# Développement Web Back-end

## Internet et les pages web

- HTML : conception de pages destinées à être publiées sur Internet
- Page html : contient le texte à afficher et des instructions de mise en page
- HTML est un langage de description de page
- Des sites de plus en plus riches en informations
  - Nécessité croissante d'améliorer le contenu de sites
  - Mises à jour manuelles trop complexes
    - Pourquoi ne pas automatiser les mises à jour ?

## Pages web statiques : fonctionnement

Leurs contenus ne changent ni en fonction du demandeur ni en fonction d'autres paramètres éventuellement inclus dans la requête adressée au serveur.  Toujours le même résultat.

- Rôle du serveur : localiser le fichier correspondant au document demandé et répond au navigateur en lui envoyant le contenu de ce fichier

## Pages web statiques : limites

Besoin de réponses spécifiques : passage de pages statiques à pages dynamiques

## Les langages de script-serveur : Définition

- Un langage de script -serveur est :
  - un programme stocké sur un serveur et exécuté par celui-ci, qui passe en revue les lignes d'un fichier source pour en modifier une partie du contenu, avant de renvoyer à l'appelant ( un navigateur par exemple) le résultat du traitement.
- La tâche d'interprétation des ordres à exécuter est déléguée à un composant, souvent appelé moteur, qui est :
  - **installé sur le serveur**, qui est doté d'une API et d'un fonctionnement identique quel que soit la plate-forme utilisée pour gérer le serveur.

- **Pages web dynamiques côté serveur ou côté client**

  - **Langage côté client :** traité par la machine qui accueille le logiciel de navigation.

➢ **Traitement** : Exécuté par la machine de l'utilisateur via le navigateur.

➢ **Variabilité** : Les résultats peuvent différer selon :

  ➢ Le navigateur (ex : Netscape, Internet Explorer, firefox…).

  ➢ La plateforme (ex : PC, Mac).

➢ **Tests** : Nécessite des tests approfondis pour garantir la compatibilité.

➢ **Visibilité du code** : Le code source est accessible et ne peut pas être masqué.

➢ **Indépendance** : Fonctionne indépendamment du serveur et de l'hébergement.

- **Pages web dynamiques côté serveur ou côté client**

  - Langage côté serveur : le travail d'interprétation du programme est réalisé par le serveur
    - Sont indépendants de la machine et du logiciel de navigation utilisés pour la consultation.
    - Sont compatibles avec tous les navigateurs et toutes leurs versions.
    - Permettent de masquer les sources de ses programmes.
    - Nécessitent de recharger la page chaque fois que celle-ci est modifiée.
  - Pages côté serveur et côté client :
    - Script côté client pour des calculs et des traitement simples.
    - Scripts côté serveur pour des calculs, des traitements et des mises à jours plus conséquents.

- **Les langages de création de pages web dynamiques côté serveur**
  - **ASP**
    - Basé sur des scripts écrits en VBscript, Jscript ou Javascript.
    - Largement répandu
    - Facilité de mise en œuvre
    - Plusieurs outils de développement intégrés (Macromédia Ultradev, Microsoft Visual Interdev).
    - L'utilisation de pages ASP avec Microsoft Internet Information Services (IIS) est actuellement prise en charge dans toutes les versions prises en charge d'IIS.
      - La dernière version d'ASP.NET se veut officiellement compatible avec Mac et Linux afin d'ouvrir le développement web de la sphère Microsoft à de nouveaux cercles de développeurs.

- **Les langages de création de pages web dynamiques côté serveur**

  - **JSP**

    - Constitue la réponse de Sun(Oracle) aux ASP de Microsoft
    - Utilisation de Java
      - Au départ simple extension du langage Java
      - Est devenu un véritable langage de développement web
    - Possède une interface de qualité
    - Lenteur relative

- **Les langages de création de page web dynamiques côté serveur**
  - **PHP**
    - Connaît un succès toujours croissant sur le Web et se positionne comme un rival important pour ASP
    - L'environnement Linux est sa plateforme de prédilection
    - Combiné avec le serveur Web Apache et la base de données MySQL, PHP offre une solution particulièrement robuste, stable et efficace
    - Gratuité : Tous les logiciels sont issus du monde des logiciels libres (Open Source).

# Un peu d'histoire

## Histoire et Origine

- PHP : Hypertext PreProcessor
- Première version de PHP a été mis au point au début d'automne par Rasmus Lerdorf en 1994
  - Version appelée à l'époque Personal Home Pages
  - Pour conserver la trace des utilisateurs venant consulter son CV sur son site, grâce à l'accès à une base de données par l'intermédiaire de requêtes SQL
- La version 3.0 de PHP fut disponible le 6 juin 1998
- A la fin de l'année 1999, une version bêta de PHP, baptisée PHP4 est apparue
- En 2001 cinq millions de domaines utilisent PHP
  - trois fois plus que l'année 2000
- PHP 5.0 s'inspire désormais largement du modèle de Java.
- La saga PHP 7: Facebook a publié en 2011 HipHop Virtual Machine dit HHVM, une machine virtuelle permettant de pré-compiler le code PHP en bytecode à la manière de Java (JIT Compiler).

# PHP : C'est QUOI ?

- **Définition**
  - Un langage de scripts permettant la création d'applications Web
  - Indépendant de la plate-forme utilisée puisqu'il est exécuté côté serveur et non côté client.
  - La syntaxe du langage provient de celles du langage C, du Perl et de Java.
  - Ses principaux atouts sont:
    - La gratuité et la disponibilité du code source (PHP est distribué sous licence GNU GPL)
    - La simplicité d'écriture de scripts
    - La possibilité d'inclure le script PHP au sein d'une page HTML
    - La simplicité d'interfaçage avec des bases de données
    - L'intégration au sein de nombreux serveurs web (Apache, Microsoft IIS, ...)

- **Principe**

  - Les scripts PHP sont généralement intégrés dans le code d'un document HTML
  - L'intégration nécessite l'utilisation de balises
    - avec le style xml : <? ligne de code PHP ?>
    - Avec le style php: <?php ligne de code PHP ?>
    - avec le style JavaScript :

<script language=«php»> ligne de code PHP </script>

    - avec le style des ASP : <% ligne de code ASP %>

- **Forme d'une page PHP**
  - Intégration directe

```
< ?php
//ligne de code PHP
?>
<html>
<head> <title> Mon script PHP </title> </head>
<body>
//ligne de code HTML
< ?php
//ligne de code PHP
?>
//ligne de code HTML
….
</body> </html>
```

- Forme d'une page PHP
  - Inclure un fichier PHP dans un fichier PHP : include()

Fichier Prinipal

```
<html>
<head>
<title> Fichier d'appel </title>
</head>
<body>
<?php
$salut = " BONJOUR" ;
include "information.inc" ;
?>
</body>
</html>
```

Fichier à inclure : information.inc

```
<?php
$chaine=$salut. " , C'est PHP "  ;
echo "  <table border=3>
 <tr> <td width = 100% >
  <h2> $chaine</h2>
  </td> </tr></table> ";
?>
```

Sécuriser votre serveur.
Avec Apache, vous pourrez prévenir l'affichage en clair du contenu d'une page .inc en ajoutant le code suivant au bas du fichier httpd.conf :

```
<Files *.inc>
  Order allow,deny
  Deny from all
</Files>
```

