

PRWB : Projet de Développement Web

Laboratoire 01

Introduction au langage PHP : bases du langage

EPFC-ULB

Benoît Penelle et Boris Verhaegen

Ressources

Les slides suivants constituent une brève introduction aux concepts de base du langage PHP.

Ces slides sont délibérément incomplets et vous êtes invités à consulter la documentation en ligne :

<http://www.w3schools.com/php/> série de tutoriels très exemplifiés

<http://php.net/manual/en/> documentation officielle

<https://openclassrooms.com/> cours en français

...

PHP ?

PHP est un langage de programmation interprété principalement utilisé pour produire des pages Web dynamiques via un serveur Web.

Le code PHP doit se trouver dans un fichier `.php` stocké sur le serveur Web.

Ce fichier peut contenir du code HTML, CSS, JavaScript, ...

Le code PHP y est inclus entre les balises `<?php` et `?>`.

hello.php (<http://localhost/hello.php>)

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



La commande echo produit du texte

Scénario de fonctionnement

Soit le fichier `hello.php` accessible à l'URL <http://localhost/hello.php>.

1. Le client (navigateur) accède à la ressource <http://localhost/hello.php>.
2. Le serveur Web exécute le **code PHP** de la ressource et le transforme en **sa sortie**.
3. Le serveur Web renvoie ce résultat au navigateur qui l'affiche.

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

Fichier stocké sur le serveur

```
<!DOCTYPE html>
<html>
  <head>
    <title>Welcome</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <h1>Welcome</h1>
    <p>
      Hello World
    </p>
  </body>
</html>
```

Document envoyé au client

Syntaxe de base du PHP

Les **commentaires** s'ecrivent comme en C++ (`//` ou `/* */`)

Chaque instruction simple est terminée par un **point-virgule**.

Chaque nom de **variable** est précédée par un `$`.

```

<?php
    $mycounter = 1;           //entier
    $myaverage = 17.5;       //réel
    $mystring  = "Hello";    //string
    $myarray   = array("One", "Two", "Three"); //tableau
?>

```

Attention, au contraire du C++ qui utilise un **typage statique**, on ne déclare pas le type de la variable : il dépend du type de son contenu (**typage dynamique**).

Les chaînes de caractères (*strings*)

Strings littéraux ('...')

```

echo '$variable est un nom de variable';
//produit $variable est un nom de variable

```

Strings avec substitution de variable ("...")

```

$average = 17.5;
echo "Votre moyenne est de $average sur 20";
//produit Votre moyenne est de 17.5 sur 20

```

Caractères d'échappement

```

echo "Il dit \"bonjour\""; //produit Il dit "Bonjour"
echo 'Il dit \'bonjour\''; //produit Il dit 'Bonjour'

```

Concaténation (symbole .)

```

$msg = "Hello " . $name . " !";

```

Chaînes de caractères multi-lignes

Il est parfois nécessaire d'écrire de (très) longues chaînes de caractères et d'en préserver les espaces et retours à la ligne. Pour ce faire, PHP propose les syntaxes suivantes. Celles-ci permettent aussi d'éviter de devoir échapper des caractères spéciaux.

```
<?php
```

```
$nom = "Boris";
```

```
echo <<<END
```

Longue portion de texte
dans laquelle les caractères blancs
sont préservés.

Les variables sont remplacées : \$nom

```
END;
```

```
$message = <<<'END'
```

Même chose mais sans remplacement de variable : \$nom

```
END;
```

Trois '<' suivi d'une balise au choix

Long texte. Ici, les variables
sont remplacées par leur valeur.

Balise fermante obligatoirement
sur un début de ligne !

Même chose sans
remplacement. Remarquez
les apostrophes autour
de la balise.

Les opérateurs et les expressions

La majorité des opérateurs sont indentiques à ceux de C++.

Opérateurs arithmétiques :

+ - * / % ++ --

Opérateurs d'assignation :

= += -= *= /= %= . =

Opérateurs de comparaison :

== != > < >= <=

Opérateurs logiques :

&& and || or ! xor

Le concept d'expression est identique à celui du C++.

Conversions automatiques des types

Comme PHP est un langage faiblement et dynamiquement typé, **PHP peut convertir les variables en fonction du contexte :**

```
<?php
    $x    = "42";
    $y    = 2;
    $res = $x * $y; //conversion automatique de "42" en 42
    echo $res;      //affiche l'entier 84
?>
```

Casting implicite et explicite

PHP convertit les types automatiquement en fonction du contexte. On parle alors de *casting* (ou transtypage) implicite.

