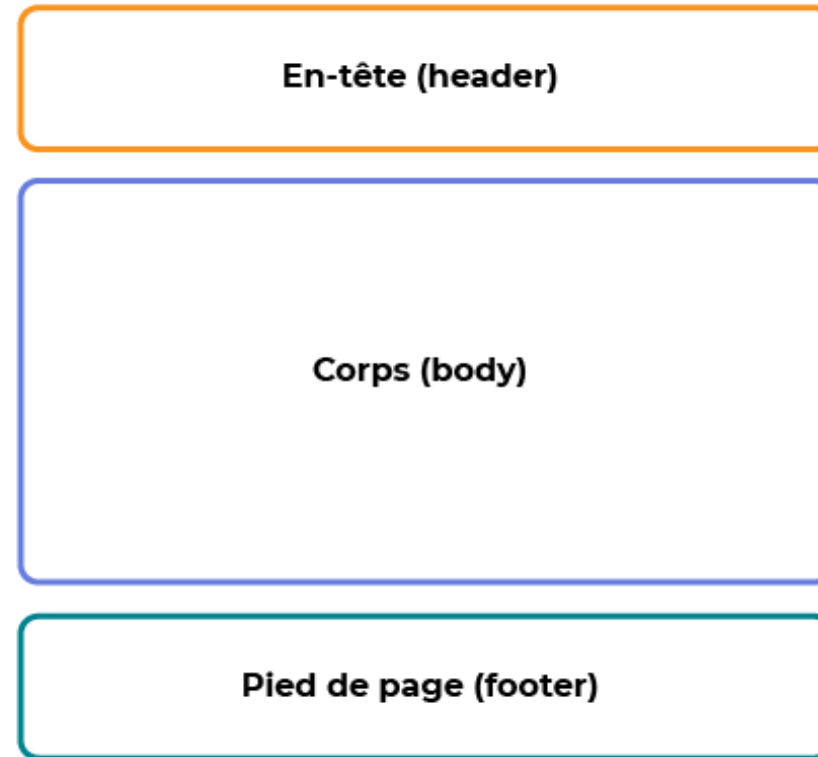
Les fonctions

- Les variables déclarées **dans une fonction** sont **locales** à cette fonction.
- Les variables déclarées à **l'extérieur d'une fonction** sont **globales** pour cette fonction.

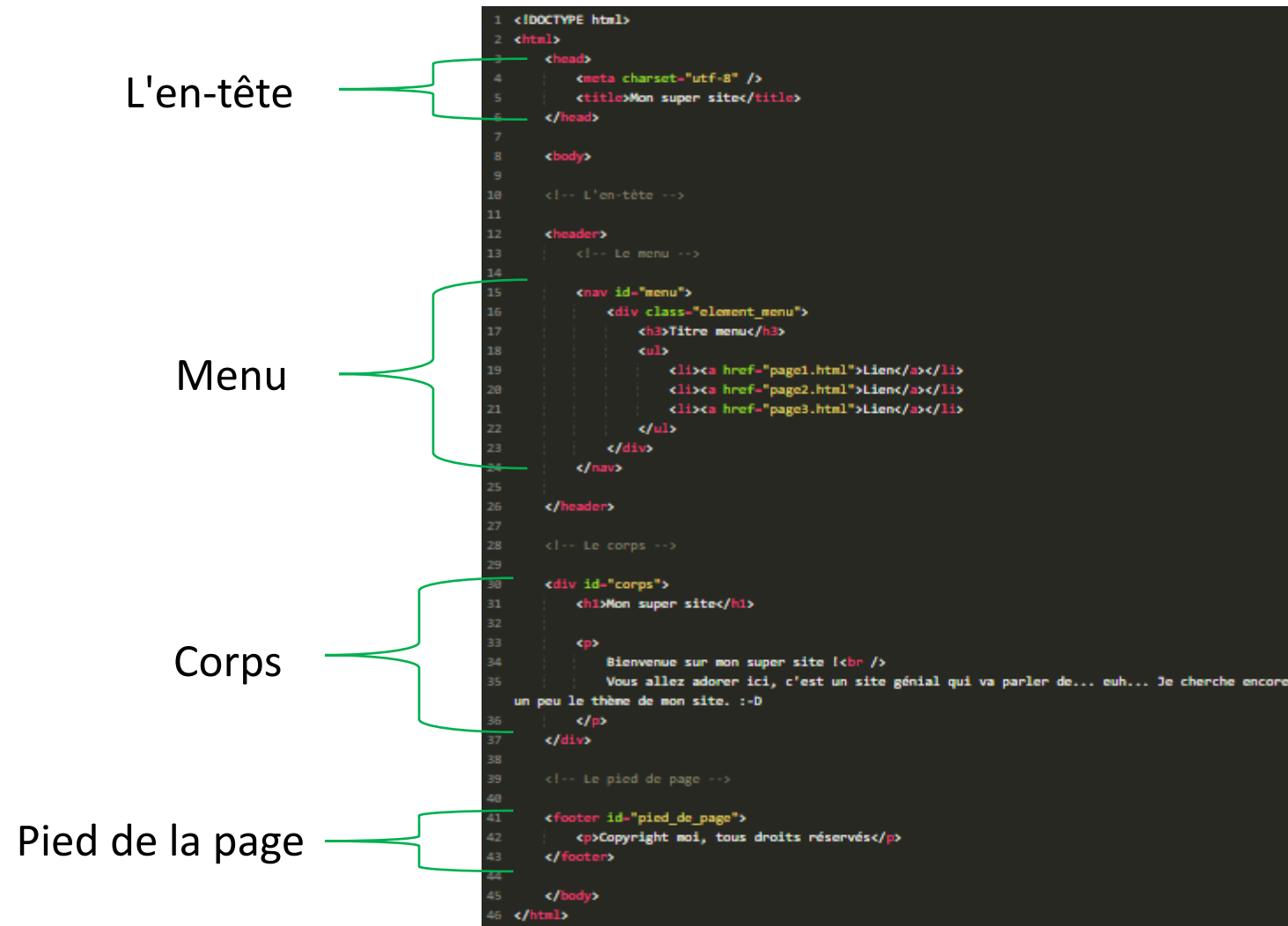
```
1 <?php
2
3 $school = 'EMSI'; // variable global
4 function studies()
5 {
6     global $school;
7
8     $classe = '3IIR'; // variable local
9     echo "$school $classe";
10 }
```

