LES BASES DE PHP

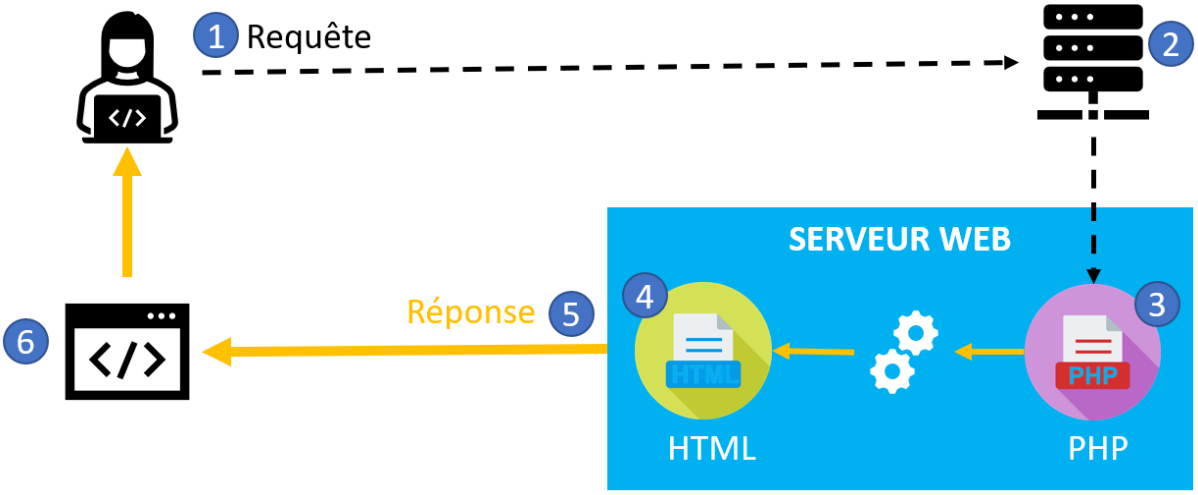
TABLE DES MATIERES

C'EST QUOI, PHP ?

PHP est un **langage de programmation** qui peut être **intégré à un serveur http** (comme Apache ou nginx) **pour générer des pages web**.

Le code php est **exécuté sur le serveur**, **générant ainsi le HTML** (ou autre), qui sera ensuite envoyé au client. Le client ne reçoit que **le résultat** du script, sans aucun moyen d'avoir accès au code qui a produit ce résultat.

Un script php sera généralement stocké dans un fichier avec une **extension .php**. C'est comme ça que le serveur sait qu'il doit exécuter du code avant de renvoyer la page au navigateur.



L'illustration ci-dessus montrer un flux classique dans l'utilisation de PHP pour générer du HTML :

1) Le client envoie en requête au serveur
2) Le serveur détermine qu'il s'agit d'un script PHP
3) Le serveur web exécute le PHP pour générer le HTML en fonction de la demande de l'utilisateur
4) et 5) Le HTML est envoyé au client qui a effectué la requête
6) Le client peut afficher le HTML qu'il a reçu.

Une fois que le HTML a été envoyé, il n'y bien sûr plus moyen de le modifier.

COMMENT FONCTIONNE PHP?

Dans un script PHP classique, on pourra retrouver à la fois **du html et du php** : dans un même script, il y a ainsi **2 modes[1] : un mode html** et un **mode php.**

Dans le **mode html**, le contenu sera envoyé **tel quel** au navigateur. Il doit contenir du **html valide** et prêt à être affiché. Dans ces parties de la page, on mettra le HTML que l'on peut considérer comme **statique**, c'est-à-dire qui sera commun à tous les affichages de la page ou, autrement dit, qui ne dépend pas de l'utilisateur qui a demandé la page.

Dans le **mode php**, le contenu sera **interprété et exécuté** par php avant d'être renvoyé au navigateur. Dans ce contexte, on ne peut retrouver **que du code php valide**. Pour « activer » le mode php, il faut entourer le code par les **balises** de php (voir paragraphe suivant). On appelle aussi ces 2 balises spéciales des **directives de traitement**.

Ce code PHP pourra effectuer des actions sur le serveur, comme **générer du code html** spécifique pour l'utilisateur, bien sûr, mais aussi **valider les données** d'un formulaire et les **enregistrer dans des fichiers** ou dans des **bases de données**, par exemple.

Exemple :

Dans ce fichier, la partie jaune fait partie du mode html (et ne contient que du html), tandis que la partie bleue fait partie du mode php (et ne contient que du code php).