Par exemple, le code suivant produit la valeur 2.5 car PHP a converti l'expression de division en un réel pour donner la valeur la plus précise.

```
<?php
    $a = 5;
    $b = 2;
    echo $a / $b . "<br>"; //produit 2.5
?>
```

Si l'on voulait un entier, on aurait du écrire :

```
<?php
    $a = 5;
    $b = 2;
    echo (int)($a / $b) . "<br>"; //produit 2
?>
```

Les autres types de transtypage sont les suivants :

- (int) (integer) pour les entiers*
- (bool) (boolean) pour les booléens*
- (float) (double) (real) pour les réels*
- (string) pour les chaînes*
- (array) pour les tableaux*
- (object) pour les objets*

Egalité et identité

PHP possède un opérateur d'égalité (==) et un opérateur d'identité (===).

```
<?php
    $a = "1000";
    $b = "+1000";
    if ($a == $b) echo "égaux";
    if ($a === $b) echo "identiques";
?>
```

L'opérateur d'égalité permet la conversion automatique des types : ainsi `$a==$b` est vrai car les deux variables sont converties en nombres.

L'opérateur d'identité interdit la conversion des types : `$a=== $b` est faux car ce sont deux *strings* différents.

Les opérateurs `!=` et `!==` sont également définis.

Les booléens

Les valeurs booléennes sont soit TRUE, soit FALSE.

On peut les écrire également en minuscule.

L'affichage d'une expression à valeur TRUE produit 1 tandis qu'une expression à valeur FALSE ne produit rien car la constante FALSE est définie à NULL (rien).

```

<?php
    echo "a: [" . (20 > 9) . "<br>"; //produit a: [1]
    echo "b: [" . (5 == 6) . "<br>"; //produit b: []
?>

```

Les instructions conditionnelles

Les instructions conditionnelles ressemblent très fort à celles du C+.

Petit détail : les mots `else` et `if` peuvent être collés :

```
<?php
if ($a > $b) {
    echo "a is bigger than b";
} elseif ($a == $b) {
    echo "a is equal to b";
} else {
    echo "a is smaller than b";
}
?>
```

Script PHP en plusieurs morceaux

Un script PHP peut être découpé en plusieurs fragments. Cela peut être utile, par exemple, si l'on veut produire de longues parties d'HTML statiques.

```
<?php
    if($password == "1234"){
?>
        <p>Mot de passe correct</p>

<?php
    } else {
?>
        <p>Mot de passe incorrect</p>

<?php
    }
?>
```

```
<?php
    if($password == "1234"){
        echo "<p>Mot de passe correct</p>";
    } else {
        echo "<p>Mot de passe incorrect</p>";
    }
?>
```

Les boucles

Les boucles (while, do while et for) sont similaires à celles de C++.

```

<?php
    $count = 1;
    while($count < 5){
        echo $count."<br>";
        ++$count;
    }

    $count = 1;
    do{
        echo $count."<br>";
        ++$count;
    } while($count < 5);

    for($count = 1; $count < 5; ++$count){
        echo $count."<br>";
    }
?>

```

Les fonctions

Voici un exemple de déclaration et d'utilisation de fonction :

```

<?php
function valeur_absolue($i){
    if($i<0){
        $i = -$i;
    }
    return $i;
}

echo valeur_absolue(-5);
?>

```

Le mot clé `function` permet de déclarer une fonction.

Comme PHP est typé dynamiquement, on ne précise ni le type de retour ni le type des paramètres.

`return` permet de renvoyer une valeur.

La portée des variables

La portée des variables est assez différente par rapport au C++ : une variable existe jusqu'à la fin de son contexte.

```

<?php
    if($a > $b){
        $msg = "ok";
    }
    echo $msg;
?>

```

Si la condition est vérifiée, la variable `$msg` est créée et le programme produit "ok".

Si la condition n'est pas vérifiée, la variable `$msg` n'est pas créée et l'exécution du programme produit une erreur.

Les variables locales

Une variable locale est une variable qui est créée dans une fonction et qui n'est accessible que dans cette fonction.

Les paramètres d'une fonction font également partie des variables locales.

Par défaut, une fonction a accès uniquement à ses variables locales.

Elles disparaissent quand la fonction est terminée.

```

<?php
    $test = "test";
    function essai_portee($a,$b){
        echo $test;           //erreur !
        $somme = $a + $b; //ok : variables locales
        echo $somme;          //ok : variable locale
    }

    essai_portee(5,6);

    echo $somme;              //erreur !
?>
  
```

Les variables globales

Une variable déclarée à l'extérieur d'une fonction est dite globale.

