# PHP

# PHP

C’est quoi ? Pourquoi ? Comment ?

# PHP

C’est quoi ?

# Le PHP, c’est quoi ?

- PHP :c’est l’acronyme de **P**HP **H**ypertext **P**rocessor
    - PHP est un nom récursive.
- C’est un langage **interprété** par **le serveur**.

# PHP

Pourquoi ?

# Le PHP, pourquoi ?

- PHP est gratuit et open source
- C’est un langage de programmation très populaire
- On va utiliser PHP pour :
  - Générer des document dynamique (HTML, CSS, PDF, XML, Json, Image, Zip, ...)
  - Créer,  lire, modifier et supprimer des fichiers sur le serveur
  - Sécuriser les accès à un service (Identifiant / Mot de passe)
  - Créer,  lire, modifier et supprimer des données dans une base de données

# PHP

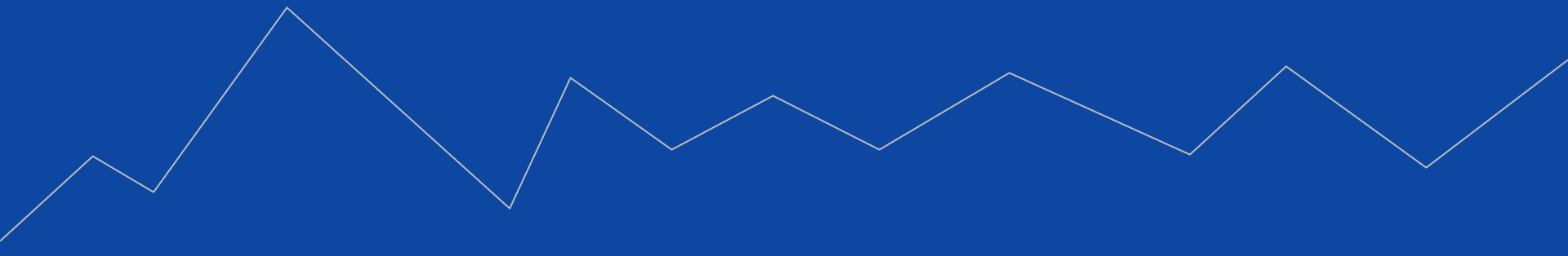
Comment ?

# Le PHP, comment ?

- PHP peut être exécuté sur Windows, Unix, Mac OS
- Pour faire du PHP vous avez besoin :
  - D’un serveur web (Apache, Nginx)
  - L’extension PHP
  - Une base de données (MySQL) pour exploiter tout l'intérêt de PHP.
- Ces trois choses se retrouve dans le package XAMPP, il en existe d’autres :
  - Wamp
  - EasyPHP

# PHP

La base



# PHP

La syntaxe

# PHP, la syntaxe

- Un script PHP s'écrit entre les balises **<?php** et **?>**
- Un fichier PHP doit avoir l'extension "**.php**"
- Généralement les fichiers PHP contiennent du HTML et / ou des scripts PHP
- Une instruction PHP se termine par un point-virgule **(;)**.

```php
<?php echo "Hello World!"; ?>
```

# PHP, les commentaires

- Un commentaire est une ligne qui ne sera ni lue, ni interprété par le programme
- C’est juste une information qui renseignera la personne qui lira le code

```
// Ceci est un commentaire

# Ceci est un autre commentaire

/*
ceci est un commentaire
multiligne.
*/
```

# PHP, La casse

- Les mots clés des instructions PHP ne sont pas sensible à la casse :
  - if
  - else
  - while
  - for
  - echo
  - noms de classes
  - noms de fonctions
  - noms de fonctions utilisateurs

```php
<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>
```

- Les noms des variables sont sensible à la casse.

# PHP

## Variables

# PHP, variables

- Une variable se déclare avec le symbole **$** suivi de son nom.
- Le nom des variables doit **commencer** par une **lettre** ou un **underscore**.
- Le nom des variables peut contenir des **caractères alpha-numérique** ou des **underscores** (a-z, 0-9, _).
- Les variables sont créées lors de la premières affectation.