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <title>Cours PHP & MySQL</title>
5.          <meta charset="utf-8">
6.          <link rel="stylesheet" href="cours.css">
7.      </head>
8.
9.      <body>
10.         <h1>Titre principal</h1>
11.         <?php
12.           $prenom = "Pierre";
13.           $age = 28;
14.
15.           echo $prenom;
16.         ?>
17.
18.         <p>Un paragraphe</p>
19.     </body>
20. </html>
```

[1] On dit aussi « contexte »

Voici ce que le client (le navigateur) recevra de son côté :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Cours PHP & MySQL</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="cours.css">
</head>
<body>
    <h1>Titre principal</h1>
    Pierre

    <p>Un paraggraphe</p>
</body>
</html>
```

On peut le voir, tout **le code PHP a disparu**. Plus exactement, il a **été interprété et exécuté par le serveur**, et seul le résultat final est transféré. Dans cet exemple en PHP, la ligne 15 affiche la valeur de la variable `$prenom` dans le HTML (la méthode `echo` permet d'afficher du texte).

LES BALISES PHP

Lorsque PHP traite un fichier, il cherche **toutes les directives** d'ouverture et de fermeture **<?php** et **?>** qui **délimitent le code** qu'il doit interpréter.

Exemple :

```
<h1>Titre de la page</h1>
<?php
    echo "Lorem ipsum dolor sit.";
?>
```

Toutes les portions de code écrites en PHP seront donc incluses entre les directives `<?php` et `?>`.

Dans un même fichier, il peut bien sûr y avoir un mélange de code HTML et de PHP. On peut retrouver une multitude de portions de code PHP à différents endroits dans une même page :

```
<h1>Titre de la page</h1>
<?php
    echo "Lorem ipsum dolor sit.";
```

```
?>
<p>Reiciendis velit dicta ullam eligendi dolorem.</p>

<h2>Sous-titre</h2>
<?php
      $ageMois = 45 * 12;
?>
<p>Le capitaine est né il y a <?= $ageMois ?> mois.</p>
```

Dans le code ci-dessus, on peut voir qu'il a 3 endroits écrits en PHP.

QUE PEUT FAIRE LE PHP ?

PHP est utilisable sur la **majorité des systèmes d'exploitation**, comme **Linux**, de nombreuses variantes Unix, Microsoft **Windows**, macOS, RISC OS et d'autres encore. PHP supporte aussi **la plupart des serveurs web** actuels dont **Apache**, **IIS** et bien d'autres.

Avec PHP vous avez le choix de votre système d'exploitation et de votre serveur web.

Avec PHP, vous n'êtes pas limité à la **production de code HTML**. Les capacités de PHP lui permettent de générer aussi bien des **images**, des fichiers **PDF** à la volée. Vous pouvez aussi générer facilement du texte, du code XML ou XHTML. PHP génère tous ces fichiers et les sauve dans le système de fichiers, ou bien les envoie directement au navigateur web.

Une des forces les plus significatives de PHP est qu'il **supporte énormément de bases de données**. Écrire une page web faisant appel à une base de données devient simple, **en utilisant une des extensions spécifiques** aux bases de données (i.e., pour mysql), ou utilisant une classe d'abstraction comme **PDO**, ou une connexion à n'importe quelle base de données supportant une connexion standard.

COMMENT GÉNÉRER DU CONTENU ?

Comme on l'a vu plus haut, le contenu peut être généré, de base, de 2 façons différentes.

Soit en **écrivant le HTML directement**, auquel cas il sera envoyé tel quel au navigateur, soit en générant le HTML **à l'aide d'instructions en PHP**.

Vous l'avez sans doute lu dans les paragraphes précédents, la 1ère commande PHP qui permet d'écrire du contenu est la commande `echo`.

Bien sûr, le PHP peut aussi générer du HTML (pas seulement du texte). La commande `echo` peut donc aussi contenir des balises :

```
<h2>Sous-titre</h2>
<?php
    $prixTotal = 100;
    $prixFinal = 92;
    $reduction = $prixTotal - $prixFinal;
    echo "<h2>Votre réduction: {$reduction} €</h2>";
?>
```

Si vous ne comprenez pas tout le code ci-dessus, c'est normal. Retenez simplement qu'on peut générer du contenu HTML avec la commande `echo` (ou toute autre commande qui permet d'afficher quelque chose).

LA SYNTAXE DE BASE

Vous avez déjà lu plus haut que le code php doit être entouré par les balises (ou directives de traitement) `<?php … ?>` ou `<?= … ?>`.

En guise d'introduction, sachez qu'une instruction en PHP **doit** toujours **se terminer par un `;` (point-virgule)**.

Il y a une exception notable que vous rencontrerez souvent où omettre le point-virgule pourra augmenter la lisibilité de votre code : quand vous utilisez la version courte de la commande `echo` :

```
<?= $ageMois ?>
```

Dans ce cas, on peut se passer du point-virgule.