Concept de modularisation

➤ La plupart des sites web sont généralement découpés selon le schéma suivant:



Concept de modularisation



Concept de modularisation

➤ L'en-tête :

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <meta charset="utf-8" />
5         <title>Mon super site</title>
6     </head>
7
8     <body>
9
10    <!-- L'en-tête -->
```

Concept de modularisation

➤ Menu:

```
11
12     <header>
13         <!-- Le menu -->
14
15         <nav id="menu">
16             <div class="element_menu">
17                 <h3>Titre menu</h3>
18                 <ul>
19                     <li><a href="page1.html">Lien</a></li>
20                     <li><a href="page2.html">Lien</a></li>
21                     <li><a href="page3.html">Lien</a></li>
22                 </ul>
23             </div>
24         </nav>
25
26     </header>
```

Concept de modularisation

➤ Corps:

```
27
28     <!-- Le corps -->
29
30     <div id="corps">
31         <h1>Mon super site</h1>
32
33         <p>
34             Bienvenue sur mon super site !<br />
35             Vous allez adorer ici, c'est un site génial qui va parler de... euh... Je cherche
encore un peu le thème de mon site. :-D
36         </p>
37     </div>
```

Concept de modularisation

➤ Pied de la page:

```
38
39     <!-- Le pied de page -->
40
41     <footer id="pied_de_page">
42         <p>Copyright moi, tous droits réservés</p>
43     </footer>
44
45 </body>
46 </html>
```

Concept de modularisation

- D'une page à l'autre, ce site contiendra à chaque fois **le même code pour l'en-tête, le menu et le pied de page !** En effet, seul le contenu du **corps** change, en temps normal.
- En **PHP**, on peut **insérer des morceaux de pages à l'intérieur d'une autre page à l'aide de include.**
- Le principe de fonctionnement des **inclusions** en PHP est plutôt simple à comprendre. Vous avez un site web composé de vingt pages. Sur chaque page, il y a un menu, toujours le même. Pourquoi ne pas écrire ce menu une seule fois dans une page **header.php** ?

Concept de modularisation

- Créons un **nouveau fichier PHP** appelé **headers.php** : nous allons insérer **uniquement le code HTML** correspondant au **menu** :

```
1 <nav id="menu">
2     <div class="element_menu">
3         <h3>Titre menu</h3>
4         <ul>
5             <li><a href="page1.html">Lien</a></li>
6             <li><a href="page2.html">Lien</a></li>
7             <li><a href="page3.html">Lien</a></li>
8         </ul>
9     </div>
10 </nav>
```

- Maintenant que vos « morceaux de pages » sont prêts :

- Reprenons la page d'accueil nommée **index.php**

- Remplacez le menu par le code PHP suivant :

```
1 <?php include('header.php'); ?>
```

Concept de modularisation

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8" />
5     <title>Mon super site</title>
6   </head>
7
8   <body>
9
10    <?php include('header.php'); ?>
11
12    <!-- Le corps -->
13
14    <div id="corps">
15      <h1>Mon super site</h1>
16
17      <p>
18        Bienvenue sur mon super site !<br />
19        Vous allez adorer ici, c'est un site génial qui va parler de... euh... Je cherche encore
20        un peu le thème de mon site. :-D
21      </p>
22    </div>
```

Concept de modularisation

➤ On va faire la même chose avec le **pied de page**:

```
10  <?php include('header.php'); ?>
11
12  <!-- Le corps -->
13
14  <div id="corps">
15      <h1>Mon super site</h1>
16
17      <p>
18          Bienvenue sur mon super site !<br />
19          Vous allez adorer ici, c'est un site génial qui va parler de... euh... Je cherche
encore un peu le thème de mon site. :-D
20      </p>
21  </div>
22
23  <!-- Le pied de page -->
24
25  <?php include('footer.php'); ?>
```

Concept de modularisation

- L'exemple suppose que notre page s'appelle `index.php` et celles qui sont incluses (comme `header.php`) sont dans le même dossier.
- Si le menu était dans un sous-dossier appelé `includes`, il aurait fallu écrire :

```
1 <?php include('includes/header.php'); ?>
```

- La page finale que reçoit le visiteur est identique, vous avez énormément gagné en flexibilité, puisque notre code n'est plus recopié plusieurs fois inutilement.

Concept de modularisation

- `include()` et `require()` ont un fonctionnement strictement le même mais la différence qui les sépare réside dans la gestion des erreurs:
 - La fonction `include()` renverra une erreur de type `WARNING` si elle n'arrive pas à ouvrir le fichier en question. De ce fait l'exécution du code qui suit dans la page sera exécuté.
 - la fonction `require()` affichera une erreur de type `FATAL` qui interrompt l'exécution du script.

```
1 <?php require('includes/header.php'); ?>
```