[IMAC S2] Programmation Back-end

TD 01 : Bases de PHP

Support

Pascale Ho - 2018 - Ingénieur IMAC

Introduction

Ceci est un support vous rappelant les syntaxes de base en PHP. Rappelez-vous, la syntaxe de PHP est similaire à la famille du langage “C” (C, C++, Java), que des langages qui sont et qui seront vus à l’IMAC ! N’hésitez pas à relire ce document (ou sur la doc officielle) pour vous rappeler de la syntaxe.

TL;DR

- Le code PHP est délimité par `<?php [...] ?>`
- **La syntaxe est similaire au langage C**
- Chaque instruction se termine par un point-virgule **;**
- Une variable a toujours **$** comme préfixe
- Il n’y a pas besoin de préciser le type : cela se fait de manière dynamique
- On concatène les chaînes de caractères avec l’opérateur **.**
- On parcourt les tableaux associatifs avec une boucle **foreach**
- Par défaut les paramètres des fonctions se font par **valeur** (ou copie)
- Essayez d’écrire vos fonctions dans des fichiers séparés pour plus de lisibilité et de clarté
- Pour que le code soit bien lisible : n’oubliez pas de faire des **tabulations** !!

Syntaxe de base

Intégration et délimitation

Ce n’est pas une nouveauté : les pages web sont au format **.html**. Côté serveur, les pages web dynamiques sont au format **.php**. Mais du code source php peut être directement inséré dans du code html. Dans tous les cas, le code php est délimité par un conteneur de ce type : `<?php [...] ?>` (il existe d’autres délimitations mais elles sont à bannir lol)

Exemple de Hello World dans un fichier **.php** intégré dans du HTML :

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World avec du PHP</title>
</head>
<body>
<?php echo "<h1>J'adore ce que vous faites</h1>"; ?>
</body>
</html>
```

Autre exemple de délimitation dans un fichier .php :

```
<?php
echo "<h1>J'adore ce que vous faites</h1>";
?>
```

Commentaires et Instructions

Commentaires :

```
// Commentaire jusqu'à la fin de la ligne
# Commentaire style shell jusqu'à la fin de la ligne
/* Commentaire sur
plusieurs lignes */
```

Comme dans la famille “C”, chaque instruction se termine par un point-virgule **“;”.**

Variables, constantes et types

On identifie une variable avec un nom précédé par un **“$”** (exemple : `$toto`)

On affecte une valeur à une variable avec **“=”** (exemple: `$toto = 1055`)

Les types :

Type	Description	Exemple
integer	entier	12, 086
double	réel	1.055, 1.2e3
boolean	booléen	True, False
string	chaîne de caractères	“J’adore ce que vous faites”, ‘Bonsoir’
array	tableau	[1055, “T’es qui”]
object	objet	
NULL	variable sans valeur	NULL

Le typage est **implicite et dynamique** en PHP. Il n'est pas nécessaire de déclarer leur type au préalable, il est déterminé par la valeur qui lui est affectée. En reprenant l'exemple précédent, `$toto = 1055`, la variable toto est automatiquement un entier. C'est le **contexte** qui détermine le type d'une variable.

```php
<?php
$a;                 // NULL
$a = 1055;          // entier
$a = "Bonsoir";     // chaine de caracteres
$a = 1 + "1"        // $a = 2
?>
```

Cependant il est possible de préciser le type désiré : on parle de **cast**.

```php
$a = (integer) "10"; // $a = 10
```

Constantes

On définit des **constantes** avec une fonction define. Contrairement aux variables, les constantes n'ont pas de $ comme préfixe. Il est fréquent de construire des fichiers ne contenant que des constantes, ils serviront de paramètres de configuration ou de traduction de manière centralisée.