LES COMMENTAIRES

Les **commentaires** sont des portions de code qui **seront** tout simplement **ignorées** par PHP. Vous pouvez y mettre ce que vous voulez. A quoi bon, puisqu'elles sont ignorées ?

Sans entrer dans les détails, les commentaires permettent au développeur d'**expliquer ce qu'il fait** et/ou de **désactiver** temporairement une partie de son code.

Pour commenter une seule ligne de code (ou une portion de ligne), utilisez le **double-slash** : `//`. Tout ce qui se trouve entre le `//` et la fin de la ligne sera ignoré.

```php
<?php
    // Cette ligne est ignorée
    $prixTotal = 100;
    $prixFinal = 92; // ce commentaire est ignoré
?>
```

La ligne 2 est complétement ignorée. Tout ce qui se trouve avant le `//` à la ligne 4 est exécuté, tandis que tout ce qui se trouve après est ignoré.

Note : tout comme en Linux, le caractère `#` (hashtag, dièse…) peut être utilisé à la place du `//` :

```php
<?php
    // Cette ligne est ignorée
    # Cette ligne est ignorée
?>
```

Pour commenter tout un bloc de code, utilisez la combinaison de `/*` et `*/`. Tout ce que se trouve entre sera ignoré.

```php
<?php
    /*
        Tout ce qui se trouve dans ce bloc est
        ignoré par PHP.
        $prix = 152;
        echo "Test";
    */
?>
```

LES VARIABLES

En PHP, les variables sont représentées par un signe dollar "$" suivi du nom de la variable. Le nom est sensible à la casse.

Une variable est une structure de données de **type primitif** (entier, réel, caractère, chaîne de caractères, booléen ou null) ou bien de **type structuré** (**tableau** ou **objet**) qui permet de **stocker temporairement** une ou plusieurs valeurs. La valeur d'une variable est susceptible d'être remplacée par une autre valeur au cours de l'exécution du programme. D'où le terme « variable ».

À un moment donné, une variable a un **nom**, un **type** (qui définit le type de contenu) et une **valeur**. La valeur et le type peuvent donc changer (voir plus loin).

Pour rappel, la **déclaration** d'une variable permet de lui donner un **nom** et un **type**, alors que l'**affectation** permet de lui donner une **valeur**.

LE NOM

Un nom de variable valide doit commencer par $ suivi d'une lettre ou d'un souligné (_), suivi de lettres, chiffres ou soulignés.

Exemples : `$prenom`, `$_test`, `$zone51`

Les noms de variables peuvent aussi **contenir des chiffres** (sauf au début du nom).

Les noms de variables sont sensibles à la casse, ce qui signifie que l'interpréteur fera **la différence entre les majuscules et les minuscules**. Par exemple, `$nom` et `$Nom` seront considérées comme deux variables complètement distinctes :

```php
<?php
    $nom = "Jean";
    $Nom = "Paul";
    echo $nom;
    echo $Nom;
?>
```

Quel résultat produit ce code ? Il écrit « JeanPaul ».

Par ailleurs, il est nécessaire de **nommer les variables avec des noms évocateurs** afin de faciliter la lecture du code. Une variable mal nommée sera source de problème lors d'un débugging ou d'une maintenance du programme des semaines plus tard.

```php
<?php
  $prenom = 'Hugo';      // Type string (chaine de caractères)
  $nom = "Hamon";        // Type string (chaine de caractères)
  $age = 19;             // Type entier
  $estEtudiant = true;   // Type booléen
  $cours = array('physique', 'informatique','philosophie'); // Type tableau
  $unEtudiant = new Etudiant(); // Objet de type Etudiant
?>
```

Par convention, les noms de variables composés de plusieurs mots (exemple : `$estEtudiant`) doivent avoir **le premier mot en minuscules** et les autres mots avec la première lettre en majuscule. Cette règle est fortement conseillée dans la mesure où elle fait partie des bonnes pratiques de programmation. En anglais, on appelle cette façon d'écrire « **lower camel case** ».

LE TYPE

On dit que PHP est un **langage de typage « faible et dynamique »**.

En effet, contrairement à des langages très typés comme C, C++ ou Java,

- **le type d'une variable est défini par son contenu au moment de l'affectation**
- **le type d'une variable peut changer en cours d'exécution**

Par exemple, pour une même variable, le programmeur est libre de lui affecter une valeur de type entier à un moment puis de lui affecter une chaine de caractères plus tard.

Cela rend l'utilisation de PHP plus souple par les développeurs mais pas forcément plus propre... Cela requière une plus grande discipline.

Autrement dit, **le type d'une variable n'est pas déclaré explicitement** comme en Java, C ou C++ **mais implicitement au moment de l'affectation d'une valeur**.

