

# Introduction au langage PHP

Cyril Rabat

`cyril.rabat@univ-reims.fr`

Licence 2 Informatique - Info0303 - Programmation Web 2

2020-2021



## Cours n°1

### *La programmation Web dynamique Éléments du langage PHP*

Version 13 septembre 2020

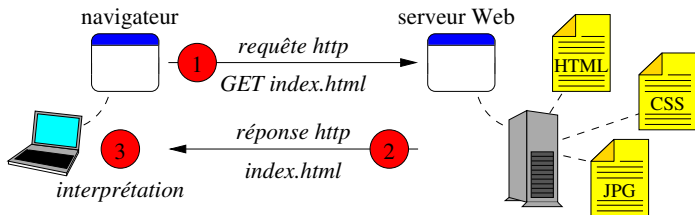
# Table des matières

- 1 Introduction
- 2 Éléments du langage PHP
- 3 Les structures de sélection et boucles
- 4 Fonctions en PHP

# La programmation Web statique

- Pas d'interprétation du côté serveur
- Téléchargement des différents éléments
- Interprétation par le navigateur

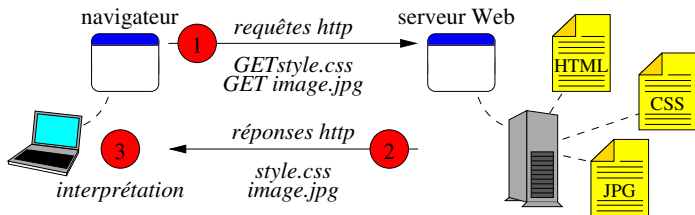
## Illustration



# La programmation Web statique

- Pas d'interprétation du côté serveur
- Téléchargement des différents éléments
- Interprétation par le navigateur

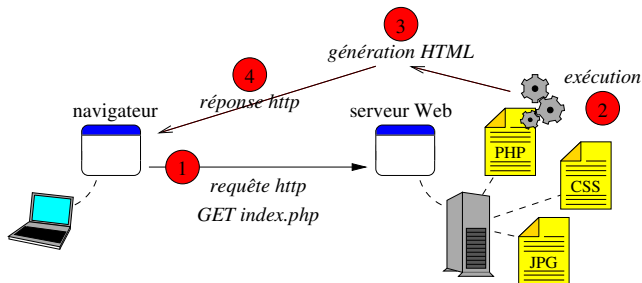
## Illustration



# La programmation Web dynamique

- Interprétation du côté serveur  
↪ Exécution d'un langage : ASP, PHP, Python, Java, etc.
- Génération à la volée d'éléments

## Illustration



# Différences par rapport à la programmation classique

- Programmation classique :
  - Application exécutée une fois pour toute
  - Interactions avec l'utilisateur durant la même exécution
  - Le contenu des variables est conservé en mémoire entre chaque action
  - Pas de limite du temps d'exécution
- Programmation Web :
  - Chaque requête entraîne une exécution
  - Le temps d'exécution est limité (par le serveur Web)
  - Pas de conservation de la mémoire d'une exécution à l'autre
  - Chaque action de l'utilisateur entraîne une nouvelle exécution

# Notion de client/serveur dans le contexte du Web

- Modèle client/serveur (voir INFO0503) :
  - Le client (le navigateur) envoie une requête au serveur
  - Le serveur Web traite la requête
    - ↪ Exécution de scripts, appels à la base de données, etc.
  - La réponse est envoyée au client
    - ↪ Mise en forme des données reçues
- Le protocole utilisé est HTTP

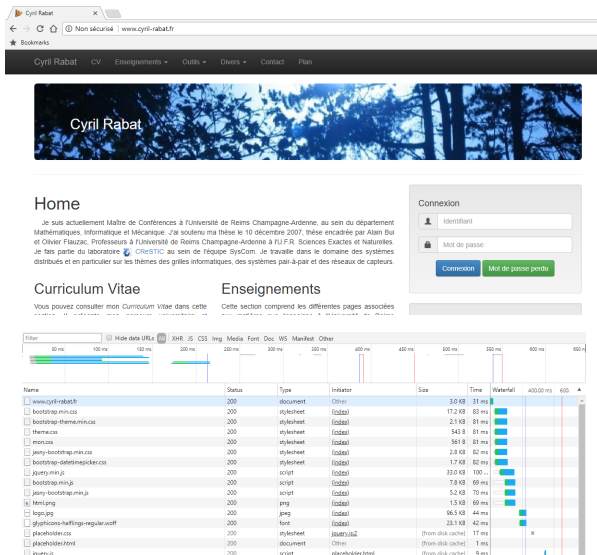
# Présentation du protocole HTTP

- HTTP pour *HyperText Transfer Protocol*
- Fait partie de la couche applicative d'Internet
- Basé sur le modèle de communication client/serveur :
  - Client :
    - ↪ Envoie des requêtes, reçoit et affiche des objets Web
    - ↪ Exemple : un navigateur Web
  - Serveur : serveur Web qui attend des requêtes et y répond
- Établissement de connexions entre les clients et le serveur...
- ... mais protocole sans état !