## Envoi du code HTML par PHP

- La fonction echo : echo Expression;
  - echo "Chaine de caracteres";
  - echo (1+2)*87;
- La fonction print : print(expression);
  - print("Chaine de caracteres");
  - print ((1+2)*87);
- La fonction printf : printf (chaîne formatée);
  - printf ("Le périmètre du cercle est %d",$Perimetre);

# Syntaxe de base : Introduction

- **Typologie**
  - Toute instruction se termine par un point-virgule
  - Sensible à la casse
    - Sauf par rapport aux fonctions
- **Les commentaires**
  - /* Voici un commentaire! */
  - // un commentaire sur une ligne

# Syntaxe de base : Les constantes

- **Les constantes**

  - **Define**("nom_constante", valeur_constante )
    - define (“ma_const”, “Vive PHP8”) ;
    - define (“an”, 2002) ;

  - Les constantes prédéfinies
    - NULL
    - _FILE_
    - _LINE_
    - PHP_VERSION
    - PHP_OS
    - TRUE et FALSE
    - E_ERROR

# Syntaxe de base : Les variables

- **Principe**
  - Commencent par le caractère $
- **Fonctions de vérifications de variables**
  - Doubleval(), empty(), gettype(), intval(),
  - is_array(), is_bool(), is_double(), is_float(), is_int(), is_integer, is_long(), is_object(), is_real(), is_numeric(), is_string()
  - Isset(), Determine si une variable est declare et est differente de **null**
  - settype(), Définit le type d'une variable
  - strval(), retourn string value d'une variable
  - unset() destroys the specified variables.
- **Affectation par valeur et par référence**
  - Affectation par valeur : $b=$a
  - Affectation par (référence) variable : $c = &$a

# Syntaxe de base : Les variables

- **Visibilité des variables**
  - Variable locale : Visible uniquement à l'intérieur d'un contexte d'utilisation
  - Variable globale : Visible dans tout le script
    - Utilisation de l'instruction global() dans des contextes locales

Exemple: $x = 10;

```
function portee1(){echo 'La valeur de $x globale est : ' .$x. '<br>'; }
function portee2(){$x = 5;echo 'La valeur de $x locale est : ' .$x. '<br>';  }
function portee3(){$y = 0;$y++; echo '$y contient la valeur : ' .$y. '<br>'; }
function portee4(){ $z = 1; }
function portee(){global $x;echo 'La valeur de $x globale est : ' .$x. '<br>';
   $x = $x + 5; //On ajoute 5 à la valeur de $x }
portee();echo '$x contient maintenant : ' .$x;
portee1();
portee2();
portee3();
portee4();
echo 'La variable locale $z contient : ' .$z;
```

# Syntaxe de base : Les variables

## Les variables dynamiques

- Permettent d'affecter un nom différent à une autre variable

```
$nom_variable = 'nom_var';

$$nom_variable = valeur; // équivaut à $nom_var = valeur;
```

- Les variables tableaux sont également capables de supporter les noms dynamiques

```php
<?php
    $tableau1 = array ('test', 'toto', 'titi');
    $tableau2 = array ('humpf', 'grmbl');

    $var = 'tableau1';
    $nb_elements = count (${$var});
    for ($i=0; $i<$nb_elements; $i++) {
        // on accède aux éléments du tableau $tableau1
        echo ${$var}[$i].'<br />';    // ce qui affichera donc test toto titi
    }
?>
```

PARTIE 1

# Syntaxe de base : Les variables

- **Variables prédéfinies**
  - Les variables d'environnement qui sont des variables contenant des informations qui concernent l'environnement d'exécution du script PHP. Cet environnement peut être soit le serveur soit le client. Pour manipuler les variables d'environnement on fait appel à la fonction getenv() ou à la variable superglobale $_SERVER qui peut également accéder aux variables serveur.
  - Les variables d'environnement dépendant du serveur

| Variable | Description |
|---|---|
| $_SERVER["SERVER_NAME"] | Le nom du serveur |
| $_SERVER["HTTP_HOST"] | Nom de domaine du serveur |
| $_SERVER["SERVER_ADDR"] | Adresse IP du serveur |
| $_SERVER["DOCUMENT_ROOT"] | Racine des documents Web sur le serveur |

# Syntaxe de base : Les variables

- **Variables prédéfinies**
  - Les variables d'environnement dépendant du client

| Variable | Description |
|---|---|
| $_SERVER["HTTP_HOST"] | Nom d'hôte de la machine du client (associée à l'adresse IP) |
| $_SERVER["HTTP_REFERER"] | URL de la page qui a appelé le script PHP |
| $_SERVER["HTTP_ACCEPT_LANGUAGE"] | Langue utilisée par le serveur (par défaut en-us) |
| $_SERVER["HTTP_ACCEPT"] | Renvoie l'en-tête Accept de la requête envoyé par le client |
| $_SERVER["CONTENT_TYPE"] | Cet en-tête indique le type de média (type MIME) des données envoyées dans le corps de la demande. |
| $_SERVER["REMOTE_ADDR"] | L'adresse IP du client appelant le script CGI |
| $_SERVER["PHP_SELF"] | Nom du script PHP |

# Syntaxe de base : Les variables

- **Variables prédéfinies**
  - Affichage des variables d'environnement
    - la fonction phpinfo()
      - `<?php phpinfo(); ?>`
      - `<?php echo phpinfo(constante);?>`

| | |
|---|---|
| INFO_CONFIGURATION | affiche les informations de configuration. |
| INFO_CREDITS | affiche les informations sur les auteurs du module PHP |
| INFO_ENVIRONMENT | affiche les variables d'environnement. |
| INFO_GENERAL | affiche les informations sur la version de PHP. |
| INFO_LICENSE | affiche la licence GNU Public |
| INFO_MODULES | affiche les informations sur les modules associés à PHP |
| INFO_VARIABLES | affiche les variables PHP prédéfinies. |

    - la fonction getenv() , $_SERVER[" mavariable"]
      - `<?php echo getenv("HTTP_USER_AGENT");?>`

# Syntaxe de base : Les types de données

- **Principe**
  - Pas besoin d'affecter un type à une variable avant de l'utiliser
    - La même variable peut changer de type en cours de script
    - Les variables issues de l'envoi des données d'un formulaire sont du type string
- **Les différents types de données**
  - Les entiers : le type Integer
  - Les flottants : le type Double
  - Les tableaux : le type array
  - Les chaînes de caractères : le type string
  - Les objets

# Syntaxe de base : Les types de données

- **Le transtypage**
  - La fonction settype() permet de convertir le type auquel appartient une variable

```
<?   $nbre=10;
Settype($nbre, "double");
Echo "la variable $nbre est de type ", gettype($nbre); ?>
```

  - Transtypage explicite : le cast
    - (int), (integer) ; (real), (double), (float); (string); (array); (object)

```
<?php   $var="100 FRF";
Echo "pour commencer, le type de la variable  $var, est " ,gettype($var);
$var =(double) $var;
Echo "<br> Après le cast, le type de la variable  $var est ", gettype($var);
Echo "<br> et a la valeur $var";    ?>
```

**Détermination du type de données**

- Gettype(), Is_long(), Is_double(), Is_string(), Is_array(), Is_object(), Is_bool()
- var_dump() une fonctions pour afficher la valeur et le type d'une variable.