De plus, et même si **ce n'est pas une bonne pratique**, **le type** d'une variable **peut changer** si on lui affecte une nouvelle valeur avec un nouveau type :

```php
<?php
  $age = 19;            // Type entier
  $age = "20 ans"       // Type string
?>
```

Pour déclarer une variable de type :

- **string** : on entoure la chaîne de caractères de **guillemets** ou **d'apostrophes**.
- **entier**, **réel** ou **flottant** : on écrit la valeur telle quelle. Pour les flottants, la virgule est remplacée par un point (écriture à l'américaine).
- **booléen** : on écrit `true` ou `false` directement.
- **sans type** : si l'on ne souhaite pas typer la variable, on lui affecte la valeur **null**.

OPÉRATIONS DE BASE

Vous allez découvrir ici les opérations de bases qui vous permettront de débuter rapidement en PHP, et qu'en plus vous utiliserez tout le temps et à toutes les sauces.

ÉCRIRE/GÉNÉRER DU CONTENU

Écrire du contenu dans la page depuis PHP se fait le plus souvent avec la fonction[2] `echo`, que l'on a déjà évoquée.

Cette fonction reçoit en paramètre le contenu à écrire et peut s'utiliser avec ou sans parenthèses. D'ailleurs, le plus souvent, vous la verrez utilisé sans :

```
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple params.";
```

Si vous avez beaucoup de HTML à générer, alors il est généralement préférable de « quitter le mode PHP » et de passer en mode HTML, c'est-à-dire d'écrire votre HTML en dehors des balises `<?php` et `?>`.

OPÉRATIONS ARITHMÉTIQUES

LES CHAÎNES DE CARACTÈRES

[2] En fait, ce n'est pas vraiment une fonction, mais ce n'est pas vraiment important de le savoir 😊

LES TABLEAUX

Un tableau permet de stocker plusieurs valeurs dans une même variable.

Il existe 3 types de tableaux en PHP :

- Les **tableaux indexés** : avec un index numérique
- Les **tableaux associatifs** : avec des paires clé-valeur
- Les **tableaux multi-dimensionnels** : des tableaux qui contiennent d'autres tableau

Ces tableaux peuvent être déclarés de 2 façons différentes. Soit sous la forme « `$tableau = array(….) ;` », soit sous la forme courte « `$tableau = […] ;` ». Pendant la déclaration, les différents éléments du tableau doivent être **séparés par une virgule**.

LES TABLEAUX INDEXÉS

Les tableaux indexés utilisent **un index entier** implicite pour manipuler ses éléments :

```php
<?php
    $cars = array("Volvo", "BMW", "Toyota");
?>
```

Ou, avec la notation courte :

```php
<?php
    $cars = ["Volvo", "BMW", "Toyota"];
?>
```

Dans les deux cas, **les valeurs sont accessibles grâce à leur index numérique**. La 1ère valeur est à la position `0`, la 2e à la position `1`, et ainsi de suite…

Le 1er élément est à la position `0`. On dit que le tableau est « *zero based* »

On accède aux éléments du tableau de la façon suivante :

```php
<?php
    echo $cars[0];
    echo $cars[1];
    echo $cars[2];
?>
```

PARCOURIR LES ÉLÉMENTS D'UN TABLEAU INDEXÉ

Pour parcourir les éléments d'un tableau, il faut utiliser une boucle `for`, `while` ou `foreach`.

BOUCLE `FOR`

```php
<?php
    $tableau = ["a","b","c","d","e","f","g"];
    $taille = count($tableau);

    for ($i = 0; i < taille; $i++) {
        echo $tableau[$i] . "<br/>";
    }
?>
```

BOUCLE `WHILE`

```php
<?php
    $tableau = ["a","b","c","d","e","f","g"];
    $taille = count($tableau);
    $i = 0;

    while ($i < $taille) {
        echo $tableau[$i] . "<br/>";
        $i++;
    }
?>
```

Dans le cas des boucles `for` et `while`, il faut connaître la taille du tableau. Pour cela, utiliser la fonction « `count($tableau)` » ou son équivalent « `sizeof($tableau)` ».

BOUCLE `FOREACH`

```php
<?php
    $tableau = ["a","b","c","d","e","f","g"];

    foreach ($tableau as $element) {
        echo $element . "<br/>";
    }
?>
```

Dans le cas d'une boucle `foreach`, vous pouvez aussi **avoir accès à l'index** :

```php
<?php
    $tableau = ["a","b","c","d","e","f","g"];
```

```php
    foreach ($tableau as $cle => $valeur) {
        echo $cle . " " . $valeur . "<br/>";
    }
?>
```

Cela peut paraître surprenant mais souvenez-vous : ces tableaux sont indexés à l'aide d'un index numérique, qui n'est rien d'autre qu'une **série de clés** numériques…

AJOUTER UN ÉLÉMENT À LA FIN

Après avoir déclaré un tableau indexé, il est toujours possible d'y ajouter des éléments.

