Introduction à PHP

Alexandre Niveau

GREYC — Université de Caen

En partie adapté du cours de Jean-Marc Lecarpentier

PHP pour *PHP: Hypertext Preprocessor*

- Un langage de script, comme Python ou bash
- Spécialement conçu pour des applications web (architecture client/serveur)
- Idée de départ : rendre « simple » la réalisation de sites web dynamiques en permettant d'inclure des instructions dans le code HTML (en pratique, très souvent mal utilisé)
- Pas le meilleur langage de programmation [http://eev.ee/blog/2012/04/09/php-a-fractal-of-bad-design/], mais s'est progressivement amélioré dans les dernières années (versions 7 et 8)
- Toujours très utilisé, notamment par Wikipédia et Facebook (mais les internautes ne le voient pas), et surtout par les principaux CMS (WordPress, Drupal, Joomla), qui permettent aux administrateurs de modifier leur fonctionnement en changeant le code
- De nos jours, PHP se veut un langage généraliste — pas seulement utile pour faire des pages web.

Historique

- 1994 : création par Rasmus Lerdorf [http://lerdorf.com/] en bidouillant le langage Perl [http://www.perl.org]
- 1998 : PHP version 3, le succès est large
- 2000 : sortie de PHP4 et arrivée du moteur Zend [http://www.zend.com]
- 2004 : arrivée de PHP5, langage complet avec de réelles fonctionnalités objet
- 2015 : PHP7
- 2020 : PHP8
- Dès le départ, PHP a été construit incrémentalement, en fonction des besoins (de Rasmus Lerdorf au début, puis de la communauté), et en agrégeant des idées de diverses origines
- Cela se sent dans la syntaxe : les variables et les boucles sont celles de Perl, les références sont celles de C++, le modèle objet est celui de Java, etc.
- Il y a de nombreuses façons de faire la même chose (plusieurs syntaxes pour le `if` par exemple)

- On ne verra pas tout le langage !

Les bases

```php
<?php
echo "Hello, world!\n";
```

- Un script PHP commence par les caractères `<?php`
- chaque instruction se termine par un point-virgule
- la commande `echo` permet d'afficher quelque chose
- Pour exécuter un script, on appelle l'interpréteur avec la commande `php`

```
$ php mon_script.php
```

```
Hello, world!
```

Commentaires

- À la C++ avec `//`, ou à la shell avec `#` (la suite de la ligne est ignorée)
- À la C avec `/* ... */` (permet de commenter plusieurs lignes)

Principaux types d'erreur

- Parse error : erreur de syntaxe dans le programme, il n'est pas exécuté du tout
- Fatal error : arrête le programme
- Warning : alerte, n'interrompt pas l'exécution
- Notice : niveau d'alerte minimal — à ne pas négliger !

Variables

```php
<?php
$message = "Bonjour, oh monde !";
echo $message;
```

```
Bonjour, oh monde !
```

- Les variables ne sont pas déclarées
- Le nom d'une variable est composé du caractère `$` suivi d'un symbole fait de lettres minuscules ou majuscules, de chiffres et de soulignés.
- Les noms de variables sont sensibles à la casse : `$toto` ≠ `$Toto` ≠ `$TOTO`
- Les variables systèmes commencent par un caractère souligné (ex : `$_REQUEST`) et il est donc conseillé de ne pas utiliser de nom de variable commençant par un souligné.
- Les valeurs sont *faiblement typées* ; il existe les types entier, booléen, flottant,

chaîne de caractères, tableau, et objet, mais les valeurs peuvent changer de type en fonction du contexte (conversions implicites)

- Toute variable n'ayant pas encore de valeur est de type `NULL`, qui n'a que la valeur `NULL`
- Utiliser une variable non initialisée n'est pas une *erreur* — cela génère simplement une « notice ». Les variables non initialisées contiennent la chaîne vide. Il est **très déconseillé de le faire** : un code propre ne doit générer aucune notice !