# Syntaxe de base : Les chaînes de caractères

- **Principe**
- Peuvent être constituées de n'importe quel caractère alphanumérique et de ponctuation, y compris les caractères spéciaux

```
\tLa nouvelle monnaie unique, l' €uro, est enfin là...\n\r
```

- Une chaîne de caractères doit être toujours entourée par des guillemets simples (')ou doubles (")

" Ceci est une chaîne de caractères valide."

'Ceci est une chaîne de caractères valide.'

"Ceci est une chaîne de caractères invalide.'

- Des caractères spéciaux à insérer directement dans le texte, permettent de créer directement certains effets comme des césures de lignes

| Car | Code ASCII | Code hex | Description |
|---|---|---|---|
| " " | 32 | 0x20 | un espace simple. |
| \t | 9 | 0x09 | tabulation horizontale |
| \n | 13 | 0x0D | nouvelle ligne |
| \r | 10 | 0x0A | retour à chariot |
| \0 | 0 | 0x00 | caractère NUL |
| \v | 11 | 0x0B | tabulation verticale |

# Syntaxe de base : Les chaînes de caractères

- **Exemple :**

```php
<?php
    $nom = 'Diop';
    $prenom = 'Moussa';
    echo 'l\'Etudiant $prenom $nom'; //les variables ne seront pas remplacées
    echo "l'Etudiant $prenom $nom";
    echo 'l\'Etudiant Moussa DIOP';
    ?>
```

**Remarques**

Les variables ne sont pas remplacées par leurs valeurs dans une chaîne entre guillemets simples.

Pour utiliser des guillemets simples à l'intérieur d'une chaîne entre guillemets simples il faut la faire précéder du caractère anti-slash (\).

# Syntaxe de base : Les chaînes de caractères

- **Quelques fonctions de manipulation**

chaîne_result = addSlashes(chaîne); ajoute un anti-slash devant tous les caractères spéciaux.

```
$str = "O'Reilly?"; echo addslashes($str);// resultat O\'Reilly?
```

chaîne_result = chop(chaîne); supprime les espaces blancs en fin de chaîne.

caractère = chr(nombre); retourne un caractère en mode ASCII

$tableau = explode(délimiteur, chaîne); scinde une chaîne en fragments à l'aide d'un délimiteur et retourne un tableau.

```
$pizza  = "piece1 piece2 piece3 piece4 piece5 piece6";
$pieces = explode(" ", $pizza);
echo $pieces[0]; // piece1
echo $pieces[1]; // piece2
```

# Syntaxe de base : Les chaînes de caractères

- **Quelques fonctions de manipulation**

chaîne_result = crypt(string $string, string $salt)

crypt() retournera une chaîne haché utilisant l'algorithme standard d'Unix basé sur DES, ou un algorithme alternatif.

```
echo 'Standard DES: ',
    crypt('rasmuslerdorf', 'rl'), "<br>";
echo 'Extended DES: ',
    crypt('rasmuslerdorf', '_J9..rasm'), "<br>";
echo 'MD5:          ',
    crypt('rasmuslerdorf', '$1$rasmusle$'), "<br>";
echo 'Blowfish:     ',
    crypt('rasmuslerdorf', '$2a$07$usesomesillystringforsalt$'), "<br>";
echo 'SHA-256:      ',
    crypt('rasmuslerdorf', '$5$rounds=5000$usesomesillystringforsalt$'), "<br>";
echo 'SHA-512:      ',
    crypt('rasmuslerdorf', '$6$rounds=5000$usesomesillystringforsalt$'), "<br>";
```

## Password_hash

La fonction crypt() est considérée comme obsolète pour le hachage de mots de passe.

Il est recommandé d'utiliser des fonctions plus modernes et sécurisées comme **password_hash() et password_verify()**

```php
$password = "monMotDePasse";
$hashedPassword = password_hash($password, PASSWORD_DEFAULT);

echo "Mot de passe haché : " . $hashedPassword;
```

. **Vérification du mot de passe** :

- Pour vérifier un mot de passe haché, utilisez `password_verify()` :

```php
if (password_verify($password, $hashedPassword)) {
    echo "Mot de passe valide !";
} else {
    echo "Mot de passe invalide.";
}
```

PARTIE 1

# Syntaxe de base : Les chaînes de caractères

- **Quelques fonctions de manipulation**

| Fonctions | Exemple | Résultat | Description |
|---|---|---|---|
| nl2br | nl2br("M \n D") | M <br /> D | insére un <br /> à chaque nouvelle ligne |
| strcmp | strcmp ("Moussa","Moise") | 12 | compare 2 chaînes de caractères. |
| strlen | strlen("Moussa") | 6 | retourne le nombre de caractère de la chaîne |
| strpos | strpos("Moussa",'s') | 3 | renvoie la position de la première occurrence d'un caractère |
| strrev | strrev("Moussa") | assuoM | inverse une chaîne |
| strtolower | strolower("MoussaDiop") | moussadiop | convertit les majuscules en miniscules |
| strtoupper | stroupper("Moussa") | MOUSSA | convertit les miniscules en majuscules |
| trim | trim(" Moussa ") | Moussa | Supprime les espaces en début et fin de chaîne |
| echo, print | echo ("Bonjour") | 1.35 | Afficher une chaîne de caractères |
| ucfirst | ucfirst("moussa") | Moussa | Transforme le premier caractère en majuscule |

- https://www.php.net/manual/fr/ref.strings.php

- **Les opérateurs**

  - les opérateurs de calcul
  - les opérateurs d'assignation
  - les opérateurs d'incrémentation
  - les opérateurs de comparaison
  - les opérateurs logiques
  - les opérateurs bit-à-dit

# Syntaxe de base : Les opérateurs

- **Les opérateurs de calcul**

**Les opérateurs arithmétiques permettent de réaliser des opérations mathématiques sur les variables. Ce sont naturellement toutes les opérations conventionnelles telles que l'addition, la multiplication, la soustraction ou la division.**

| Opérateur | Dénomination | Effet | Exemple | Résultat |
|---|---|---|---|---|
| + | opérateur d'addition | Ajoute deux valeurs | $x+3 | 10 |
| - | opérateur de soustraction | Soustrait deux valeurs | $x-3 | 4 |
| * | opérateur de multiplication | Multiplie deux valeurs | $x*3 | 21 |
| / | plus: opérateur de division | Divise deux valeurs | $x/3 | 2.3333333 |
| = | opérateur d'affectation | Affecte une valeur à une variable | $x=3 | Met la valeur 3 dans la variable $x |

# Syntaxe de base : Les opérateurs

- **Les opérateurs d'assignation**

| Opérateur | Effet |
|---|---|
| += | addition deux valeurs et stocke le résultat dans la variable (à gauche) |
| -= | soustrait deux valeurs et stocke le résultat dans la variable |
| *= | multiplie deux valeurs et stocke le résultat dans la variable |
| /= | divise deux valeurs et stocke le résultat dans la variable |
| %= | donne le reste de la division deux valeurs et stocke le résultat dans la variable |
| \|= | Effectue un OU logique entre deux valeurs et stocke le résultat dans la variable |
| ^= | Effectue un OU exclusif entre deux valeurs et stocke le résultat dans la variable |
| &= | Effectue un Et logique entre deux valeurs et stocke le résultat dans la variable |
| .= | Concatène deux chaînes et stocke le résultat dans la variable |