La syntaxe est la suivante :

```php
<?php
    $cars = ["Volvo", "BMW", "Toyota"];

    $cars[] = "Peugeot";
    $cars[] = "Mercedes";
?>
```

`$cars[]`, sans spécifier d'index, permet d'**ajouter un élément à la fin du tableau**.

LES TABLEAUX ASSOCIATIFS

Les tableaux associatifs utilisent une clé au format *string* (chaine de caractères) pour pouvoir accéder à ses éléments. On dit qu'ils utilisent des **paires clé-valeur**.

Dans ce cas-ci, contrairement aux tableaux indexés, **les clés sont explicitement définies au moment de la déclaration**. Il existe **3 façons** équivalentes de déclarer les tableaux associatifs :

```php
<?php
    $age = array(
        "Peter" => "35",
        "Ben" => "37",
        "Joe" => "43"
    );
?>
```

```php
<?php
    $age = [
        "Peter" => "35",
        "Ben" => "37",
        "Joe" => "43"
    ];
?>

<?php
    $age["Peter"] = "35";
    $age["Ben"] = "37";
    $age["Joe"] = "43";
?>
```

Les clés doivent être uniques. Si vous utilisez deux fois la même clé, l'élément qui utilisait déjà cette clé sera écrasé par le nouveau.

Pour **accéder à un élément spécifique** du tableau, on utilise sa clé :

```php
<?php
    echo $age["Peter"];
    echo $age["Ben"];
    echo $age["Joe"];
?>
```

PARCOURIR UN TABLEAU ASSOCIATIF

Pour **parcourir un tableau associatif**, vous pouvez utiliser une boucle `foreach` ;

```php
<?php
    $tableau = [
        "Peter" => "35",
        "Ben" => "37",
        "Joe" => "43"
    ];

    foreach ($tableau as $cle => $valeur) {
        echo $cle . " " . $valeur . "<br/>";
    }
?>
```

LES FONCTIONS

C'est quoi ?

A quoi ça sert ?

Comment les utiliser ?

Paramètre / arguments

Passage par valeur / par référence

Bonnes pratiques

PHP OU HTML ?

Une question qui va se poser régulièrement, c'est **« Comment je génère mon contenu dans ce cas précis ? J'écris directement en HTML ou je passe par le PHP ? »**

La réponse simple est : **s'il y a plus d'HTML statique que de PHP, alors écrivez du HTML** dans lequel vous viendrez insérer des `<?= … ?>` pour insérer votre contenu dynamique. Votre code sera plus lisible.

Prenez l'exemple suivant :

```
<div class="item-info">
    <div class="item-overlay bottom text-right">
        <a href="#" class="btn-favorite"><i class="fa fa-heart-o"></i></a>
        <a href="#" class="btn-more" data-toggle="dropdown">
             <i class="fa fa-ellipsis-h"></i>
         </a>
        <div class="dropdown-menu pull-right black lt"></div>
    </div>
    <div class="item-title text-ellipsis">
        <a href="track.detail.html">Blind Me</a>
    </div>
    <div class="item-author text-sm text-ellipsis">
        <a href="artist.detail.html" class="text-muted">Fifty</a>
    </div>
</div>
```

Dans cet exemple, il y a **beaucoup de HTML**, dont la plupart est **statique** (ne dépend pas de la valeur d'une ou plusieurs variables). On remarque que seules les lignes surlignées ont un contenu qui est susceptible de changer en fonction d'une ou plusieurs variables. Ce contenu pourra dès lors être généré via du code PHP, comme suit :

```
<div class="item-info">
    <div class="item-overlay bottom text-right">
        <a href="#" class="btn-favorite"><i class="fa fa-heart-o"></i></a>
        <a href="#" class="btn-more" data-toggle="dropdown">
            <i class="fa fa-ellipsis-h"></i>
         </a>
        <div class="dropdown-menu pull-right black lt"></div>
    </div>
    <div class="item-title text-ellipsis">
        <a href="track.detail.html"><?= $trackTitle ?></a>
    </div>
    <div class="item-author text-sm text-ellipsis">
        <a href="artist.detail.html" class="text-muted"><?= $artistName ?></a>
    </div>
</div>
```

Générer de grandes quantités de HTML à l'aide de la fonction `echo` peut aussi devenir très vite **fastidieux** à cause des nombreux guillemets qu'il faudra **échapper**.