En pratique, plusieurs connexions sont établies en parallèle avec le serveur Web (cela dépend du navigateur Web)

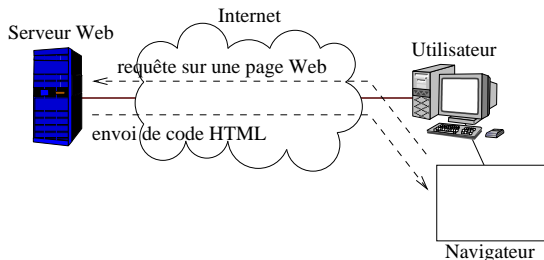


# Exemple de requêtes vers un site Web



# Fonctionnement de HTTP

- HTTP est encapsulé dans TCP :  
↪ Le serveur Web écoute sur le port 80
- Une fois la connexion établie, échanges de messages HTTP  
↪ Messages au format ASCII



## Les messages : requêtes

- Première ligne : ligne de requête  
↪ Commandes GET, POST, HEAD (PUT et DELETE non activées)
- Lignes suivantes : lignes d'en-tête  
↪ Fin indiquée par un retour charriot (ligne vide)
- Les données éventuelles sont placées dans la suite

### Exemple

```
GET /~crabat/ HTTP/1.1
Host: cosy.univ-reims.fr
User-agent: Mozilla/5.0
Accept: text/html, image/gif, image/jpeg
Accept-language: fr
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1, utf-8; q=0.7, *, q=0.7
Keep-Alive: 300
Connection: keep-alive
```

## Les messages : réponses

- Première ligne : ligne d'état (protocole, code d'état, message d'état)
- Lignes suivantes : lignes d'en-tête
  - ↪ Fin indiquée par un retour charriot (ligne vide)
- Les données éventuelles sont placées dans la suite
- Fin du message : les données

### Exemple

```
HTTP/1.x 200 OK
Connection: close
Date: Wed, 03 Feb 2020 06:09:02 GMT
Server: Apache
Content-Length: 6821
Content-Type: text/html
```

données données données données données

# MIME : *Multimedia Mail Extension*

- Définit un ensemble de types pour déchiffrer les données
- Quelques types :
  - Texte (plain, html)
  - Image (jpeg, fig), audio (basic, 32kadpcm), vidéo (mpeg, quicktime)
  - Application (msword, octet-stream)
- Lignes d'en-tête en HTTP :
  - Content-Type: image/jpeg → type MIME
  - Content-Length: 66 → taille en octets
  - Content-Transfer-Encoding: base64 → encodage des données

# Codes de réponse et authentification

- Codes de réponse :
  - 200 *OK* : requête réussie et l'objet demandé est à la suite
  - 301 *Moved Permanently* : l'objet demandé a changé définitivement de place, son nouvel emplacement est indiqué dans la suite
  - 400 *Bad Request* : la requête est erronée
  - 404 *Not Found* : le document n'est pas disponible sur le serveur
  - 505 *HTTP Version Not Supported*
- Authentification
  - HTTP fournit des codes et des en-têtes d'états pour l'authentification
  - Procédure :
    - 1 Le client émet une requête
    - 2 Le serveur retourne le code 401 *Authorization Required*
    - 3 Le client envoie alors les informations (nom utilisateur et mot de passe)

# Serveur Web

- Application capable de répondre à des requêtes HTTP
  - ↪ Par défaut, sur le port 80
  - ↪ Serveur HTTP
- Couplé à un moteur PHP (ou autre), à un gestionnaire de bases de données, etc.
  - ↪ L'ensemble forme le Serveur Web
- Quelques exemples :
  - Apache, IIS, Nginx...
- Possibilité d'utiliser des distributions complètes :
  - ↪ Wamp (pour Windows, Apache, MySQL, PHP), Lamp (Linux)
  - ↪ EasyPHP...
- Ou installer chaque élément indépendamment...