Exemple:

```
$a = 3;

$a += 5; // affecte la valeur 8 à la variable $a correspond à l'instruction '$a = $a + 5';
```

# Syntaxe de base : Les opérateurs

- Les opérateurs d'incrémentation

| Opérateur | Opération | Exemple | Résultat |
|---|---|---|---|
| ++ | Pré-Incrémentation | ++$a | Incrémente $a, puis retourne $a |
| ++ | Post-Incrémentation | $a++ | Retourne $a, puis incrémente $a |
| -- | Pré-Décrémentation | --$a | Décrémente $a, puis retourne $a |
| -- | Post-Décrémentation | $a-- | Retourne $a, puis décrémente $a |

Les opérateurs d'incrémentation / décrémentation

```php
<?php
echo '<h3>Post-incrémentation</h3>';
$a = 5;
echo "Devrait valoir  5: " . $a++ . "<br>";
echo "Devrait valoir  6: " . $a . "<br>";
echo '<h3>Pre-incrémentation</h3>';
$a = 5;
echo "Devrait valoir  6: " . ++$a . "<br>";
echo "Devrait valoir  6: " . $a . "<br>";
?>
```

```php
<?php
echo '<h3>Post-décrémentation</h3>';
$a = 5;
echo "Devrait valoir  5: " . $a-- . "<br>";
echo "Devrait valoir  4: " . $a . "<br>";
echo '<h3>Pre-décrémentation</h3>';
$a = 5;
echo "Devrait valoir  4: " . --$a . "<br>";
echo "Devrait valoir  4: " . $a . "<br>";
?>
```

# Syntaxe de base : Les opérateurs

- **Les opérateurs de comparaison**

| Opérateur | Opération | Exemple | Résultat |
|---|---|---|---|
| == | Egalité en valeur | $a == $b | Vérifie que les valeurs de $a et $b sont identiques |
| === | Egalité en valeur et type | $a === $b | Vérifie que les valeurs et types de $a et $b sont identiques |
| != | Différence en valeur | $a != $b | Vérifie que les valeurs de $a et $b sont différentes |
| !== | Différence en valeur et type | $a !== $b | Vérifie que les valeurs et types de $a et $b sont différents |
| <> | Différence en valeur | $a <> $b | Alias de != |
| < | Infériorité stricte | $a < $b | Vérifie que $a est strictement inférieur $b |
| <= | Infériorité ou égalité | $a <= $b | Vérifie que $a est strictement inférieur ou égal à $b |
| > | Supériorité stricte | $a > $b | Vérifie que $a est strictement supérieur $b |
| >= | Supériorité ou égalité | $a >= $b | Vérifie que $a est strictement supérieur ou égal à $b |

Les opérateurs de comparaison

Ils sont essentiellement utilisés dans les structures conditionnelles (if, elseif, else, for, while...) afin de comparer des valeurs entre elles. Les tests renverront TRUE si la comparaison est juste ou bien FALSE si la comparaison est fausse.

# Syntaxe de base : Les opérateurs

- **Les opérateurs logiques**

| Opérateur | Opération | Exemple | Résultat |
|---|---|---|---|
| && | ET | $a && $b | TRUE si $a ET $b sont vrais |
| AND | ET | $a AND $b | Alias de && |
| \|\| | OU | $a \|\| $b | TRUE si $a OU $b est vrai |
| OR | OU | $a OR $b | Alias de \|\| |
| XOR | OU exclusif | $a XOR $b | TRUE si $a OU $b est vrai mais pas les deux |
| ! | NON | !$a | TRUE si $a est faux |

Les opérateurs logiques

La raison pour laquelle il existe deux types de "ET" et de "OU" est qu'ils ont des priorités différentes. Les opérateurs && et || ont une priorité plus élevée par rapport à leur semblable littéral respectif.

# Syntaxe de base : Les opérateurs

- **Les opérateurs bit-à-bit**

| Opérateur | Dénomination | Effet | Syntaxe | Résultat |
|---|---|---|---|---|
| & | ET bit-à-bit | Retourne 1 si les deux bits de même poids sont à 1 | 9 & 12 (1001 & 1100) | 8 (1000) |
| \| | OU bit-à-bit | Retourne 1 si l'un ou l'autre des deux bits de même poids est à 1 (ou les deux) | 9 \| 12 (1001 \| 1100) | 13 (1101) |
| ^ | OU bit-à-bit | Retourne 1 si l'un des deux bits de même poids est à 1 (mais pas les deux) | 9 ^ 12 (1001 ^ 1100) | 5 (0101) |
| ~ | Complément (NON) | Retourne 1 si le bit est à 0 (et inversement) | ~9 (~1001) | 6 (0110) |

Ces opérateurs permettent de manipuler les bits dans un entier. Si les paramètres de gauche et de droite sont des chaines de caractères, l'opérateur de bits agira sur les valeurs ASCII de ces caractères. (extrait de la documentation officielle).

# Syntaxe de base : Les opérateurs

## Les priorités

La priorité des opérateurs spécifie l'ordre dans lequel les valeurs doivent être analysées. Par exemple, dans l'expression 1 + 5 * 3, le résultat est 16 et non 18, car la multiplication ("*") a une priorité supérieure par rapport à l'addition ("+"). Des parenthèses peuvent être utilisées pour forcer la priorité, si nécessaire. Par exemple : (1 + 5) * 3 donnera 18.

| Priorité des opérateurs | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | () | [] | | | | | | | | | | |
| | -- | ++ | ! | ~ | - | | | | | | | |
| | * | / | % | | | | | | | | | |
| | + | - | | | | | | | | | | |
| | < | <= | >= | > | | | | | | | | |
| | == | != | | | | | | | | | | |
| | & | | | | | | | | | | | |
| | ^ | | | | | | | | | | | |
| | \| | | | | | | | | | | | |
| | && | | | | | | | | | | | |
| | \|\| | | | | | | | | | | | |
| | ? | : | | | | | | | | | | |
| | = | += | -= | *= | /= | %= | <<= | >>= | >>>= | &= | ^= | \|= |
| | AND | | | | | | | | | | | |
| | XOR | | | | | | | | | | | |

PARTIE 1

# Syntaxe de base : Les instructions conditionnelles

- **L'instruction if**
  - if (condition réalisée) { liste d'instructions }
- **L'instruction if ... Else**
  - if (condition réalisée) {liste d'instructions}
    else { autre série d'instructions }
- **L'instruction if ... elseif ... Else**
  - if (condition réalisée) {liste d'instructions}
    elseif (autre condition ) {autre série d'instructions }
    else (dernière condition réalisée) { série d'instructions }
- **Opérateur ternaire**
  - (condition) ? instruction si vrai : instruction si faux

```
$age = 20;
print ($age >= 18) ? "Adult" : "Not Adult";
```