[Exemples]

Pour vous aider à répondre à cette question, il peut être utile de comprendre **comment PHP travaille avec le HTML.** Pour cela, vous pouvez vous imaginer que **chaque ligne de votre contenu HTML est transformée par le moteur php en une série de `echo`s successifs lors de l'interprétation de votre fichier.**

Par exemple, imaginez le html suivant :

```
<h1>Avengers</h1>
<h2>Age of Ultron</h2>
<h3>Introduction</h3>
```

Vous pouvez considérer qu'en interne php va automatiquement transformer ces lignes en :

```
<?php
```

```
        echo "<h1>Avengers</h1>";
        echo "<h2>Age of Ultron</h2>";
        echo "<h3>Introduction</h3>";
    ?>
```

Chaque ligne de HTML est « transformée » en son équivalent avec `echo`.

C'est important de l'imaginer comme ça car ça devrait vous aider à comprendre que **le HTML fait partie intégrante de vos programmes**. Vous pouvez générer des portions entières de html de façon conditionnelle (dans un `if…else`) et/ou répétitive (dans des boucles).

Prenez l'exemple suivant :

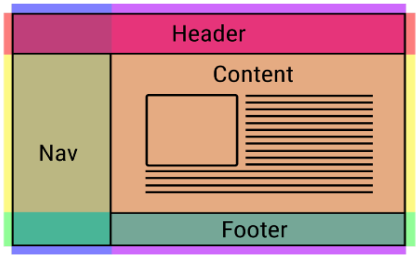
```
    <?php
        if ($age < 18) {
    ?>
        <h1>Avertissement</h1>
        <p>Vous ne pouvez pas accéder à ce contenu.</p>
    <?php
        }
    ?>
```

Ce code ne génère le titre et le paragraphe que si la variable `$age` est < à 18 et peut être imaginé comme ceci :

```
    <?php
        if ($age < 18) {
            echo "<h1>Avertissement</h1>";
            echo "<p>Vous ne pouvez pas accéder à ce contenu.</p>";
        }
    ?>
```

DIVISER POUR MIEUX RÉGNER

Vous allez très vite remarquer que **de nombreux éléments de vos pages sont répétés** dans plusieurs pages, parfois de nombreuses fois. C'est souvent le cas pour les **menus** de navigation, l'**entête** et le **pied de page**…



Au lieu de copier-coller ce contenu à chaque fois, il est plus facile de l'**extraire dans un fichier PHP à part entière** et d'**inclure ce fichier partout où vous en avez besoin**. **Le résultat généré sera le même**, votre code deviendra **plus lisible** et, surtout, deviendra beaucoup **plus maintenable**.

En effet, une fois votre contenu centralisé dans un fichier séparé, vous ne devez **modifier qu'un seul fichier** à chaque fois que vous devez apporter des modifications. Ces modifications seront automatiquement incorporées partout où le fichier a été inclus.

Ce qui est génial, c'est que cela fonctionne aussi bien pour du contenu que pour du code PHP. En effet, vous pouvez aussi **inclure des fonctions/méthodes PHP**[3] **pour créer des librairies de code réutilisables**, que vous pourrez simplement inclure dans les fichiers où vous en avez besoin.

COMMENT FAIRE ?

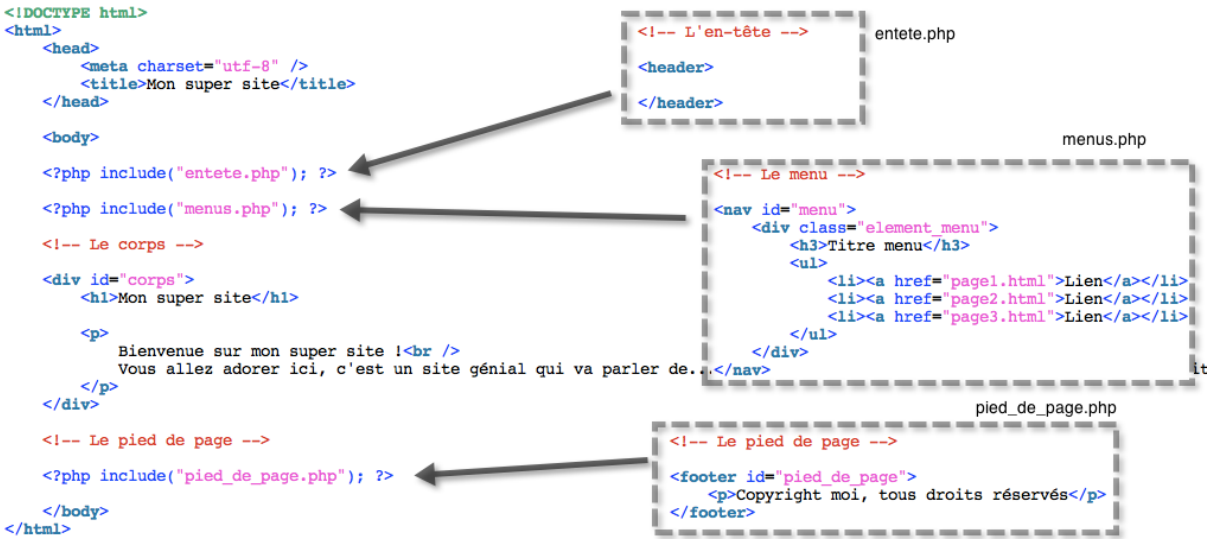
Pour inclure un fichier dans un autre, vous pouvez utiliser `include`.

```php
<?php include "header.php"; ?>
```

Vous donnez le nom du fichier à inclure. Le chemin du fichier est spécifié de manière relative.