# PHP, variables

- Les noms des variables sont sensibles à la casse.
  - `$var` et `$VAR` sont deux variables différentes.
- Une chaîne de caractères affectée à une variable devra être mise entre quotes (simples ou doubles).

```php
<?php
$a = "Hello world!";
$x = 42;
$y = 21.5;
?>
```

# PHP, afficher des variables

```php
<?php
$a = "world";
echo "Hello " . $a . "!";   // Exemple 1 : Hello world!
echo "Hello $a!";           // Exemple 2 : Hello world!
echo 'Hello $a!';           // Exemple 3 : Hello $a!
?>
```

- Les exemples 1 et 2 produisent le même résultat

- Les variables passées entre simple quotes ne seront pas interprétés

# PHP, portée des variables

- Les variables peuvent être déclarées n’importe ou dans le programme.
- La portée d’une variable dépend de la partie du programme dans laquelle elle est déclarée.
- PHP possède trois types de portée.
  - local
  - global
  - static

# PHP, portée des variables (**globale** et locale)

- Une variable déclarée **hors d'une fonction** aura une **portée globale** mais ne sera **pas accessible dans une fonction**.

```php
<?php
$x = 5; // $x est une variable globale

function myTest() {
    // Utilisé dans une fonction, $x générera une erreur
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>
```

# PHP, portée des variables (globale et **locale**)

- Une variable déclarée **dans une fonction** aura une **portée locale** et ne sera accessible **que dans la fonction** dans laquelle elle est déclarée.

```php
<?php
function myTest() {
      $x = 5; // $x est une variable local
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();

// $x n’est pas accessible à l’extérieur de la fonction
echo "<p>Variable x outside function is: $x</p>";
?>
```

# PHP, portée des variables (globale)

- Une variable déclarée **hors d'une fonction** pourra être appelé dans une fonction grâce au mot clé global.

```php
<?php
$x = 5;
$y = 10;

function myTest() {
    global $x, $y;
    $y = $x + $y;
}

myTest();
echo $y; // en sortie, $y vaut 15
?>
```

# PHP, portée des variables (static)

- Une variable déclarée dans une fonction avec la portée `static` ne sera pas détruite à la fin de la fonction.

```php
<?php
function myTest() {
    static $x = 0;
    echo $x;
     $x++;
}

myTest();
myTest();
myTest();
?>
```

# PHP

## Echo & Print

# PHP, echo & print

- echo et print permettent de créer une sortie PHP.
- echo n’a pas de valeur de retour.
- print retourne 1
- echo peut prendre plusieurs paramètres.
- print n’accepte qu’un seul paramètre.
- echo est plus rapide

# PHP

## Les types de données

# PHP, les types de données

- `string` chaîne de caractères : `“Hello world”;`
- `integer` les entiers : `42;`
- `float / double` nombre à virgule : `10.5;`
- `boolean` booléen : true / false;
- `null` valeur nulle : `null;`

# PHP, les types de données

- `resource` les ressources ne sont pas des types de données mais un

  stockage de références à des fonctions PHP.

- `array` tableau : `array('green', 'red', 42);`

- `object` objet :
```
class myObject {
private $a;
private function myTest() {
}
}
```

# PHP

## Les fonctions de chaînes

# PHP, les fonctions de chaînes

- `strlen` retourne le nombre de caractères d'une chaîne
  - ```php
    strlen("Hello world!"); // retourne 12
    ```
- `str_word_count` retourne le nombre de mots d'une chaîne
  - ```php
    str_word_count("Hello world!"); // retourne 2
    ```
- `strrev` inverse une chaîne
  - ```php
    strrev("Hello world!"); // retourne !dlrow olleH
    ```
    ```

# PHP, les fonctions de chaînes

- `strpos` retourne l'index du premier caractère dans la chaîne
  - ```php
    strpos("Hello world!", "world"); // retourne 6
    ```
- `str_replace` remplace une chaîne
  - ```php
    str_replace("Hello", "Goodbye", "Hello world!"); // retourne Goodbye world!
    ```

# PHP

## Les constantes

# PHP, les constantes

- à l'inverse des variables, les constantes ne changent pas de valeur.
- On déclare une constante avec la fonction PHP `define();`
  - `define(name, value, case-insensitive);`
    - `name` Nom de la constante
    - `value` Valeur de la constante
    - `case-insensitive` Booléen (false par défaut) qui définit si le nom de la constante est insensible à la casse.

# PHP, les constantes

- Dans les bonnes pratiques, on écrit le nom des constantes en majuscule et on laisse le drapeau `case-insensitive` à `false`.
- La portée d'une constante est globale, elle sera accessible partout dans le programme.
- Le nom d'une constante doit être unique.

# PHP

## Les opérateurs

# PHP, les opérateurs Arithmétique

- `+` Addition
- `-` Soustraction
- `*` Multiplication
- `/` Division
- `%` Modulo (retourne le reste d'une division)
- `++` Incrémentation
- `--` Décrémentation
- `**` Exposant

# PHP, les opérateurs d'Assignation

L'opérateur d'assignation (=) assigne une valeur à une variable.

- `=`
- `+=` `x += y;` équivaut à `x = x + y;`
- `-=` `x -= y;` équivaut à `x = x - y;`
- `*=` `x *= y;` équivaut à `x = x * y;`
- `/=` `x /= y;` équivaut à `x = x / y;`
- `%=` `x %= y;` équivaut à `x = x % y;`

# PHP, les opérateurs de chaîne

- Seul le `.` peut être utilisé comme opérateur de chaîne.