```php
define("nom_de_constante", "valeur");
define("PI", 3.14);
define("HEXA_BLACK", "#000000");
echo PI;    // affiche 3.14

// autre moyen
const CONSTANT = "Bonsoir";
```

Fonctions utiles

Deux fonctions utiles (notamment pour débugger) pour connaître les informations d'une variable :

- var_dump : affiche les informations structurées d'une variable (type + valeur)
- print_r : affiche les informations lisibles pour une variable (valeur)

```php
$a = 1055;
print_r($a);            // affiche 1055
var_dump($a);           // affiche int(1055)

$tab = array(1.5, "bonsoir", true);
print_r($tab);
/* affichage :
```

```
Array
(
      [0] => 1.5
      [1] => bonsoir
      [2] => 1
)
*/

var_dump($tab);
/* affichage :
array(3) {                  // tableau de taille 3
      [0] =>                // élément 0 de tab
      float(1.5)            // l'élément 0 de tab est un réel qui vaut
1.5
      [1] =>
      string(7) "bonsoir"
      [2] =>
      bool(true)
}
*/
```

Autres fonctions utiles sur les variables :

- `empty($var)` : vérifie si la variable est vide (renvoie `true`)
- `isset($var)` : vérifie si la variable existe
- `unset($var)` : détruit la variable $var
- `gettype($var)` : retourne le type de $var
- `settype($var, "type")` : modifie le type de $var (équivalent à cast)
- on a aussi des fonctions pour savoir si la variable est de tel type : is_long(), is double(), is_string(), is_array(), is_null(), …

Chaînes de caractères

Il y a deux façons de délimiter des chaînes : avec des simples quotes **'** ou avec des doubles **""**. Il faut savoir qu'il est possible de remplacer les variables à l'intérieur d'une chaîne de caractères par leur valeur.

En cas de problèmes avec le remplacement de variables, rajoutez des accolades {} autour de la variable (souvent lorsque la variable est complexe)

```
$a="bonsoir";
$tab = ["bonjour", "bonsoir"];
echo "Je te dis $a !!";        // affiche : Je te dis bonsoir !!
echo "Je te dis {$tab[1]} !!"; // une manière sécurisée est de mettre
des
```

```
                                // {} entre les variables complexes
```

Comme dans le langage C, on peut protéger certains caractères avec **** .

```
echo 'J\'adore ce que vous faîtes';
// affiche : J'adore ce que vous faîtes

$a="bonsoir";
echo "Je te dis \"$a\" !!";   // affiche : Je te dis "bonsoir" !!
echo "Je te dis \$a !!";      // affiche : Je te dis $a !!
```

Comme en C, on a **\n** pour revenir à la ligne et **\t** pour faire des tabulations.

Commandes utiles

Vous avez sûrement remarqué l'utilisation de **echo** dans les exemples ci-dessus. Naïvement, cela permet d'afficher une chaîne du texte au navigateur du client. Mais echo peut faire plus que ça ! Il envoie précisément des octets au navigateur du client, du coup il est possible d'envoyer :

- du texte,
- du contenu HTML, CSS, JavaScript, **JSON** (oh tiens donc !), …
- des données d'une image,
- du contenu d'un fichier PDF, …
- …

Reprenons notre Hello World :

```
<?php
echo "<h1>J'adore ce que vous faites</h1>";
?>
```

Le client va suite à cet exemple interpréter le résultat comme étant du code HTML.

Et comme en C, il existe la fonction printf :

```
printf('J\'%s ce que vous faîtes %03d fois', adore, 5);
// affiche : J'adore ce que vous faîtes 005 fois
```

Opérateurs

Concaténation de chaînes

En PHP, l'assemblage de chaînes de caractères se fait avec l'opérateur point : "**.**"