# Le langage PHP

- PHP pour *Personal Home Pages*
- Code inclus dans du HTML (vision réductrice !)
- Exécution du code sur le serveur
- Permet d'accéder aux ressources du serveur :
  - Les fichiers
  - Les bases de données
  - Les sessions. . .
- Exécution dans le contexte du serveur Web
- Extension des fichiers `.php` :
  - ↪ Permet d'identifier le code à exécuter par le moteur PHP



# Utilisation du PHP dans du HTML

- Code inclus en tout point du HTML
- Utilisation des balises `<?php` et `?>` :
  - ↪ Le code PHP est interprété par le serveur
  - ↪ Remplacement du code par le résultat
- Pas de PHP sur le client !
- Peut générer du HTML, CSS, *Javascript*...

## Remarque

Si le fichier se termine par un script PHP, il est conseillé de ne pas utiliser la balise fermante `?>`

# Syntaxe

- Proche du C/C++/Java
- Syntaxe très riche :
  - Nombreux opérateurs
  - Beaucoup de souplesse
- Instructions séparées par des ";"
- Commentaires :
  - Sur une ligne : //
  - Sur plusieurs lignes : /\* ... \*/
- Utilisation de la POO :
  - Pas obligatoire
  - Permet cependant une structuration du code
  - Possède toutes les fonctionnalités classiques :
    - ↪ Héritage, polymorphisme, surcharge, redéfinition, ...

# Un exemple

## Contenu du fichier index.php

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8">
    <title>Exemple</title>
  </head>
  <body>
    <h1>Bonjour</h1>
    <p>Aujourd'hui, nous sommes le <?php echo date("d/m/Y"); ?>.</p>
  </body>
</html>
```

## Exécution

**Bonjour**

Aujourd'hui, nous sommes le 30/08/2020.

## Les versions

- Actuellement, version 7 (7.4)
  - ↪ Plusieurs *releases* corrigeant différents bugs
- Apporte une rigueur sur la syntaxe :
  - ↪ Certaines pratiques deviennent interdites !
- Ajout d'opérateurs
  - ↪ Pas nécessaires, mais simplification du code
  - ↪ Exemple : ?? (*null coalescing*) et <=> (comparaison)
- Accélération diverses
- Optimisation mémoire...

### Attention

Les codes écrits dans les dernières versions ne sont pas compatibles avec les versions précédentes !

# Exécuter du PHP

- Contrairement au HTML/CSS : nécessite un serveur Web
- Installation d'une distribution : *Wamp*, etc.
- Fichiers placés dans le répertoire `www` (répertoire d'installation)  
    ↪ Création d'un sous-répertoire `CM01` (par exemple)
- Pour visualiser :
  - Démarrer les serveurs (automatisé avec *Wamp*)
  - Ouvrez un navigateur
  - Saisissez l'adresse : `http://localhost/CM01/index.php`

## Sortie écran

- La sortie écran correspond aux données qui seront envoyées au client
- Elles seront interprétées en fonction du type spécifié
- `echo` : affichage de chaînes, de variables de type primitif  
↪ Exemple : `echo "Bonjour";`
- `print_r` : tous types acceptés (tableaux, objets, *etc.*)  
↪ Exemple : `print_r("Coucou");`
- `var_dump` : plutôt en mode développement (affichage du type)  
↪ Exemple : `var_dump($a)`

### Remarque

La sortie correspond à la génération de code HTML ou du texte.  
Le PHP permet de générer d'autres types de données (XML, JSON, images, *etc.*).

# Variables

- Utilisation du \$ pour indiquer le nom de la variable  
↪ Nom de variable : lettres, chiffres (sauf au début), \_
- Noms sensibles à la casse (préférez les minuscules)
- Pas de déclaration au préalable :  
↪ Une variable peut changer de type à tout moment
- 10 types basiques :
  - Types scalaires : boolean, integer, float, string
  - Types composés : array, object, callable, iterable
  - Types spéciaux : resource, null

## Exemple

```
<?php
$a = 12;
echo "La_valeur_de_a_=". $a;
```

## Fonctions utiles pour les variables

- `isset` : teste si une variable existe  
↪ Retourne `true` ou `false` (booléen)
- `empty` : teste si une variable est vide  
↪ Retourne `true` si une chaîne n'existe pas ou est égale à ""  
↪ Pour les autres types : 0 pour un entier, `NULL`, "0", *etc.*
- Pour supprimer une variable : `unset`
- `gettype` : retourne le type de la variable  
↪ `settype` permet de spécifier le type d'une variable
- `is_int` : teste si la variable est un entier  
↪ Idem avec `is_string`, `is_float`, `is_double`
- Conversions : `floatval`, `intval`, *etc.*



# Généralités sur les chaînes de caractères

- Contient une suite de caractères codés sur 1 octet  
↪ Pas de support de l'Unicode (problèmes insolubles du PHP 6)
- Entourée de guillemets simples ( ' )
- Entourée de guillemets doubles ( " ) :
  - Support des caractères d'échappement  
↪ Exemple : `\n`, `\t`, `\"`, etc.
  - Évaluation des variables  
↪ Entourez-les de `{` et `}` pour les types complexes

## Avec guillemets simples