```
$str1 = “Hello”;

$str2 = “World”;

$str3 = $str1 . “ “ . $str2; // Hello World

$str4 = “Goodbye”;

    str4 .= “ “.str2; // Goodbye World
```

# PHP, les opérateurs de comparaison

- == égale à *(valeur)*
- === strictement égale à *(valeur ET type)*
- != différent *(valeur)*
- !== strictement différent *(valeur OU type)*
- > supérieur à
- < inférieur à
- >= supérieur ou égale à
- <= inférieur à égale à
- ? opérateur ternaire

# PHP, les opérateurs logiques

- `&&` `and` ET logique
- `||` `or` OU logique
- `xor` OU Exclusif
- `!` NON logique

# PHP, les opérateurs d’incrémentation/décrémentation

- `&&` `and` ET logique
- `||` `or` OU logique
- `xor` OU Exclusif
- `!` NON logique

# PHP

Les conditions if...else...elseif

# PHP, les conditions

- Une condition est une instruction
- Elle permet de tester si une condition est réalisée ou non
- Le test de la condition retourne `true` ou `false`.

# Les conditions : if (Syntaxe)

```
if ( condition à analyser ) {

    // Code exécuté si la condition est vraie.

}
```

# Les conditions : if … else (Syntaxe)

```
if ( condition à analyser ) {

    // Code exécuté si la condition est vraie.

}

else {

    // Code exécuté si la condition est fausse.

}
```

# Les conditions : if … elseif … else (Syntaxe)

```
if ( première condition à analyser ) {

    // Code exécuté si la première condition est vraie.
}

elseif ( seconde condition à analyser ) {

    // Code exécuté si la seconde condition est vraie.
}

else {

    // Code exécuté si aucune des conditions n'est vraie.
}
```

# PHP

## Le Switch

# PHP, l’instruction switch

- Le commutateur `switch` et équivalent à la condition `if … elseif … else`.
- il est :
  - plus lisible
  - plus rapide à écrire
  - plus rapide à exécuter

# PHP, l'instruction switch (syntaxe)

```
switch ( condition à tester ) {
    case Valeur1:
        /* code à exécuter la condition == Valeur1 */
        break;
    case Valeur2:
        /* code à exécuter la condition == Valeur2 */
        break;
    default:
        /* code à exécuter la condition != Valeur1 && condition !=
    Valeur2 */
    break;
}
```

# PHP

Les boucles

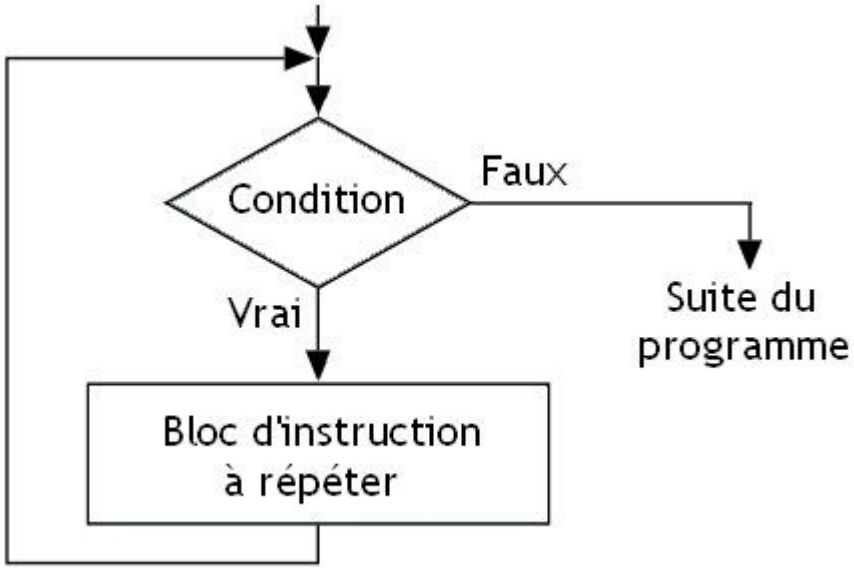
# PHP, les boucles

- Un boucle répète une action tant que la condition est satisfaite.
- Dans quels cas utiliser des boucles :
  - Pour parcourir des tableaux.
  - Pour répéter des tâches.