Constantes

- `define('TOTO', valeur);` pour définir la constante `TOTO`
- NB: les constantes ne commencent pas par un dollar.
- NB: à la déclaration avec `define`, on met le nom de la constante dans une chaîne de caractères, mais on utilise la constante sans guillemets !
- Accepte seulement une valeur scalaire (booléen, entier, float ou chaîne de caractères), ou un tableau (PHP ≥ 7)
- Portée des constantes : globale
- Attention : si on utilise une constante sans l'avoir définie, PHP la définit comme contenant son nom en chaînes de caractères, et génère une notice
- On peut aussi déclarer une constante avec le mot-clef `const` :

```
const TOTO = 'Toto Dupont';
```

Cependant ça ne fonctionne pas tout à fait pareil : les constantes déclarées avec `const` sont définies à la compilation, alors que celles déclarées avec `define` le sont à l'exécution. En pratique :

```
$var = 'une variable';
define('TOTO', $var);  /* OK */
const TUTU = $var;  /* Fatal error */
```

Inclusion de fichiers

- On a très souvent besoin d'inclure du code d'un autre fichier PHP (ou HTML)
- 2 possibilités
  - `include` : génère simplement un warning en cas d'échec
  - `require` : arrête le programme avec une erreur fatale en cas d'échec, donc rien ne s'affiche
- Instructions `include_once` et `require_once` pour éviter une inclusion multiple
- La gestion des chemins (quels accès sont autorisés ou non, quelle est la référence des chemins relatifs dans un script inclus depuis un autre script…) est relativement infernale : ne pas essayer de faire des choses subtiles, ou prévoir un stock d'aspirine

Déboguer

- Fonctions utiles
  - `print_r`, `var_export` ou `var_dump` affichent le contenu d'une variable de façon lisible (démo [demo/dump.php])
  - `debug_print_backtrace()` affiche la backtrace de PHP (appels aux fonctions, aux fichiers inclus / requis, etc)

Opérateurs

- Arithmétiques : `+ - * / %`
- Concaténation de chaînes : `.` — *Attention*, ce n'est pas `+` !)

```php
<?php
echo "cou" . "cou\n";  // concaténation
//echo "cou" + "cou\n";  // somme : les chaînes étaient implicitement
                         // converties en entiers avant PHP 8,
                         // maintenant ça génère une TypeError
echo "\n";
echo 7 . 2;  // affiche 72
echo "\n";
echo 7. . .2;  // affiche 70.2 (équivaut à "7." . ".2")
```

Résultat :

```
coucou

72
70.2
```

- Assignation : `= += -= *= /= .=`
- Comparaisons : `== != <> <= >= === !==` (voir plus loin)
- Logiques : `|| && and or xor !`
- Conversion explicite de type : `(int) (float) (string) (array) (object)`
- Opérateur ternaire : `$condition ? 'valeur si vrai' : 'valeur si faux'` (attention, leur chaînage est contre-intuitif [http://stackoverflow.com/questions/20559150/ternary-operator-left-associativity] : il faut toujours utiliser des parenthèses)
- Voir les opérateurs [http://fr.php.net/manual/fr/language.operators.php] dans le manuel [http://fr.php.net/manual/fr/]

[#double-egal]

Opérateurs d'égalité faible

- Les opérateurs de comparaison == et != sont très dangereux, pour deux raisons distinctes.

  1. Il y a des conversions implicites lorsque l'on compare des valeurs de types différents (typage faible).

     Une conséquence amusante est que l'opérateur == n'est *pas transitif* : par exemple, `'0' == 0` (la chaîne de caractère est automatiquement convertie en entier), `0 == 'toto'` (idem, et la conversion donne 0 car la chaîne ne commence pas par un nombre), mais `'0' != 'toto'`.

Les résultats peuvent être surprenants : `false == '0'` mais `false != '00'` par exemple. (Tableaux des résultats de comparaison en fonction des types) [http://php.net/manual/en/types.comparisons.php]

2. Beaucoup plus grave : si on compare deux chaînes de caractères « numériques » (c'est-à-dire qu'elles contiennent un nombre), **c'est leur valeur numérique qui est comparée** !