Au moment où PHP rencontre cette méthode, il charge, lit et interprète complètement le contenu du fichier spécifié puis continue à traiter le fichier courant. C'est comme si PHP **copiait** le contenu du fichier spécifié pour le coller lui-même dans le fichier courant.

[3] Nous verrons les méthodes et fonctions par ailleurs. Ce sont des bouts de codes réutilisables



Il est aussi possible d'utiliser `require`, au lieu de `include`. La seule différence étant que `require` provoque une erreur si le fichier spécifié n'existe pas (et arrête donc l'exécution du script) alors que `include` ne provoque qu'un avertissement (et donc l'exécution de la page se poursuit).

LE BON CHEMIN

Le chemin que vous passez à `include`/`require` est un chemin relatif. C'est-à-dire qu'il est spécifié par rapport à l'endroit où se trouve le fichier courant (le fichier « hôte »). Si le fichier à inclure se trouve dans le même dossier ou dans un sous-dossier, c'est simple, il suffit de spécifier le chemin relatif :

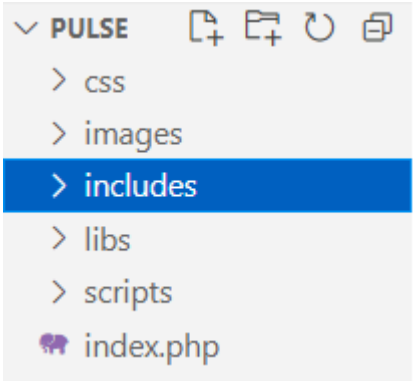
```
include "header.php";
include "composants/carousel.php";
```

C'est un peu plus compliqué lorsque vous devez inclure des fichiers qui se trouvent dans un autre dossier parent, ou qui se trouve au même niveau (frère)

Pour éviter les problèmes quand vous spécifiez le chemin du fichier à inclure, le plus simple est d'utiliser la forme suivante, qui permet de toujours inclure vos fichiers depuis la racine de votre dossier « www » (c'est-à-dire la racine, sur le disque, de votre site) :

```
include $_SERVER['DOCUMENT_ROOT'] . "/includes/header.php";
```

Prenons l'exemple suivant :



Votre site possède un dossier « `includes` » dans lequel vous avez mis vos fichiers à inclure.

Une pratique simple

UTILISATION AVANCÉE

Enfin, les fichiers à inclure qui définissent des variables et des fonctions, et qui sont susceptibles d'être **inclus plusieurs fois** dans une même page, devraient être inclus en utilisant les variantes `include_once` et `require_once` afin d'éviter des erreurs dues à la redéfinition. Ces variantes copieront le contenu du fichier spécifié lors du premier appel, et ne feront rien lors des appels suivants.