```
echo "T'es "."qui";     // affiche : T'es qui
```

```php
$a = 1+1;
echo "J'ai ".$a." ans"; // affiche : J'ai 2 ans
```

Opérateurs arithmétiques

Pareil qu'en C.

`$a + $b`	Somme
`$a - $b`	Différence
`$a * $b`	Multiplication
`$a / $b`	Division
`$a % $b`	Modulo (reste de la division entière)
`$a++`	Post incrémentation (retourne la valeur de $a puis augmente sa valeur de 1)
`++$a`	Pré incrémentation (augmente la valeur de $a de 1 puis retourne sa valeur)
`$a--` `--$a`	post décrémentation pré décrémentation

Opérateurs de comparaison

`$a == $b`	Égalité entre $a et $b (peu importe le type)
`$a != $b`	Différence entre $a et $b (peu importe le type)
`$a < $b`	Infériorité stricte
`$a > $b`	Supériorité stricte
`$a <= $b`	Inférieur ou égal
`$a >= $b`	Supérieur ou égal
`$a === $b`	Égalité entre $a et $b (valeur et type)
`$a !== $b`	Différence entre $a et $b (valeur et type)

Opérateurs logiques

`$a and $b` `$a && $b`	ET logique (vrai si $a et $b sont vraies)
`$a or $b` `$a || $b`	OU logique (vrai si $a ou $b ou les deux sont vraies)
`$a xor $b`	OU exclusif (vrai si $a ou $b est vraie mais pas les deux)

`!$a`	Négation (vrai si $a est faux)

Opérateurs binaires (bit à bit)

`$a & $b`	ET binaire
`$a | $b`	OU binaire
`$a ^ $b`	XOR binaire
`~$a`	Négation bit à bit
`$a << $b`	décalage de $a à gauche de $b rangs
`$a >> $b`	décalage de $a à droite de $b rangs

Autres opérateurs

Comme en C, il existe en plus du simple opérateur d'affectation "**=**" il existe des opérateurs combinés du type **+=**, ***=**, etc. N'oubliez que pour concaténer on utilise l'opérateur "**.**" .

```
$a = "J'adore";
$a .= " ce que vous faîtes";
```

N'oubliez pas l'**ordre de priorité** des opérateurs ! En cas de doute, n'oubliez pas les parenthèses !

http://php.net/manual/fr/language.operators.precedence.php

Structures de contrôle

Elles fonctionnent comme en C.

Conditions

Condition simple if :

```
if (condition)
{
    echo "true"; // bloc d'instructions exécuté si la condition est
vraie
}

if (condition)
    echo "true"; // {} optionnelles s'il n'y a qu'une seule instruction
```

Condition double if / else :

```php
if (condition){
    echo "plus petit";
}
elseif (condition){
    echo "égal";
    /* ce bloc est exécuté si la condition précédente est fausse mais
    cette condition est vraie */
}
else {
    echo "plus grand";
    // bloc exécuté si les condition précédentes sont fausses
}
```

Condition multiple switch :

```php
switch (val) {
    case v1:
        // instructions exécutées si val == v1
    case v2:
        // instructions exécutées si val == v1 ou v2
    ...
    default:
        // instructions dans tous les cas
}
```

val, v1, v2, … peuvent être de type integer, float, string, NULL

On peut se servir de **break** pour sortir de la structure de contrôle.

```php
switch (val) {
    case v1:
        // instructions exécutées si val == v1
        break; // sortie du switch
    case v2:
        // instructions exécutées si val == v2
        break;
    default:
        // instructions si val ne vaut pas v1 ou v2
}
```

Boucles

```php
while (condition) {
      /* tant que la condition est vraie on répète le bloc
d'instructions */
}
```

```
do {
    /* on exécute une fois le bloc puis on vérifie la condition
    si la condition est vraie : on répète */
} while (condition);
```

```
for (initialisation; condition; fin_de_chaque_iteration) {
    /* bloc d'instructions répété tant que la condition est vraie*/
}

for ($i = 0; $i < 10; $i++) {
    // ...
}
// équivalent à :
$i = 0;
while ($i < 10) {
    // ...
    $i++;
}
```

N'oubliez pas qu'on se sert de `break` pour sortir de la boucle et pour passer directement à l'itération suivante on utilise `continue` .

Tableaux

Un tableau PHP peut être vu comme une série de valeurs (peu importe le type). Chaque valeur a un index (ou une clé), qui peut être un entier ou une chaîne de caractères. On parle alors de **tableaux associatifs**.

Il existe deux types de tableaux selon leur manière de les indexer : soit de manière "classique" avec des index numériques (n'oubliez pas qu'un tableau commence à partir de l'index 0 !!), soit de manière associative. On peut mixer les deux manières.