   ➤ Par exemple, `'1' == '1.0'`, `'10' == '010'`, et `'1000' == '1e3'` (notation scientifique)… et les chaînes de caractères suivantes sont *toutes égales* (car leur valeur numérique est 0) : `'0'`, `'0000'`, `'0.'`, `'0.0000'`, `'0e1'`, mais aussi `'1234e-56789'` (à cause de l'arrondi) !

Opérateurs d'égalité stricte

- PHP fournit également un opérateur d'égalité stricte (identité) `===` (et sa négation `!==`) : il vérifie l'égalité des valeurs et des types, *et* se comporte correctement sur les chaînes de caractères.

  ➤ **Toujours utiliser `===` et `!==` pour comparer des chaînes de caractères** (ou des données dont le type n'est pas certain) !

  ➤ En revanche, pour comparer des nombres qui peuvent être entiers ou flottants, il peut être opportun d'utiliser `==` (car par exemple `0 !== 0.0`, ce qui dans certains cas n'est pas ce qu'on veut)

Chaînes de caractères et substitution des variables

- PHP utilise deux délimiteurs pour les chaînes de caractères, qui fonctionnent comme en shell :
  - les caractères d'une chaîne entre apostrophes (*single quotes*, `'`) sont laissés tels quels
  - dans une chaîne entre guillemets (*double quotes*, `"`, les variables sont substituées par leur valeur, et on peut utiliser certaines séquences d'échappement
- Les séquences d'échappement les plus utiles : `\n` pour fin de ligne, `\t` pour tabulation, `\$` pour dollar, `\"` pour `"` et `\\` pour `\`
- Entre simples quotes, il n'existe que `\'` et `\\` comme échappement. Pour `x` autre que `'` ou `\`, `\x` donne `\x`.

Caractères spéciaux dans les chaînes

- PHP manipule les chaînes de caractères comme des suites d'octets
- Si on met un caractère spécial, la chaîne contiendra la suite d'octet correspondant à l'encodage du fichier `.php` dans lequel elle est définie
- Les fonctions de manipulation de chaînes « de base » considèrent qu'un caractère = un octet, ce qui n'est pas forcément vrai si les chaînes ne contiennent pas de l'ASCII
- Il faut utiliser les fonctions dédiées aux chaînes « multi-octets » [https://

www.php.net/manual/fr/ref.mbstring.php]

Structures de contrôle

- Les structures suivantes fonctionnent de même manière qu'en C (et Java)
  - `if ... else if ... else`
  - `for (...; ...; ...)`
  - `while`
  - `do ... while`
  - `switch` avec `case ... break;`
- Voir les détails dans le manuel [http://fr.php.net/manual/fr/language.control-structures.php]

Fonctions

```php
<?php
function fact($x) {
  if ($x <= 1)
    return 1;
  return $x * fact($x-1);
}

echo "5! = " . fact(5);
echo "50! = " . fact(50);
```

```
5! = 12050! = 3.0414093201713E+64
```

- Mot-clef `function`
- On déclare une liste de paramètres formels
- La valeur des paramètres formels est une copie (paresseuse) des paramètres réels
- `return` pour sortir de la fonction et retourner (ou pas) une valeur
- Les variables utilisées dans une fonction sont locales à la fonction

Détails sur les fonctions

- Les noms des fonctions ne sont **pas** sensibles à la casse :

```php
<?php
function toto() { echo "fonction toto\n"; }
function Toto() { echo "fonction Toto\n"; }
```

```
PHP Fatal error:  Cannot redeclare Toto() (previously declared in
/blabla/toto.php:2) in /blabla/toto.php on line 3
```

- **Attention**, il est autorisé de passer trop d'arguments à une fonction : cela ne génère même pas une notice.
- On peut définir des paramètres optionnels ayant une valeur par défaut :

```php
<?php
function saluer($nom="chère personne") {
  echo "Bonjour, $nom !\n";
}

saluer("Jean-Michel");
saluer();
```

```
Bonjour, Jean-Michel !
Bonjour, chère personne !
```