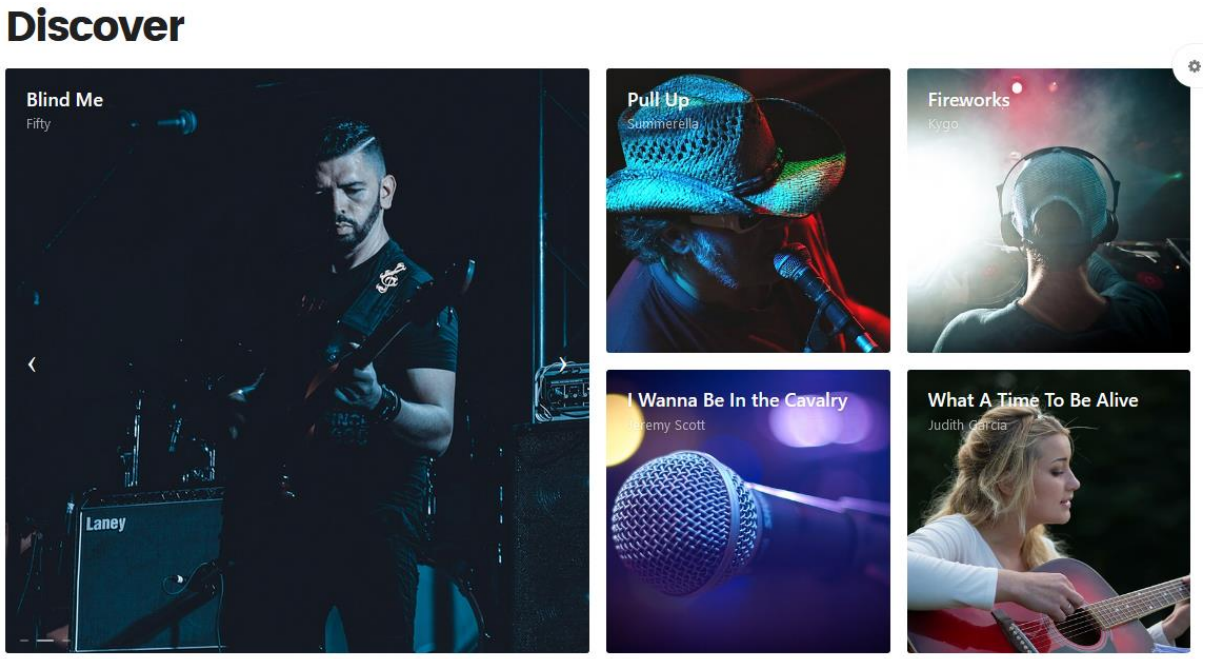
DÉCOUPER SON SITE EN COMPOSANTS

Nous avons vu plus haut qu'inclure du HTML et du PHP avec des `include`s revient à diviser le contenu de vos pages en une forme simple de **composants**.

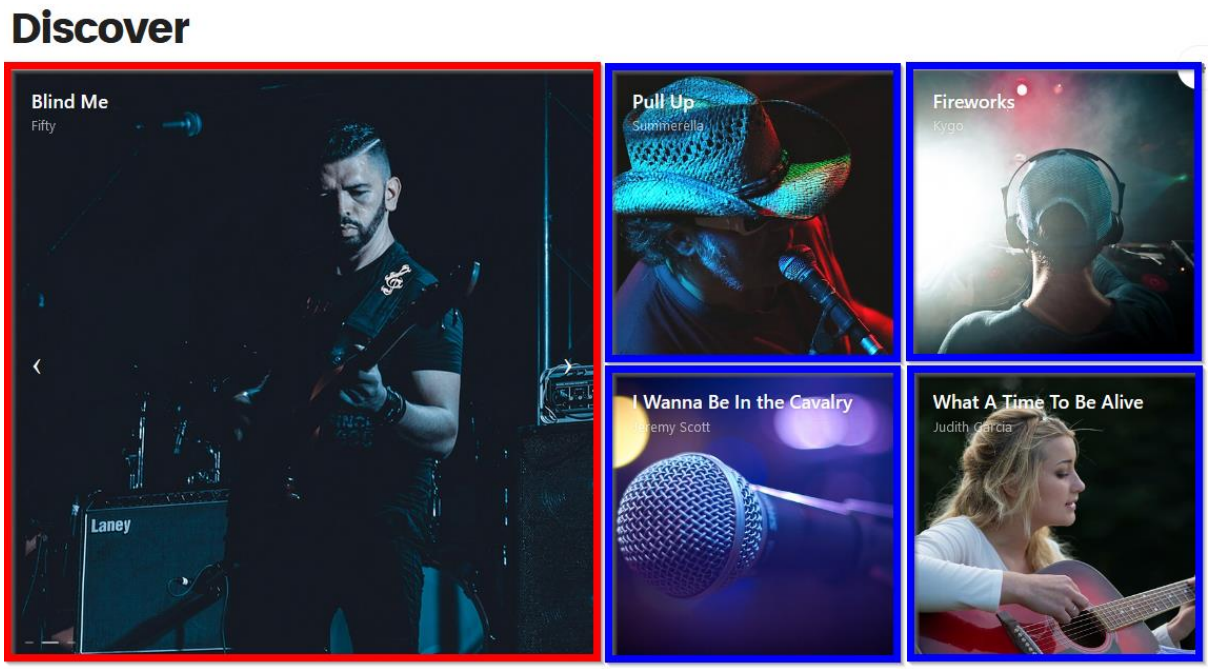
La barre de navigation, l'entête, le pied de page… peuvent tous être considérés comme des composants, que l'on réutilise à volonté.

Là où se devient encore plus intéressant (et où ça se complique un peu), c'est que ces composants peuvent aussi inclure d'autres composants.

Prenons l'exemple de ce site musical :



La grande image est incluse dans un composant de type « carrousel / *slider* » qui fait défiler des pochettes d'albums automatiquement. Les quatre petites images sont simplement des pochettes. On pourrait découper cette page en plusieurs composants :



ARCHITECTURE MVC SIMPLIFIÉE

PASSER DES PARAMÈTRES À UNE PAGE