# Syntaxe de base : Les instructions conditionnelles

- **L'instruction switch**

L'instruction switch équivaut à une série d'instructions if. En de nombreuses occasions, vous aurez besoin de comparer la même variable (ou expression) avec un grand nombre de valeurs différentes, et d'exécuter différentes parties de code suivant la valeur à laquelle elle est égale. C'est exactement à cela que sert l'instruction switch.

```
switch (Variable) {
case Valeur 1: Liste d'instructions break;
case Valeur 2: Liste d'instructions break;
...
case Valeurs n: Liste d'instructions break;
default: Liste d'instructions break;
}
```

# Syntaxe de base : **Les instructions conditionnelles**

- **La boucle for**
  - for ($i=1; $i<6; $i++) { echo "$i<br>"; }
- **La boucle while**
  - While(condition) {bloc d'instructions ;}
  - While (condition) :Instruction1 ;Instruction2 ;
    …. endwhile ;
- **La boucle do…while**
  - Do {bloc d'instructions ;}while(condition) ;
- **La boucle foreach**
  - Foreach ($tableau as $valeur) {insts utilisant $valeur ;}

# Syntaxe de base : **Les fonctions**

- **Déclaration et appel d'une fonction**

Une fonction utilisateur est définie par deux principaux éléments. Sa signature et son corps. La signature est elle même composée de deux parties : le nom et la liste de paramètres. Le corps, quant à lui, établit la suite d'instructions qui devront être exécutées.

La déclaration d'une nouvelle fonction se réalise au moyen du mot-clé ***function*** suivi du nom de la fonction et de ses arguments. Le code suivant illustre tout ça :

```
function nom_fonction($arg1, $arg2, ...$argn)
    {
        déclaration des variables ;
        bloc d'instructions ;
        //fin du corps de la fonction
        return $resultat ;
    }
```

# Syntaxe de base : Les fonctions

- **Fonction avec nombre d'arguments inconnu**
  - **func_num_args()** : fournit le nombre d'arguments qui ont été passés lors de l'appel de la fonction
  - **func_get_arg($i)** : retourne la valeur de la variable située à la position $i dans la liste des arguments passés en paramètres.
    - Ces arguments sont numérotés à partir de 0

```php
<?php
function produit()
{
    $nbarg = func_num_args();
    $prod = 1;
    // la fonction produit a ici $nbarg arguments
    for ($i = 0; $i < $nbarg; $i++) {
        $prod *= func_get_arg($i);
    }
    return $prod;
}
echo "le  produit est : ", produit(3, 77, 10, 5, 81, 9), "<br />";
// affiche le produit est 8 419 950
?>
```

# Syntaxe de base : **Les fonctions**

- **Passage de paramètre par référence**

  - Changer la valeur d'un argument dans la fonction ne change pas sa valeur à l'extérieur de la fonction) Si vous voulez que vos fonctions puissent changer la valeur des arguments, vous devez passer ces arguments par référence.
  - Pour passer une variable par référence, il faut que son nom soit précédé du symbole & (exemple &$a)

```
function dire_texte($qui, &$texte){ $texte = "Bienvenue $qui";}
$chaine = "Bonjour ";
dire_texte("cher codeur",$chaine);
echo $chaine; // affiche "Bienvenue cher codeur"
```

# Syntaxe de base : **Les fonctions**

- **L'appel récursif**
  - PHP admet les appels récursifs de fonctions
  - La récursivité consiste à créer une méthode ou une procédure qui appelle elle-même.
  - Chaque appel constitue un niveau de récursivité.

```
function display($number) {
if($number<=5){
        echo "$number <br/>";
        display($number+1);
                }
                                }
display(1);
```

# Syntaxe de base : **Les fonctions**

## Appel dynamique de fonctions

- Exécuter une fonction dont le nom n'est pas forcément connu à l'avance par le programmeur du script
- L'appel dynamique d'une fonction s'effectue en suivant le nom d'une variable contenant le nom de la fonction par des parenthèses

```php
<?php
function foo() {
    echo "dans foo()<br />\n";
}
function bar($arg )
{
    echo "Dans bar(); l'argument était '$arg'.<br />\n";
}
function echoit($string)
{
    echo $string;
}

$func = 'foo';
$func();        // Appel foo()

$func = 'bar';
$func('test');  // Appel bar()

$func = 'echoit';
$func('test');  // Appel echoit()
?>
```

PARTIE 1

# Syntaxe de base : **Les fonctions**

**Variables locales et variables globales**

- variables en PHP : global, static, local
- toute variable déclarée en dehors d'une fonction est globale
- utiliser une variable globale dans une fonction, l'instruction **global** suivie du nom de la variable
- Pour conserver la valeur acquise par une variable entre deux appels de la même fonction : l'instruction static.
  - Les variables statiques restent locales à la fonction et ne sont pas réutilisables à l'extérieur.

```
function cumul($prix)    {
     static $cumul = 0;
     static $i = 1;
     echo "Total des achats $i = ";
     $cumul += $prix;
     $i++;
     return $cumul;
 }echo cumul(175), "<br />";echo cumul(65), "<br />";echo cumul(69), "<br />";
```

PARTIE 1

# Syntaxe de base : Les tableaux

- **Principe**

- Un tableau en PHP est en fait une carte ordonnée. Une carte est un type qui associe des valeurs à des clés. Ce type est optimisé pour différentes utilisations ; il peut être considéré comme un tableau, une liste, une table de hachage, un dictionnaire, une collection, une pile, une file d'attente et probablement plus. On peut avoir, comme valeur d'un tableau, d'autres tableaux.
- Création à l'aide de la fonction array()
- Les éléments d'un tableau peuvent appartenir à des types distincts
- L'index d'un tableau en PHP commence de 0
- Pas de limites supérieures pour les tableaux
- La fonction count() pour avoir le nombre d'éléments d'un tableau

# Syntaxe de base : Les tableaux

- **Les tableaux indicés et les tableaux associatifs**
  - **Tableau indicé**
    - Accéder aux éléments par l'intermédiaire de numéros

```
$tableau[indice] = valeur;
$tableau = array(valeur0, valeur1,..., valeurN);
    $jour = array("Dimanche", "Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi",
"Samedi");
$note = array(20, 15, 12.6, 17, 10, 20, 11, 18, 19);
$jour[3] = "Mercredi";
$note[0] = 20;

$variable = $tableau[indice];
$JJ = $jour[6]; // affecte "Samedi" à $JJ
echo $note[1] + $note[5];
```

PARTIE 1

# Syntaxe de base : Les tableaux

- **Les tableaux indicés et les tableaux associatifs**
  - **Tableau associatif** (ou table de hachage)
    - Les éléments sont référencés par des chaînes de caractères associatives en guise de nom: la clé d'index

```
$tableau["indice"] = valeur;

$tableau = array(ind0 => val0, ind1 => val1,..., indN => valN);

$jour = array("Dimanche" => 1, "Lundi" => 2, "Mardi" => 3, "Mercredi" => 4, "Jeudi" => 5, "Vendredi" => 6, "Samedi" => 7);

$jour["Dimanche"] = 7;

$jour["Mercredi"] = "Le jour des enfants";

$variable = $tableau["indice"];

$JJ = $jour["Vendredi"]; //affecte 6 à $JJ

echo $jour["Lundi"]; //retourne la valeur 2
```