```
alert(1);
alert(2);
alert(3);
alert(4);
alert(5);
   ...
```

# PHP, les boucles : While

- Exécute la boucle tant que la condition est vérifiée.

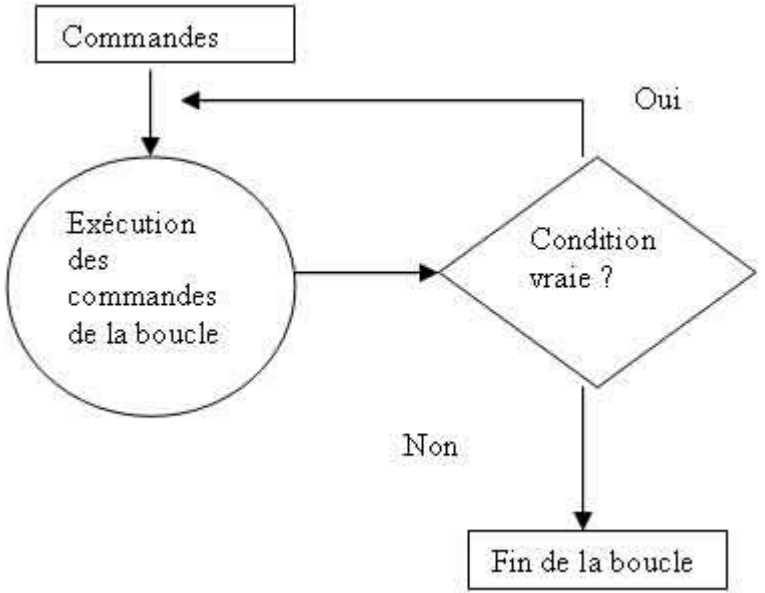


# PHP, les boucles : While (Syntaxe)

```
while( condition de bouclage )
{
   // instructions
}
```

# PHP, les boucles : Do … While

- Do … While va d'abord exécuter un bloc d'instruction, avant de tester la première condition de bouclage.

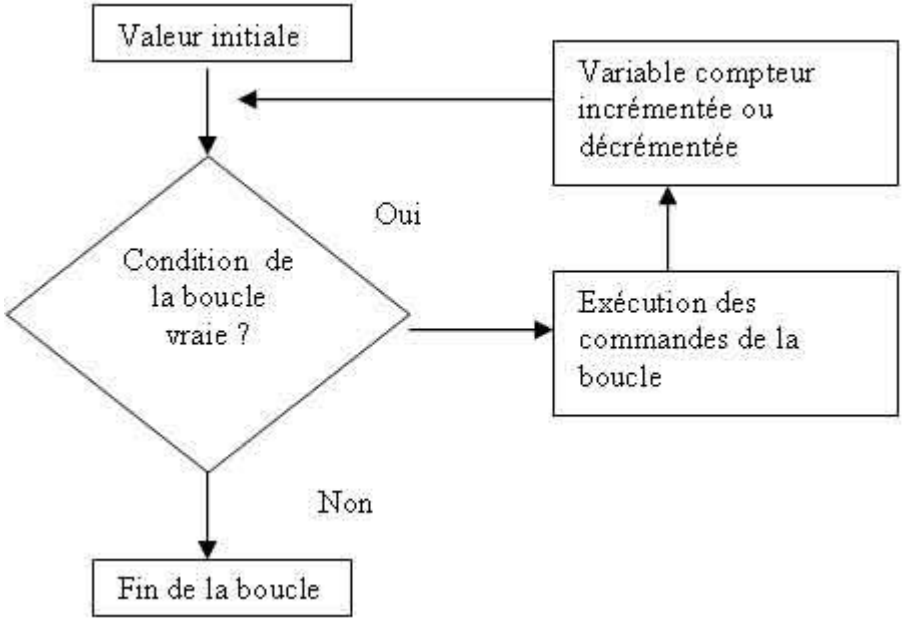


# PHP, les boucles : Do … While (Syntaxe)