```
// tableaux "classiques"
$tab1 = [1055, "bonsoir",  $variable];
$tab1 = array(1055, "bonsoir",  $variable);

$tab2[] = 1055;
$tab2[] = "bonsoir";
$tab2[] = $variable;

$tab3[0] = 1055;
$tab3[1] = "bonsoir";
$tab3[2] = $variable;
```

```php
// tableaux associatifs
$tab4['prenom'] = "Pascale";
$tab4["age"] = 25;
$tab4["ville"] = "Schlag-sur-Marne";
$tab4[1055] = $variable;

$tab5 = array(
    "prenom" => "Pascale",
    "age" => 25,
    "ville" => "Schlag-sur-Marne"
    1055 => $variable
);
echo $tab['ville'];     // affiche : Schlag-sur-Marne
```

Parcours de tableaux

On peut utiliser les boucles qu'on a vues plus haut pour parcourir les tableaux via les index des cases. Mais concernant les tableaux associatifs, ce n'est pas possible avec ces boucles ! Il existe des structures de contrôle qui permettent de parcourir les éléments d'un tableau : les boucles **foreach**.

```php
$variable = "ui";
$tab3[0] = 1055;
$tab3[1] = "bonsoir";
$tab3[2] = $variable;

foreach($tab3 as $value) {
    echo "val: $value\n";
}
/* affiche :
val: 1055
val: bonsoir
val: ui
*/

$tab4['prenom'] = "Pascale";
$tab4["age"] = 25;
$tab4["ville"] = "Schlag-sur-Marne";
$tab4[1055] = $variable;

foreach($tab4 as $key => $value) {
    echo "key: $key / val: $value\n";
}
/* affiche :
key: prenom / val: Pascale
```

```
key: age / val: 25
key: ville / val: Schlag-sur-Marne
key: 1055 / val: ui
*/
```

Fonctions

Comme PHP est faiblement typé, il n'y a pas besoin de renseigner le type des arguments et le type de la valeur de retour.

```php
function moyenne($a, $b) {
    return ($a+$b)/2. ; // 2. == 2.0
}
echo moyenne(2, 4);     // affiche : 3
```

Passage par référence

Par défaut, le passage des arguments se fait par **valeur** : il y a une copie de la valeur du paramètre dans une variable locale à l'intérieur de la fonction, il n'est donc pas possible de modifier dans la fonction la variable passée en paramètre. Pour pouvoir faire cela, il faut passer les paramètres par **référence** (avec un **&** devant les variables).

```php
function swap (&$a, &$b) {
    $c = $a;
    $a = $b;
    $b = $c;
}

$a = 12;
$b = 1055;
swap($a, $b);
echo $a;    // affiche : 1055
echo $b;    // affiche : 12
```

Arguments par défaut

Il y a possibilité de définir une valeur par défaut d'un argument s'il n'est pas appelé dans une fonction.

```php
function bonsoir($nom="t'es qui ?") {
    echo "Bonsoir $nom";
}
bonsoir();         // affiche : Bonsoir t'es qui ?
bonsoir("Victor"); // affiche : Bonsoir Victor
```

Inclusion de code

Il est conseillé (car **pratique et propre**) d'écrire des fonctions dans un fichier à part (library) afin de les réutiliser dans d'autres scripts PHP, et de ne pas les définir à plusieurs reprises dans plusieurs pages. (exemples : connexion à une base, authentification, …) Pour cela, on va utiliser les fonctions suivantes :

- include("mon fichier");
- include_once("mon_fichier");
- require("mon_fichier");
- require_once("mon_fichier);

```
// fichier ma_fonction.php
<?php
function ma_fonction($a){...}
?>

// autre fichier
[...]
require("ma_fonction.php");
$var = true;
ma_fonction($var);
[...]
```

Quelle différence entre include et require ? Lorsqu'il y a une erreur, **include** va générer un **warning** (le script va continuer à tourner) tandis que **require** va produire une **fatal error** (arrêt du script).

Et quelle est la différence entre avec ou sans _once ? Avec _once, si le code a déjà été inclus, il ne le sera pas une seconde fois.

Il est fréquent d'inclure un fichier par rapport au fichier courant :

```
include_once dirname(__FILE__).'/ma_fonction.php';
```

Gestion des erreurs

Dans certains cas, on souhaite arrêter l'exécution du code PHP (variables non définies, valeurs erronées, échec de connexion, …). On peut arrêter brutalement le code avec les fonctions **die** ou **exit** (ce sont des alias).