Pour accéder ou créer une variable globale dans une fonction, il faut utiliser le mot clé `global` suivi du nom de la variable.

```
<?php
    $variable_globale = 5;

    function essai_globale(){
        global $variable_globale;
        echo $variable_globale."<br>"; //produit 5
        $variable_globale = 10;

        global $nouvelle_globale;
        $nouvelle_globale = 123;
    }

    essai_globale();

    echo $variable_globale."<br>"; //produit 10
    echo $nouvelle_globale."<br>"; //produit 123
?>
```

Les variables globales peuvent servir à simuler un passage par référence (qui n'existe plus en PHP).

Les variables statiques

Une variable statique (`static`) déclarée dans une fonction est une variable dont la valeur survit à plusieurs appels de cette fonction.

```

<?php
function test(){
    static $count = 0;
    echo $count."<br>";
    $count++;
}

test();    //produit 0
test();    //produit 1
test();    //produit 2
?>

```

Attention, une variable statique ne peut être initialisée qu'avec une valeur prédéfinie (pas un calcul). Ainsi, `static $var = 1+2;` n'est pas permis.

Inclusion de fichiers


Afin d'éviter de dupliquer du code, il est possible d'inclure un fichier dans fichier PHP.

```

<?php
    include "library.php";
    //...
?>

```

Insère le contenu du fichier `library.php` à cet endroit là ("copié collé")



Il est conseillé d'utiliser `include_once` afin d'éviter d'inclure le même fichier plusieurs fois.

Avec `include` et `include_once`, si le fichier à inclure n'existe pas, le programme continue. Si l'on ne veut pas ce comportement, il faut utiliser `require` ou `require_once`.

Les tableaux indexés

Comme en C++, les tableaux sont indexés à partir de 0.

Voici 3 façons d'initialiser le même tableau :

```
$tab[] = "a";    $tab[0] = "a";    $tab = array("a","b","c");
$tab[] = "b";    $tab[1] = "b";
$tab[] = "c";    $tab[2] = "c";    print_r($tab);

print_r($tab);    print_r($tab);
```


*La fonction `print_r()`
affiche un tableau de
façon lisible.*

```
Array
(
    [0] => a
    [1] => b
    [2] => c
)
```

Affichage d'un élément : `echo $tab[1];`

Les tableaux associatifs

Un tableau associatif est un dictionnaire (map) qui **associe des clés** (uniques) à **des valeurs**.



```
$utilisateur["nom"]    = "Dupont";
$utilisateur["prenom"] = "Jean";
$utilisateur["age"]    = 25;

echo $utilisateur["prenom"]."<br>";
```

On peut également initialiser un tel tableau à l'aide du mot clé `array` qui prend en paramètre une liste de couples `clé => valeur` séparés par des virgules :

```
$utilisateur = array("nom"    => "Dupont",
                    "prenom" => "Jean",
                    "age"    => 25);
```

La boucle foreach

La boucle foreach permet de parcourir un tableau facilement (de gauche à droite).

A chaque tour de boucle, un élément du tableau est assigné à la variable décrite après le mot as

```
$tab = array("a","b","c");
```

```
foreach($tab as $item){  
    echo $item."<br>";  
}
```

a
b
c

```
$utilisateur = array("nom" => "Dupont",  
                    "prenom" => "Jean",  
                    "age" => 25);
```

```
foreach($utilisateur as $elem){  
    echo $elem."<br>";  
}
```

Dupont
Jean
25

Remarquez la syntaxe pour itérer sur des paires clé-valeur

```
foreach($utilisateur as $cle => $valeur){  
    echo $cle." : ".$valeur."<br>";  
}
```

nom : Dupont
prenom : Jean
age : 25

Les tableaux à plusieurs dimensions (tableaux de tableaux)

Il est possible de créer des tableaux de tableaux : une case d'un tableau peut être un tableau...

On parle alors de tableau à plusieurs dimensions.

```

$users = array(
    array("username" => "boris",
        "password" => "1234"),
    array("username" => "benoit",
        "password" => "5678")
);

foreach($users as $user){
    foreach($user as $key => $value){
        echo $key." : ".$value."<br>";
    }
}
    
```

```

username : boris
password : 1234
username : benoit
password : 5678
    
```

Quelques fonctions à propos des tableaux

`is_array($tableau)` vérifie que la variable contient un tableau

`count($tableau)` renvoie le nombre d'éléments du tableau

`count($tableau, 1)` renvoie le nombre d'éléments du tableau et des sous-tableaux (comptage récursif).