```
$a = 12;  
$s1 = 'Coucou\nValeur_de_a=$a.';  
echo $s1;
```

## Sortie

Coucou\nValeur de a=\$a.



# Généralités sur les chaînes de caractères

- Contient une suite de caractères codés sur 1 octet  
↪ Pas de support de l'Unicode (problèmes insolubles du PHP 6)
- Entourée de guillemets simples ( ' )
- Entourée de guillemets doubles ( " ) :
  - Support des caractères d'échappement  
↪ Exemple : `\n`, `\t`, `\"`, etc.
  - Évaluation des variables  
↪ Entourez-les de `{` et `}` pour les types complexes

## Avec guillemets doubles

```
$a = 12;  
$s1 = "Coucou\nValeur_de_a=$a.";   
echo $s1;
```

## Sortie

Coucou Valeur de a=12.



# Manipulation

- Accès aux caractères avec la notation [ ]  
↪ Exemple : `$tab[0]`
- Taille d'une chaîne : `strlen`
- Concaténation de deux chaînes : opérateur `"."`
- Conversion en chaîne automatiquement si nécessaire  
↪ Possible d'utiliser `strval` pour forcer la conversion
- Conversion d'une chaîne en valeur :  
↪ Idem (en entier ou réel selon la chaîne)  
↪ Rappels : `floatval`, `intval`, *etc.*

## Attention

Ne pas confondre le `"."` et le `"+"`.

# Fonctions de l'API

- Très nombreuses fonctions :  
↪ Manipulation, extraction, conversion, *etc.*
- Manuel PHP :  
<http://php.net/manual/fr/ref.strings.php>
- Quelques exemples :
  - `strpos` : première occurrence dans une chaîne  
↪ `stripos` : idem sans tenir compte de la casse
  - `strcmp` : comparaison binaire de deux chaînes
  - `strtoupper` : renvoie la chaîne en majuscules  
↪ `strtolower` en minuscules
  - `substr` : retourne un segment de chaîne
  - `explode` : découpe une chaîne en segments en fonction d'un délimiteur

## *Heredoc et Nowdoc*

- Syntaxe PHP permettant de définir des chaînes de caractères sur plusieurs lignes
- Utilisation de <<< suivi d'un identifiant
- Fin de la chaîne sur une ligne vide contenant l'identifiant suivi d'un ' ;'

### Exemple de *Heredoc*

```
$a = 2;  
echo <<<HTML  
    Le contenu de la variable est {$a}.  
HTML;
```

- *Nowdoc* : comme le Heredoc mais fonctionnent comme les chaînes avec '
- L'identifiant du début doit être encadré par des '

# Généralités sur les tableaux

- Permettent de regrouper des données différentes sous un même nom
- En PHP : cartes ordonnées
  - À une clé, on associe une valeur
  - L'ordre d'ajout est conservé
- Utilisation du mot-clé : `array`  
↔ Syntaxe courte : crochets
- Pour accéder à un élément / le modifier :  
↔ Utilisation des crochets

## Exemple

```
<?php
$stab = array(1, 2, "Chaine");
var_dump($stab);
echo $stab[0];
```

## Exemple (sans array)

```
<?php
$stab = [1, 2, "Chaine"];
var_dump($stab);
echo $stab[0];
```

# Tableaux associatifs

- Pour créer un tableau associatif, utilisation de "=>"
- Une clé = une chaîne de caractères ou un entier
- Accéder à la valeur associée à une clé : `$tab["cle"]` (ou `$tab['cle']`)
- Pour vérifier l'existence d'une clé : `array_key_exists`
- Pour supprimer un élément : `unset`

## Exemple