# Syntaxe de base : Les tableaux

## Tableaux multidimensionnels

- Les tableaux multidimensionnels sont des tableaux qui stockent un autre tableau à chaque index au lieu d'un seul élément. En d'autres termes, nous pouvons définir les tableaux multidimensionnels comme des tableaux de tableaux.

```
$tab1 = array(Val0, Val1,..., ValN);
$tab2 = array(Val0, Val1,..., ValN);
// Création d'un tableau à deux dimensions
    $tableau = array($tab1, $tab2);
      $mois = array("Janvier", "Février", "Mars", "Avril", "Mai", "Juin", "Juillet", "Août", "Septembre", "Octobre", "Novembre", "Décembre");
    $jour = array("Dimanche", "Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi");
    $element_date = array($mois, $jour);

    $variable = $tableau[indice][indice];
    $MM = $element_date[0][0]; //affecte "Janvier" à $MM
    echo $MM;
    echo $element_date[1][5] . " 7 " . $element_date[0][2] . "2002"; // retourne "vendredi  7 Mars 2002"
```

# Syntaxe de base : Les tableaux

**Lecture des éléments d'un tableau**

- Avec une boucle for

```
for ( $i=0; $i<count($tab) ; $i++){
if ($tab[$i]== "a") {echo $tab[$i], "<br  />"; }}
```

- Avec une boucle while

```
$i=0;
while ($tab[$i]){
if ($tab[$i][0] =="a" ) {echo $tab[$i], "<br /> "; }}
```

- Avec La boucle foreach

```php
$jour = array("Dimanche", "Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi");
foreach ($jour as $i => $JJ) {
    echo "La cellule n° " . $i . " : " . $JJ . "<br>";
}
```

PARTIE 1

# Syntaxe de base : Les tableaux

- **Lecture des éléments d'un tableau**
  - Parcours d'un tableau associatif
    - Réalisable en ajoutant avant la variable $valeur, la clé associée

```
$tableau = array(clé1 => val1, clé2 => val2, ..., cléN => valN);
foreach($tableau as $clé => $valeur) {
echo "Valeur ($clé): $valeur"; }

$jour = array("Dimanche" => 7, "Lundi" => 1, "Mardi" => 2,
  "Mercredi" => 3, "Jeudi" => 4, "Vendredi" => 5, "Samedi" => 6);
foreach($jour as $sJJ => $nJJ) {
echo "Le jour de la semaine n° ". $nJJ . " : " . $sJJ . "<br>"; }
```

# Syntaxe de base : Les tableaux

- **Fonctions de tri**
  - Tri selon les valeurs
    - La fonction sort() effectue un tri sur les valeurs des éléments d'un tableau selon un critère alphanumérique :
      - Le tableau initial est modifié et non récupérables dans son ordre original
      - Pour les tableaux associatifs les clés seront perdues et remplacées par un indice créé après le tri et commençant à 0
    - La fonction rsort() effectue la même action mais en ordre inverse.
    - La fonction asort() trie également les valeurs selon le critère des codes ASCII, mais en préservant les clés pour les tableaux associatifs
    - La fonction arsort() la même action mais en ordre inverse des codes ASCII

# Syntaxe de base : Les tableaux

- **Fonctions de tri**
  - Tri sur les clés
  - La fonction ksort() trie les clés du tableau selon le critère des codes ASCII, et préserve les associations clé /valeur
    - La fonction krsort() effectue la même action mais en ordre inverse des codes ASCII

```
$tab2 = array("1622" => "Molière", "1802" => "Hugo", "1920" => "Vian");
ksort($tab2);
echo "<h3 >  Tri sur les clés de \$tab2 </h3>";
foreach ($tab2 as $cle => $valeur) {
    echo "<b> l'élément a pour clé : $cle; et pour valeur : $valeur </b> <br />";
}
```

# Syntaxe de base : Les tableaux

- **Les fonctions des tableaux**

$tableau = array_count_values($variable);

retourne un tableau comptant le nombre d'occurrences des valeurs d'un tableau.

```
$array = array(1, "hello", 1, "world", "hello");
print_r(array_count_values($array));//Affiche des informations lisibles pour une variable comme var_dump()
```

$tableau = array_diff($var_1, $var_2, ..., $var_N);

retourne dans un tableau contenant les valeurs différentes entre deux ou plusieurs tableaux.

```
$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");
$a2=array("e"=>"red","f"=>"green","g"=>"blue");
$result=array_diff($a1,$a2);
print_r($result);
```

# Syntaxe de base : Les tableaux

- **Les fonctions de tableaux**

$tableau = array_intersect($var_1, $var_2, ..., $var_N);

retourne un tableau contenant les enregistrements communs aux tableaux entrés en argument.

```
$array1 = array("a" => "green", "red", "blue");
$array2 = array("b" => "green", "yellow", "red");
$result = array_intersect($array1, $array2);
print_r($result);
```

$tableau = array_flip($variable);

intervertit les paires clé/valeur dans un tableau.

```
$input = array("oranges", "apples", "pears");
$flipped = array_flip($input);
print_r($flipped);
```

# Syntaxe de base : Les tableaux

$tableau = array_filter($variable, "fonction")
retourne un tableau contenant les enregistrements filtrés d'un tableau à partir d'une fonction.

```
function impair($var)
{return ($var % 2 == 1);}

function pair($var)
{return ($var % 2 == 0);}

$array1 = array ("a"=>1, "b"=>2, "c"=>3, "d"=>4, "e"=>5);
$array2 = array (6, 7, 8, 9, 10, 11, 12);
echo "Impairs :\n";
print_r(array_filter($array1, "impair"));
echo "Pairs :\n";
print_r(array_filter($array2, "pair"));
```

## Principe

- PHP prend en charge l'accès au système de fichiers du système d'exploitation du serveur
- Les opérations sur les fichiers concernent la création, l'ouverture, la suppression, la copie, la lecture et l'écriture de fichiers
- Les possibilités d'accès au système de fichiers du serveur sont réglementées par les différents droits d'accès accordés au propriétaire, à son groupe et aux autres utilisateurs
- La communication entre le script PHP et le fichier est repérée par une variable, indiquant l'état du fichier et qui est passée en paramètre aux fonctions spécialisées pour le manipuler

## Ouverture de fichiers

- La fonction **fopen()** permet d'ouvrir un fichier, que ce soit pour le lire, le créer ou y écrire

: entier fopen(chaine nom du fichier, chaine mode);

- **mode :** indique le type d'opération qu'il sera possible d'effectuer sur le fichier après ouverture. Il s'agit d'une lettre (en réalité une chaîne de caractères) indiquant l'opération possible:
  - **r** (read) indique une ouverture en lecture seulement
  - **w** (write) indique une ouverture en écriture seulement (la fonction crée le fichier s'il n'existe pas)
  - **a** (append) indique une ouverture en écriture seulement avec ajout du contenu à la fin du fichier (la fonction crée le fichier s'il n'existe pas)
- lorsque le mode est suivie du caractère **+** celui-ci peut être lu et écrit
- le fait de faire suivre le mode par la lettre **b** entre crochets indique que le fichier est traité de façon binaire.