Bibliothèques de fonctions

- Existence de nombreuses bibliothèques, par ex.
- BC Maths [http://fr.php.net/manual/fr/ref.bc.php] pour travailler sur des nombres de grande taille
- Date et heure [http://fr.php.net/manual/fr/book.datetime.php] pour créer et/ou afficher des dates
- CURL [http://fr.php.net/manual/fr/ref.curl.php] pour gérer la communication avec différents serveurs et protocoles
- Images [http://fr.php.net/manual/fr/ref.image.php] avec un ensemble de fonctions utilisant la bibliothèque GD (exemple [demo/histo.php])
- Liste complète [http://fr.php.net/manual/fr/funcref.php]

Types composites

- PHP propose deux types composites
  - les objets, qu'on verra plus tard
  - les tableaux (*arrays*)
- On différencie usuellement tableaux numériques et tableaux associatifs, mais c'est une seule et même structure de données, le « tableau associatif ordonné » (*ordered map*). On peut d'ailleurs mélanger les deux types d'indices.

Tableaux numériques

```php
<?php
$couleurs = array(
    "black", "yellow", "fuchsia",
);

$couleurs[] = "pink";

echo "My favorite colors:\n";
for ($i = 0; $i < count($couleurs); $i++) {
  echo " - the color {$couleurs[$i]};\n";
}
```

```
My favorite colors:
 - the color black;
 - the color yellow;
 - the color fuchsia;
 - the color pink;
```

- La fonction `array` crée un tableau ; on peut aussi utiliser la syntaxe « littéral »

avec des crochets : `$tab = [ 4, 8, 7 ];`

- La fonction `count` donne le nombre d'entrées du tableau, qui est dynamique.
- L'accès à un élément de tableau se fait à l'aide de l'opérateur « crochets ».
- On peut ajouter un élément à la fin du tableau en faisant `$couleurs[] = "rouge";`
- Accolades pour la substitution dans les chaînes (pas toujours nécessaire mais plus sûr)

Tableaux associatifs

- PHP permet de mettre des chaînes de caractères comme indices ; on parle alors de tableaux associatifs
- Ils se comportent à peu près comme des dictionnaires en Python, mais sont *ordonnés*

```php
<?php
$couleurs = array(
    "red" => "#FF0000",
    "white" => "#FFFFFF",
    "blue" => "#0000FF",
);

echo $couleurs["blue"];
```

```
#0000FF
```

- On peut préciser la valeur de l'indice dans l'expression `array(...)` avec `=>`
- On peut aussi ajouter un élément au tableau en faisant `$couleurs["green"] = "#00FF00";`

Parcours d'un tableau

- foreach permet d'itérer de façon simple sur toutes les valeurs d'un tableau :

```php
<?php
foreach ($couleurs as $c) {
  echo "$c\n";
}
```

```
#FF0000
#FFFFFF
#0000FF
```

- La syntaxe permet aussi de récupérer l'indice (la clef) en utilisant `=>`

```php
<?php
foreach ($couleurs as $c => $rgb) {
  echo "la couleur $c a pour code hexa
}
```

```
la couleur red a pour code hexa #
la couleur white a pour code hexa
la couleur blue a pour code hexa
```

- L'ordre d'affichage sera toujours le même, c'est-à-dire celui dans lequel les éléments ont été ajoutés (les tableaux associatifs en PHP sont ordonnés)

Spécifications et normes

- Manuel de PHP [http://www.php.net/manual/fr/index.php] (en français)

Tutoriels

- Tableaux PHP [http://sylvie-vauthier.developpez.com/tutoriels/php/grand-debutant/?page=tableaux] (en français)

Lectures complémentaires

- What every programmer needs to know about encodings and charsets to work with text [http://kunststube.net/encoding/] (avec des détails sur PHP)

 [http://creativecommons.org/licenses/by-nc-sa/4.0/]

Ce cours est mis à disposition selon les termes de la licence Creative Commons Attribution — Pas d'utilisation commerciale — Partage dans les mêmes conditions 4.0 International [http://creativecommons.org/licenses/by-nc-sa/4.0/].