```
<?php
$tab = array("prenom" => "Cyril", "nom" => "Rabat", "age" => 35);

echo $tab["prenom"]." ".$tab["nom"]." ".$tab["age"]." _an(s) ";
```

## Fonctions utiles

- Nombre d'éléments : `count` (ou l'alias `sizeof`)
- Ajout d'un (ou plusieurs) élément(s) à la fin : `array_push` ou `[] =`  
    ↪ Exemple : `array_push($tab, 2, 3, 4)`  
    ↪ `array_push($tab, 2)` est équivalent à `$tab[] = 2`
- Tri : `sort` (clés modifiées  $\neq$  `asort`)
- Vérifie la présence d'une valeur : `in_array`
- Plus de fonctions :  
    ↪ Manuel PHP :  
    <http://php.net/manual/fr/ref.array.php>



## Affichage du contenu d'un tableau

- Impossible d'utiliser `echo` :  
↪ Affiche par défaut "Array"
- Utilisation de `print_r` ou `var_dump`
- Ou bien utilisation d'une boucle

### Exemple avec une boucle `for`

```
<?php
$tab = [1, 2, 3, 4];

for($i = 0; $i < count($tab); $i++)
    echo $tab[$i]."  
";
```

## Opérateurs sur les tableaux

- `+` : union de deux tableaux  
↪ Attention ! Réalisé sur les clés
- `==` : teste l'égalité de deux tableaux (mêmes valeurs, mêmes clés)
- `===` : idem, mais vérifie aussi l'ordre
- `!=`, `!==` : teste si les deux tableaux sont différents ou pas identiques

### Attention

À utiliser avec prudence, le comportement peut être inattendu !

# Constantes

- Mot-clé `define` ou `const` (plutôt dans les classes)
- Définition d'une variable non modifiable
- Depuis PHP 7.0, possible d'utiliser des tableaux avec `define`
- Ne peuvent être évaluées dans des chaînes (avec `'`)

## Exemple

```
<?php
define("AUTEUR_SITE", "CYRIL_RABAT");
define("DATE_MODIF", [ 30, 08, 2020 ]);

echo "L'auteur du site est ".AUTEUR_SITE;
echo "Date de modification: ".DATE_MODIF[0]."/".
    DATE_MODIF[1]."/".DATE_MODIF[2];
```

# Constantes magiques

- Constantes accessibles suivant les extensions activées de PHP
- `__LINE__` : ligne courante dans le script
- `__FILE__` et `__DIR__` : nom et répertoire du script
- `__FUNCTION__` : le nom de la fonction
- `__CLASS__` : le nom de la classe (avec l'espace de nom)
- `__METHOD__` : le nom de la méthode
- `__NAMESPACE__` : l'espace de nom actuel
- `__TRAIT__` : le nom du trait actuel (voir la partie du cours concernée)

# Les opérateurs classiques

- Affectation : '=', '+=', '-=', '\*=', '/=', '().''
- Arithmétiques : '+', '-', etc.
  - `+$a` : conversion de `$a` vers `int` ou `float`
  - `$a ** $b` correspond à `$a$b`
  - La division retourne un flottant sauf si les 2 opérandes sont entiers (ou convertis en entier)
  - Les opérandes du modulo `%` sont convertis en entiers (suppression de la partie décimale)
  - Le signe du résultat du modulo est le même que le premier opérande
- Incrémentation : `++$a` (pré-incrémentation) `$a++` (post-incrémentation)

## Les opérateurs logiques

- `$a and $b` (ET) : TRUE si `$a` et `$b` valent TRUE
- `$a or $b` (OU) : TRUE si `$a` ou `$b` valent TRUE
- `$a xor $b` (XOR) : TRUE si `$a` ou `$b` valent TRUE mais pas en même temps
- `!$a` (NON) : TRUE si `$a` n'est pas TRUE
- `$a && $b` (ET) : TRUE si `$a` et `$b` valent TRUE ; plus prioritaire que `and`
- `$a || $b` (OU) : TRUE si `$a` ou `$b` valent TRUE ; plus prioritaire que `or`

## Les opérateurs sur les bits

- $\$a \& \$b$  (ET) : les bits positionnés à 1 dans  $\$a$  et dans  $\$b$  sont positionnés à 1
- $\$a | \$b$  (OU) : les bits positionnés à 1 dans  $\$a$  ou dans  $\$b$  sont positionnés à 1
- $\$a \wedge \$b$  (XOR) : les bits positionnés à 1 dans  $\$a$  ou dans  $\$b$  mais pas les deux sont positionnés à 1
- $\sim \$a$  (NON) : les bits qui sont positionnés à 1 dans  $\$a$  sont positionnés à 0 et vice-versa
- $\$a \ll \$b$  : les bits de  $\$a$  sont décalés de  $\$b$  fois vers la gauche

# Les opérateurs de comparaison

- Comparaison large == : comparaison des valeurs après transtypage
- Comparaison stricte === : valeurs et types identiques
- Idem avec != (ou <>) et !==
- Classiques : <, >, <= et >=
- Combiné <=> : un entier inférieur, égal ou supérieur à 0 lorsque le premier opérande est inférieur, égal ou supérieur au deuxième



## D'autres opérateurs

- Opérateur ternaire : `expr1 ? expr2 : expr3`  
 ⇨ Si `expr` est TRUE, exécution de `expr2` sinon exécution de `expr3`
- ?? (fusion null) : `expr1 ?? expr2`  
 ⇨ Si `expr` est NULL, retourne `expr2` sinon `expr1`

### Avec fusion null