- **Ouverture de fichiers**

| MODES D'OUVERTURE DE FICHIER EN PHP | | | | |
|---|---|---|---|---|
| Caractère | Accès | Placement sur le fichier | Création si le fichier n'existe pas ? | Notes |
| r | lecture | début | non | - |
| r+ | lecture + écriture | début | non | - |
| w | écriture | début | oui | Réduit la taille du fichier à 0 |
| w+ | lecture + écriture | début | oui | Réduit la taille du fichier à 0 |
| a | écriture | fin | oui | Fonction fseek() n'a aucun effet |
| a+ | lecture + écriture | fin | oui | Fonction fseek() que pour la lecture |
| x | écriture | début | oui | Si le fichier existe, fopen() échoue → E_WARNING |
| x+ | lecture + écriture | début | oui | Si le fichier existe, fopen() échoue → E_WARNING |
| c | écriture | début | oui | Si le fichier existe, n'est pas tronqué |
| c+ | lecture + écriture | début | oui | Si le fichier existe, n'est pas tronqué |

- **Ouverture de fichiers**
  - Exemple

```
$fp = fopen("fichier.txt","r"); //lecture

$fp = fopen("fichier.txt","w"); //écriture depuis début du fichier
```

  - De plus, la fonction fopen permet d'ouvrir des fichiers présents sur le web grâce à leur URL.
    - Exemple : un script permettant de récupérer le contenu d'une page d'un site web:

```
<?
$fp = fopen("http://www.mondomaine.fr","r"); //lecture du fichier
while (!feof($fp)) { //on parcoure toutes les lignes
  $page .= fgets($fp, 4096); // lecture du contenu de la ligne
}
?>
```

## Ouverture de fichiers

- Il est généralement utile de tester si l'ouverture de fichier s'est bien déroulée ainsi que d'éventuellement stopper le script PHP si cela n'est pas le cas

```
<?
if (!$fp = fopen("fichier.txt", "r")) {
        echo "Echec de l'ouverture du fichier";
        exit;
    } else { // votre code;
        echo "Ouverture du fichier reussi";
    }

?>
```

- Un fichier ouvert avec la fonction **fopen()** doit être fermé, à la fin de son utilisation, par la fonction **fclose()** en lui passant en paramètre l'entier retourné par la fonction fopen()

## Lecture et écriture de fichiers

- Il est possible de lire le contenu d'un fichier et d'y écrire des informations grâce aux fonctions:
  - **fputs()** (ou l'alias **fwrite()** ) permet d'écrire une chaîne de caractères dans le fichier. Elle renvoie 0 en cas d'échec, 1 dans le cas contraire
    - booléen fputs(entier Etat_du_fichier, chaine Sortie);
  - **fgets()** permet de récupérer une ligne du fichier. Elle renvoie 0 en cas d'échec, 1 dans le cas contraire
    - fgets(entier Etat_du_fichier, entier Longueur);
      - Le paramètre Longueur désigne le nombre de caractères maximum que la fonction est sensée récupérer sur la ligne
      - Pour récupérer l'intégralité du contenu d'un fichier, il faut insérer la fonction fgets() dans une boucle while. On utilise la fonction **feof()**, fonction testant la fin du fichier.

- **Lecture et écriture de fichiers**

```php
<?php
if (!$fp = fopen("fichier.txt", "r")) {
    echo "Echec de l'ouverture du fichier";
} else {
    $Fichier = "";
    while (!feof($fp)) {
    $Ligne = fgets($fp, 255); // On récupère une ligne
    echo $Ligne ,'<br>';// On affiche la ligne
    $Fichier .= $Ligne . "<BR>"; // On stocke l'ensemble des lignes dans une variable
    }
    fclose($fp); // On ferme le fichier
}?>
```

## Lecture et écriture de fichiers

- Pour stocker des informations dans le fichier, il faut dans un premier temps ouvrir le fichier en écriture en le créant s'il n'existe pas
  - Deux choix : le mode 'w' et le mode 'a'.

```
<?
$nom = "Jean";
$email = "jean@dupont.fr";
$fp = fopen("fichier.txt", "a"); // ouverture du fichier en écriture
fputs($fp, "\n"); // on va a la ligne
fputs($fp, $nom . "|" . $email); // on écrit le nom et email dans le fichier
fclose($fp);

?>
```

- **Les tests de fichiers**

  - **is_dir()** permet de savoir si le fichier dont le nom est passé en paramètre correspond à un répertoire.

    - La fonction is_dir() renvoie 1 s'il s'agit d'un répertoire, 0 dans le cas contraire

booléen is_dir(chaine Nom_du_fichier);

```
echo'<br>';
var_dump(is_dir('fichier.txt'));
echo'<br>';
var_dump(is_dir('..'));
```

- **Les tests de fichiers**

  - **is_executable()** permet de savoir si le fichier dont le nom est passé en paramètre est exécutable.

    - La fonction is_executable() renvoie 1 si le fichier est exécutable, 0 dans le cas contraire : booléen is_executable(chaine Nom_du_fichier);

```
$fp1 = "fichier.txt";
if (is_executable($fp1)) {
    echo $fp1.' est exécutable';
} else {
    echo $fp1.' n\'est pas exécutable';
}
```

- **Le téléchargement de fichier**

  - Le langage PHP dispose de plusieurs outils facilitant le téléchargement vers le serveur et la gestion des fichiers provenant d'un client

  - Un formulaire avec un champ file permet de telecharger un fichier, qui sera ensuite traité par un script PHP."

```
<form method="POST" action="traitement.php"
      enctype="multipart/form-data">
 <input type="hidden" name="MAX_FILE_SIZE" value="Taille_Octets">
 <input type="file" name="fichier" size="30"><br>
 <input type="submit" name="telechargement" value="telecharger">
</form>
```

## Le téléchargement de fichier en PHP

- Un champ caché doit être présent dans le formulaire afin de spécifier une taille maximum (MAX_FILE_SIZE) pour le fichier à télécharger. Cette taille est par défaut égale à deux mégaoctets.
- En PHP, le tableau associatif global $_FILES contient plusieurs informations sur le fichier téléchargé.

  - **$_FILES['fichier']['name'] :** fournit le nom d'origine du fichier.
  - **$_FILES['fichier']['type'] :** fournit le type MIME du fichier.
  - **$_FILES['fichier']['size'] :** fournit la taille en octets du fichier.
  - **$_FILES['fichier']['tmp_name'] :**fournit le nom temporaire du fichier.

- **Le téléchargement de fichier**

  - Par défaut, le fichier envoyé par le client est stocké directement dans le répertoire indiqué par l'option de configuration upload_tmp_dir dans le fichier **php.ini**.

    **upload_tmp_dir = c:\PHP\uploadtemp**

  - Plusieurs fonctions spécialisées permettent la validation d'un fichier téléchargé pour son utilisation ultérieure.

    - **La fonction is_uploaded_file** indique si le fichier a bien été téléchargé par la méthode POST.

    $booleen=**is_uploaded_file(**$_FILES'fichier']['tmp_name']**);**

  - **La fonction move_uploaded_file** vérifie si le fichier a été téléchargé par la méthode HTTP POST, puis si c'est le cas le déplace vers l'emplacement spécifié.