```
do {
    // instructions;
} while( condition de bouclage )
```

# PHP, les boucles : For

- Boucle avec incrémentation d'une valeur de bouclage.
- à besoin de 3 paramètres :
  - initialisation
  - condition
  - incrémentation



# PHP, les boucles : For (Syntaxe)

```
for( initialisation ; condition ; incrementation )
{
    // instructions;
}
```

# PHP, les boucles : Foreach

- Parcourir un tableau

```php
foreach ( tableau as index => valeur ) {

    // instruction;

}
```

# PHP

## Les fonctions utilisateur

# PHP, les fonctions

- Une fonction est une procédure, un ensemble d’instructions.
- on distingue deux types de fonctions :
  - les fonctions native, implanté à PHP.
  - les fonctions utilisateur, créée par l’utilisateur de PHP *(les développeurs)*

# PHP, les fonctions utilisateur

- Il est nécessaire de définir une fonction avant de l'utiliser.
- On définit une fonction avec
  - le mot clé `function`
  - suivi de son nom
  - une liste d'argument entre ( )
  - une série d'instructions entre { }

# PHP, les fonctions utilisateur : Déclaration

```
function maFonction ( argument1, argument2 = “default”, … ) {
    instruction1;
    instruction2;
}
```

# PHP, les fonctions : valeur par défaut

- Les arguments peuvent avoir une valeur par défaut.

```
function multiplier( a, b=1 ) {
    return a * b;
}

multiplier(5); // 5
multiplier(5,2); // 10
```

# PHP, les fonctions : utilisation

- l'**utilisation d'une fonction** s'appel aussi **appel d'une fonction**.
- Appel de la fonction, sans passage de paramètre :
  - maFonction();
- Appel de la fonction, avec passage de paramètres :
  - maFonction( "Trillian" );

# PHP, les fonctions : valeur de retour

- Une fonction peut retourner une valeur avec le mot clé `return`.

```
function cube( nombre ) {
    return nombre * nombre * nombre;
}
cube(2); // 8
```

# PHP

## Les tableaux

# PHP, les tableaux

- On utilise la fonction `array()` pour créer un tableau.
  - Depuis PHP7 les crochets suffisent pour déclarer un tableau : `[]`.
- Il y a trois types de tableaux :
  - Numérique : avec index numérique
  - Associatif : avec un nom comme clé
  - Multidimensionnel : un tableau contenant des tableaux

# PHP, les tableaux numérique

```php
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

# PHP, les tableaux associatif

```php
<?php

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

echo "Peter is " . $age['Peter'] . " years old.";

?>
```

# PHP, fonctions de tri des tableaux

- `sort()` tri un tableau dans l'ordre ascendant.
- `rsort()` tri un tableau dans l'ordre descendant.
- `asort()` tri un tableau associatif dans l'ordre ascendant en se basant sur les valeurs.
- `ksort()` tri un tableau associatif dans l'ordre ascendant en se basant sur les clés.
- `arsort()` tri un tableau associatif dans l'ordre descendant en se basant sur les valeurs.
- `krsort()` tri un tableau associatif dans l'ordre descendant en se basant sur les clés.

# PHP

## Les superglobals

# PHP, les superglobales

- `$_SERVER` contient des informations sur l'entête et les emplacements de script.
- `$_REQUEST` collecte des données après la soumission d'un formulaire.
- `$_POST` collecte des données après la soumission d'un formulaire en méthode POST.
- `$_GET` collecte des données après la soumission d'un formulaire en méthode GET.
- `$_COOKIE` contient les informations des cookies.
- `$_SESSION` contient les informations de session.