```
<?php
$action = $_POST['action'] ?? '
    default';
```

### Equivalent

```
<?php
if(isset($_POST['action']))
    $action = $_POST['action'];
else
    $action = 'default';
```

# Les conditionnelles (1/2) : les différentes formes

## Paramètres

cond, cond1 et cond2 sont des expressions booléennes

## Si...

```
if(cond) {  
    // Exécuté si cond vraie  
}
```

## Si... sinon...

```
if(cond) {  
    // Exécuté si cond vraie  
}  
else {  
    // Exécuté si cond fausse  
}
```

## Écriture condensée

```
if(cond1) {  
    // Exécuté si cond1 vraie  
}  
elseif(cond2) {  
    // Exécuté si cond1 fausse  
    // Exécuté si cond2 vraie  
}  
else {  
    // Exécuté si cond1 fausse  
    // Exécuté si cond2 fausse  
}
```

## Les conditionnelles (2/2) : exemple

### Exemple : code PHP

```
<?php
define("VALEUR", 12);
$a = VALEUR;
if($a > VALEUR)
    echo "Sup. à ".VALEUR;
elseif($a < VALEUR)
    echo "Inf. à ".VALEUR;
else
    echo "Egal à ".VALEUR;
?>
```

### Exemple : exécution

Egal à 12

# Le cas parmi : exemple

## Exemple : code PHP

```
<?php
$a = 2;
switch($a) {
    case 1:
        echo "un";
        break;
    case 2:
        echo "deux";
        break;
    default:
        echo "rien";
}
?>
```

## Exemple : exécution

deux

# La boucle `for` : présentation

- Permet de répéter plusieurs fois un bloc d'instructions
- Syntaxe générale :
  - ↪ `for($i = 0; $i < 10; $i++) { ... }`
  - ↪ Si une seule instruction, accolades non nécessaires
- La variable de contrôle (ici `$i`) :
  - 1 Est initialisée lors de l'entrée dans la boucle (ici à 0)
  - 2 Puis incrémentée à chaque tour de boucle (ici de 1 en 1)
    - ↪ `$i++` est équivalent à `$i = $i + 1`
  - 3 Tant que la condition est vérifiée (ici tant que *i* inférieur à 10)

# La boucle for : exemple

## Exemple : code PHP

```
<?php
echo "<ul>";
for($i = 0; $i < 3; $i++)
    echo "<li>_Point_{$i}_</li>";
echo "</ul>";
?>
```

## Sortie

- Point 0
- Point 1
- Point 2

# La boucle `while` : présentation

- Permet de répéter un bloc d'instructions tant que la condition est vérifiée (est vraie)
- Syntaxe générale :  
`↪ while(condition) { ... }`
- Pas de variable de boucle nécessaire
- Condition : expression booléenne

# La boucle while : exemple

## Exemple : code PHP

```
<?php
echo "<ul>";
$i = 10;
while($i > 0) {
    echo "<li>_Point_{$i}_</li>";
    $i = $i - 3;
}
echo "</ul>";
?>
```

## Sortie

- Point 10
- Point 7
- Point 4
- Point 1



# La boucle `for each`

- Utilisée uniquement pour les tableaux (et les objets)
- Permet de parcourir les éléments d'un tableau

## Exemple : affichage d'un tableau

```
<?php
$tab = [1, 2, 3, 4];

foreach($tab as $valeur) {
    echo $valeur." ";
}
?>
```

### Sortie

1 ; 2 ; 3 ; 4 ;



# La boucle for each

- Utilisée uniquement pour les tableaux (et les objets)
- Permet de parcourir les éléments d'un tableau

## Exemple : récupération des clés