`sort($tableau)` trie un tableau indexé

`ksort($tableau)` trie un tableau associatif sur base des clés

`asort($tableau)` trie un tableau associatif sur base des valeurs

`shuffle($tableau)` mélange un tableau indexé

Beaucoup d'autres fonctions existent. N'hésitez pas à vous renseigner.

Les tableaux super-globaux

PHP propose des **tableaux super-globaux accessibles de partout**.

Ils servent à **obtenir de l'information sur l'environnement** : le serveur, les données envoyées via la méthode GET ou POST, les variables de session (voir plus loin), ...

En voici quelques-uns :

<code>\$GLOBALS</code>	Les variables globales au script
<code>\$_GET</code>	Les variables passées via HTTP GET
<code>\$_POST</code>	Les variables passées via HTTP POST
<code>\$_FILES</code>	Les fichiers uploadés via HTTP POST
<code>\$_COOKIE</code>	Les variables passées via un cookie
<code>\$_SESSION</code>	Les variables de session
<code>\$_REQUEST</code>	L'information transmise par le navigateur (GET, POST et COOKIE)

Rappels sur les méthodes **GET** et **POST**

La méthode **GET** encode des valeurs (par exemple d'un formulaire) directement **dans l'URL**, sous forme de paramètres.

Ainsi un formulaire avec deux valeurs (**n=test** et **m=toto**) est envoyé à l'URL
`http://localhost/displayform.php?n=test&m=toto`

La méthode **POST** encode les différentes valeurs **dans le corps de la requête HTTP** (et donc *n'est pas visible dans la barre d'adresse*)

Bonnes pratiques pour la méthode POST

La méthode POST **doit** être employée quand:

Les **données sont grandes**: il y a une limite sur la taille maximum d'un URL (typiquement de l'ordre de 8k mais parfois moins – 2k dans certaines versions d'Internet Explorer)

Les **données contiennent un mot de passe**.

Les URL sont visibles dans la barre d'adresse et souvent enregistrés par le ou les serveurs.

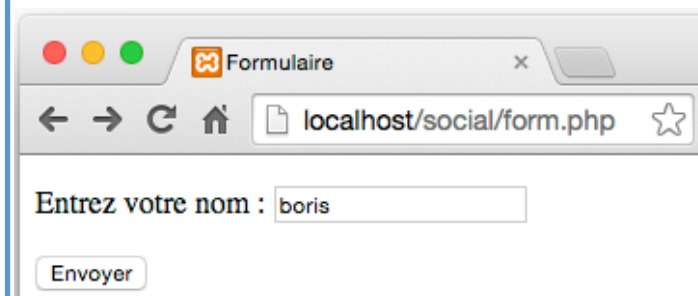
Le **traitement du formulaire implique des changements non réversibles** (ajouter un enregistrement dans une table par exemple).

Un GET devrait être «idempotent»: on devrait pouvoir exécuter la requête **plusieurs fois sans dommage** (exemple: une recherche)

Exemple : récupérer les données d'un formulaire

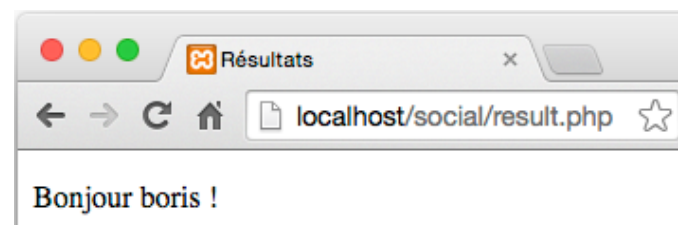
localhost/social/form.php

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Formulaire</title>
</head>
<body>
  <form action="result.php" method="POST">
    <p>Entrez votre nom : <input type="text" name="nom"/></p>
    <p><input type="submit" value="Envoyer"/></p>
  </form>
</body>
</html>
```



localhost/social/result.php

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Résultats</title>
</head>
<body>
  <?php
    if(isset($_POST["nom"])){
      $name = $_POST["nom"];
      echo "<p>Bonjour ".$name."</p>";
    }
  <?>
</body>
</html>
```



La fonction `isset()` permet de vérifier qu'une variable existe et possède une valeur (pas NULL).

Attention : ne jamais faire confiance aux données reçues

Attention, il est important de ne jamais faire confiance aux données reçues par le serveur web : elles sont encodées par un utilisateur qui pourrait se tromper ou être malveillant (injection de code, ...).

Vérifiez donc toujours les données reçues :

- La donnée est-elle présente (isset()) ?

- La donnée est-elle du bon type (casting, ...) ?

- La donnée est-elle cohérente (bornes, ...) ?

- La donnée ne tente-t-elle pas de corrompre le système ?

Nous en reparlerons plus tard.