```
// quitte normalement
exit();
die();
// quitte avec un message
die("Message d'erreur à afficher");
```

```php
exit("Message d'erreur à afficher");
```

Ressources Supplémentaires

PHP :

- Manuel PHP du site officiel : http://php.net/manual/fr/index.php

[IMAC S2] Programmation Back-end

TD 01 : Bases de PHP

Compléments

Pascale Ho - 2018 - Ingénieur IMAC

Plus de syntaxe

Opérateur conditionnel ternaire (? :)

Cet opérateur aura besoin de trois opérandes. Cet opérateur est souvent utilisé comme raccourci pour la déclaration de if/else. Selon la condition de la première opérande, l'opérateur va retourner la valeur de la deuxième ou de la troisième opérande.

```
$var = condition ? expr1 : expr2;
// équivalent à
if (condition) {
    $var = expr1;
else {
    $var = expr2;
}
```

Syntaxe Heredoc

Il existe une façon de délimiter une chaîne de caractère sur plusieurs lignes (comme une chaîne entourée de **“”**) qui est la **syntaxe Heredoc** <<<. La structure est la suivante:

```
$a = <<<IDENTIFIANT
du texte sur
plusieurs lignes
IDENTIFIANT;
```

Un identifiant soit commencer juste après le dernier <. De plus il doit être indiqué à la fin de la ligne (sans tabulations). Il est composé uniquement de caractères alphanumériques et des underscores. Les variables peuvent être interprétées (n'oubliez pas d'utiliser les {} dans le cas de variables plus complexes). On va se servir notamment pour délimiter du contenu HTML ou des requêtes SQL.

```
$var = ["tu", "vous"];
$b = <<<HTML
```

```
    <div>J'adore ce que {$var[1]} faîtes</div>
HTML;
echo $b
/* affiche : <div>J'adore ce que vous faîtes</div>
sur le navigateur, ce sera interprété par un "J'adore ce que vous faîtes
dans une balise div
*/
```

Standards

Afin de travailler en groupe et aussi pour rendre le code lisible et clair, il est important de poser les mêmes règles sur la syntaxe. Il existe des standards appelés **PSR** (PHP Standards Recommandations) qui vous posent des règles qu'il faut suivre.

https://www.php-fig.org/psr/

PHP 7

Introduction

Le support de cours qui vous a été distribué fonctionne à partir de la version 5 de PHP. Mais depuis fin 2015, une nouvelle version a été distribuée (version 7). La différence entre les deux version concerne la performance : en effet pour un même code, PHP 7 peut l'améliorer de 25% à 70% (valeurs prises au pifomètre). Cette partie va vous présenter les nouveautés niveau syntaxe de PHP 7.

Nouvel opérateur <=> (Spaceship)

Cet opérateur est utilisé pour comparer deux expressions. Il retourne 0 si les deux expressions sont égales, 1 si celle de gauche est plus grande que celle de droite ou -1 si c'est l'expression de droite qui est la plus grande.

```
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1

echo "a" <=> "a"; // 0
echo "a" <=> "b"; // -1
echo "b" <=> "a"; // 1
```

Cet opérateur peut servir notamment pour classer une liste d'éléments.

Déclaration du type de retour

On sait qu'il n'est pas nécessaire de typer les paramètres d'une fonction. Depuis PHP 7 il est maintenant possible de déclarer un type de retour d'une fonction.

```
function foo(): array {
     return [];
}

function boo(): int {
     return null; // incorrect, doit être un entier
}
```

Opérateur ?? (Null coalescent)

Cet opérateur peut être utile car il permet de gagner du temps lorsqu'on veut tester l'existence d'une variable. Si elle existe ou ne vaut pas NULL, l'opérateur renvoie le contenu de cette variable, sinon l'opérateur renvoie la valeur après ??.

```
// récupère la valeur de $user sinon "t'es qui ?" si $user n'existe pas
$id = $user ?? "t'es qui ?" ;
// équivalent à
if (isset($user)) {
    $id = $user;
} else {
    $id = "t'es qui ?";
}
// équivalent à
$id = isset($user) ? $user : "t'es qui ?";
```

Tableaux constants

Pour définir des constantes on utilise la fonction define ou le mot-clé const. Il faut savoir qu'il n'était pas possible avant 7 de définir des tableaux constantes (avec const c'est possible mais uniquement dans les dernières versions de PHP 5). C'est maintenant possible de le faire avec define.

```
define('ANIMAUX', ["chien", "chat", "oiseau"]);
echo ANIMAUX[1]; // affiche : chat
```