```
<?php
$tab = [1, 2, 3, 4];

foreach($tab as $cle => $val) {
    echo "$cle_=>_<val;_";
}??>
```

## Sortie

0 = 1; 1 = 2; 2 = 3; 3 = 4;



# La boucle for each

- Utilisée uniquement pour les tableaux (et les objets)
- Permet de parcourir les éléments d'un tableau

## Exemple : modification d'un tableau

```
<?php
$tab = [1, 2];

foreach($tab as &$valeur) {
    $valeur = $valeur * 2;
}

print_r($tab);
?>
```

### Sortie

```
Array ( [0] => 2 [1] => 4 )
```



# Écriture d'une fonction

- Fonctions définies dans le script en cours ou dans un autre fichier
- Signature d'une fonction :
  - Mot-clé `function` suivi du nom de la fonction
  - Paramètres de la fonction (séparés par une virgule, entre parenthèses)
- Le corps de la fonction : bloc de code quelconque
  - ↪ Accolades obligatoires
- Peut retourner une valeur avec le mot-clé `return`
  - ↪ L'exécution de la fonction s'arrête immédiatement après
  - ↪ Instructions suivantes ignorées (boucles interrompues, *etc.*)
  - ↪ En algorithmique, un seul appel à `return`
- Si pas de `return` : procédure

# Exemple d'une fonction et d'une procédure

## Exemple de fonction

```
function somme($a, $b) {  
    return $a + $b;  
}  
  
$resultat = somme(1, 2);  
echo "Résultat : " . $resultat . " ;
```

## Résultat

Le résultat est : 3.

## Exemple de procédure

```
function affiche($a) {  
    echo "<strong>$a".  
        "</strong><br/>";  
}  
  
$variable = 1;  
affiche(2);  
affiche($variable);
```

## Résultat (ici en HTML)

```
<strong>2</strong><br/>  
<strong>1</strong><br/>
```

## Valeur par défaut

- Permet de spécifier des valeurs par défaut  
↪ Évite de devoir spécifier tous les paramètres
- Solution en PHP pour faire de la surcharge
- Remarque : il ne peut y avoir un paramètre avec une valeur par défaut, suivi par un paramètre sans valeur par défaut

### Exemple

```
function somme($a, $b = 0) {  
    return $a + $b;  
}
```

```
$resultat = somme(1);  
echo "Le_résultat_est_:_{$resultat}";
```

### Résultat

Le résultat est : 1.

# Passage par valeur *versus* par adresse

- Valeur passée aux paramètres  
⇨ Pas de répercussion dans le script appelant
- C'est vrai pour les types primitifs, tableaux  
⇨ Pas pour les objets...

## Exemple

```
function modifie1($a) {  
    $a = $a + 1;  
}  
$a = 0;  
modifie1($a);  
echo "Le_résultat_est_:_$a.";
```

## Résultat

Le résultat est : 0.



# Passage par valeur *versus* par adresse

- Référence d'une variable passée en paramètre :  
↪ Variable du script appelant modifiée
- Nécessite de passer une variable en paramètre

## Exemple

```
function modifie2(&$a) {  
    $a = $a + 1;  
}  
$a = 0;  
modifie2($a);  
echo "Le_résultat_est_:_{$a}.";
```

## Résultat

Le résultat est : 1.





# Typage des paramètres

- Possible de typer les paramètres (obligatoire en INFO0303)
- Oblige l'utilisateur à utiliser les bons types
- Conversion implicite si nécessaire (et possible)...

## Exemple

```
function affiche(array $tab, string $sep = ";_") {  
    for($i = 0; $i < count($tab) - 1; $i++)  
        echo $tab[$i].$sep;  
    echo $tab[$i]."<br/>";  
}  
$t = [1, 2, 3, 4];  
affiche($t);
```

## Sortie

1; 2; 3; 4

# Typage strict

- Possible d'activer le typage strict
- Génération d'une erreur en cas de valeur de mauvais type passée en paramètre
- Activation dans un script avec l'instruction `declare(strict_types=1)`  
 ↪ S'applique uniquement aux appels réalisés dans ce script

## Exemple

```
function test1(float $a) : float {
    return $a+10;
}
echo test1(1). "<br/>";
echo test1("1"). "<br/>";
```

## Résultat

```
11
11
```



# Typage strict

- Possible d'activer le typage strict
- Génération d'une erreur en cas de valeur de mauvais type passée en paramètre
- Activation dans un script avec l'instruction `declare(strict_types=1)`  
 ↪ S'applique uniquement aux appels réalisés dans ce script

## Exemple

```
<?php
declare(strict_types=1);
function test1(float $a) : float {
    return $a+10;
}
echo test1(1). "<br/>";
echo test1("1"). "<br/>";
```

## Résultat

```
11
Fatal error: insults...
```



## Typage du retour (PHP 7)

- Spécifie le type de retour de la fonction
- Après les paramètres :
  - ↪ Utilisation de ":" suivi du type

### Exemple

```
function somme(int $a, int $b) : int {  
    return $a + $b;  
}  
$c = somme(3, 4);  
var_dump($c);
```

### Sortie

```
int 7
```

## Retour d'un tableau/objet ou null

- En cas d'erreur, par exemple, possible de retourner `null`  
↪ Retour par défaut (sans `return`)
- Définition du type de retour précédé de '?'

### Exemple

```
function tableau(int $a) : ?array {  
    $resultat = null;  
    if($a > 0) {  
        $resultat = [];  
        for($i = 0; $i < $a; $i++)  
            $resultat[] = $i;  
    }  
    return $resultat;  
}  
  
$t = tableau(3);  
var_dump($t);
```

# Retour de plusieurs valeurs

- Pour retourner plusieurs valeurs, usage des tableaux  
↪ Classique...
- Possible d'initialiser plusieurs variables à l'aide de la *déconstruction*
- Ancienne syntaxe : `list(...)` (toujours fonctionnelle)

## Exemple

```
function exemple() : array {  
    return [ 1, 2, 3];  
}  
[ $a, $b, $c ] = exemple();  
echo "{$a}_/_{$b}_/_{$c}_<br/>";
```

## Paramètres variables

- Support des fonctions à nombre de paramètres variable
- Usage de ... (comme en C)
- Valeurs passées dans la variable suivante sous forme d'un tableau

### Exemple

```
function somme($coeff, ...$nombres) : int {  
    $resultat = 0;  
    foreach($nombres as $n) {  
        $resultat += intval($n) * $coeff;  
    }  
    return $resultat;  
}  
echo somme(5, 1, "2", 3.5, 4);
```

# Fonctions variables

- PHP est capable d'évaluer une chaîne de caractères comme l'appel à une fonction
  - ↪ Ca marche également avec les appels de méthode d'une classe
- Exécution : nom de la variable suivi par ()
  - ↪ Possible d'avoir des paramètres

## Exemple

```
function test1() : void {  
    echo "Cool<br/>";  
}  
function test2(int $a) : int {  
    return $a*2;  
}  
$t1 = "test1";  
$t1();  
$t2 = "test2";  
echo $t2(12);
```



# Fonctions anonymes

- Permet de créer des fonctions sans spécifier de nom  
↪ Utilisée comme fonction de rappel (*callback*)
- Utile pour un usage unique

## Exemple

```
$chaine = "bonjour_tout_le_monde";  
echo preg_replace_callback('~_([a-z])~', function($match) {  
    return strtoupper($match[1]);  
}, $chaine);
```

## Résultat

bonjourToutLeMonde

# Variables globales versus locales

- Variable définie dans une fonction : par défaut locale
- Possible d'accéder à une variable définie dans le script appelant (contexte global) :  
    ↪ Mot-clé : `global`
- Possible de définir une variable globale dans une fonction
- La variable `$GLOBALS` contient les variables globales

## Exemple d'utilisation de `global` (1/2)

### Exemple

```
<?php
function fonction1() : void {
    if(isset($globale))
        echo "<p>_Valeur_=_$globale._</p>";
    else
        echo "<p>_Variable_inconnue._</p>";
    echo "<p>_La_valeur_est_:_" . $GLOBALS["globale"] . "._.</p>";
}

function fonction2() : void {
    global $globale;
    if(isset($globale))
        echo "<p>_Valeur_=_$globale._</p>";
    else
        echo "<p>_Variable_inconnue._</p>";
}

$globale = "valeur";
fonction1();
fonction2();
```

## Exemple d'utilisation de `global` (2/2)

### Exemple

```
<?php
function fonction1() : void {
    global $toto;
    $toto = "truc";
}

function fonction2() : void {
    global $toto;

    echo "<p>_Valeur_de_\"$toto\"=_$toto._</p>";
}

fonction1();
fonction2();
```

# Variables superglobales

- Rappel : variables globales accessibles depuis tous les contextes  
↪ Pas besoin d'utiliser `global`
- Exemples : `$_GET`, `$_POST`, `$_SESSION`, `$_COOKIE`, ...
- Variable `$_REQUEST` : contient par défaut `$_GET`, `$_POST` et `$_COOKIE`
- Variable `$_SERVER` : variables concernant le serveur et l'exécution