## Le téléchargement de fichier

- Il est possible de télécharger plusieurs fichiers en même temps, en utilisant des crochets à la suite du nom du champ afin d'indiquer que les informations relatives aux fichiers seront stockées dans un tableau.

```
<form action="traitement.php" method="POST‟ enctype="multipart/form-data">
  <input type="file" name= "fichier[]” ><br>
  ...
  <input type="file" name= "fichierN[]” > <br>
  <input type="submit" value="Envoyer" name="soumission">
</form>
for($i = 0; $i < sizeof($_FILES['fichier']['name']); $i++){
echo "Nom du fichier :  ». $_FILES['fichier']['name'][$i];}
```

- Les fichiers téléchargés sont automatiquement effacés du répertoire temporaire au terme du script.
  - il est nécessaire de déplacer les fichiers vers un autre endroit ou de les renommer si ceux-ci doivent être conservés.

```
<!-- Fichier : formulaire.html -->
<html><body>
    <form method="POST" action="traitement.php" enctype="multipart/form-data">
      <input type="hidden" name="MAX_FILE_SIZE" value="1000000">
      <input type="file" name="fichier" size="30"><br>
      <input type="submit" name="telechargement" value="telecharger">
    </form> </body></html>
```

```php
<?php  /* Fichier : traitement.php*/
$repertoire = 'dossierExterne/';
if (is_uploaded_file($_FILES['fichier']['tmp_name'])) {
    $fichier_temp = $_FILES['fichier']['tmp_name'];
     echo "<h3>Le fichier a été téléchargé avec succès " . "à l'emplacement suivant :
<br>'" . $fichier_temp . "'</h3>";
    $nom_fichier = $_FILES['fichier']['name'];
    echo "<h3>Le nom d'origine du fichier est '" . $nom_fichier . "'.</h3>";
    echo "<h3>Le type du fichier est '" . $_FILES['fichier']['type'] . "'.</h3>";
       echo "<h3>La taille du fichier est de '" . $_FILES['fichier']['size'] . "
octets'.</h3>";
    copy($fichier_temp , $repertoire . $nom_fichier);
} else {
    echo '<h3 style="color:#FF0000">ATTENTION, ce fichier peut être à l\'origine'
        . ' d\'une attaque : ' . $_FILES['fichier']['name'] . "!</h3>";}
```

## La manipulation de fichiers distants

- Il peut être utile de manipuler des fichiers à distance, c'est-à-dire par le biais des protocoles de transferts HTTP ou FTP.
  - PHP autorise l'ouverture d'un fichier par l'intermédiaire d'une adresse URL dans la fonction fopen

```
$id_fichier = fopen("http://www.site.com/index.html", "r");
```

  - allow_url_fopen doit être activée dans php.ini

## La manipulation de fichiers distants

- Alternatives modernes

1. cURL (plus flexible et sécurisé) :

```
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, "http://www.site.com/index.html");
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$contenu = curl_exec($ch);
curl_close($ch);
```

2. file_get_contents() (plus simple pour les petits fichiers) :

```
$contenu = file_get_contents("http://www.site.com/index.html");
```

- **La manipulation de fichiers distants**
  - L'écriture sur un serveur distant est possible, à condition de passer en argument une adresse FTP à la fonction fopen() et que ce fichier soit nouveau.

```
$id_fichier = fopen("ftp://ftp.site.com/new_page.html", "w");
```

  - L'accès en écriture directement sur un site, nécessite souvent, la saisie d'un nom d'utilisateur et d'un mot de passe dans l'adresse afin d'éviter toutes intrusions inopportunes

```
ftp://nom_utilisateur:mot_passe@ftp.site.com/nouvelle_page.html
```

  - La modification d'un fichier distant n'est pas réalisable par ce moyen

- **La manipulation de fichiers distants : Limitations**
  - Sécurité - Les identifiants transitent en clair (préférer SFTP/FTPS)
  - Fiabilité - Sensible aux problèmes de réseau
  - Performance - Plus lent qu'une connexion directe

```
$ch = curl_init();
$fp = fopen('localfile.html', 'r');
curl_setopt($ch, CURLOPT_URL, "ftp://ftp.site.com/new_page.html");
curl_setopt($ch, CURLOPT_UPLOAD, 1);
curl_setopt($ch, CURLOPT_INFILE, $fp);
curl_setopt($ch, CURLOPT_INFILESIZE, filesize('localfile.html'));
curl_exec($ch);
curl_close($ch);
```

- **Les fonctions de système de fichiers**

| FONCTIONS SUR LES FICHIERS | | | |
|---|---|---|---|
| Nom | Description | Retour OK | Retour PAS OK |
| **fopen**($filename)<br>**fclose**($file) | Ouvre un fichier<br>Ferme un fichier | {FICHIER]<br>"TRUE" | "FALSE"<br>"FALSE" |
| **is_readable**($filename)<br>**is_writ[e]able**($filename)<br>**is_file**($filename)<br>**is_dir**($filename)<br>**file_exists**($filename) | Fichier disponible en lecture ?<br>Fichier disponible en écriture ?<br>Est-ce un fichier ?<br>Est-ce un répertoire ?<br>Est-ce que le fichier existe ? | "TRUE"<br>"TRUE"<br>"TRUE"<br>"TRUE"<br>"TRUE" | "FALSE"<br>"FALSE"<br>"FALSE"<br>"FALSE"<br>"FALSE"<br>"FALSE" |
| **feof**($file) | Y a-t-il des donnés à lire sur le fichier ? | "TRUE" | "FALSE" |
| **file**($filename)<br>**file_get_contents**($filename) | Récupérer le contenu du fichier dans un tableau<br>Récupérer le contenu du fichier dans une chaîne | ARRAY<br>STRING | "FALSE"<br>"FALSE" |
| **fgetc**($file)<br>**fgets**($file [,$length])<br>**fscanf**($file, $format [,&$args...])<br>**fread**($file, $length) | Lit un caractère<br>Lit une ligne<br>Lit une ligne formatée<br>Lit jusqu'à $length octets | STRING<br>STRING<br>ARRAY \| INT<br>STRING | "FALSE"<br>"FALSE"<br>-<br>"FALSE" |
| **fprintf**($file, $format [,$args...])<br>**fwrite/fputs**($file, $string [, $length]) | Écrit une ligne formatée dans un fichier<br>Écrit une chaîne dans un fichier (éventuellement de $length octets) | INT<br>INT | -<br>"FALSE" |
| **rename**($old, $new)<br>**unlink**($filename)<br>**dirname**($path)<br>**mkdir**($pathname, $rights, $recursive)<br>**rmdir**($dirname) | Renommer un répertoire/fichier<br>Supprimer un fichier<br>Récupérer le chemin du dossier parent<br>Créer un répertoire (ou ensemble de répertoires)<br>Supprimer un répertoire | "TRUE"<br>"TRUE"<br>STRING<br>"TRUE"<br>"TRUE" | "FALSE"<br>"FALSE"<br>STRING<br>"FALSE"<br>"FALSE" |
| **ftell**($file)<br>**fseek**($file, $offset, $from)<br>**rewind**($file) | Récupérer la position du curseur dans le fichier<br>Déplace le curseur de $offset octets depuis la position $from<br>Replace le pointeur au début du fichier | INT<br>0<br>"TRUE" | "FALSE"<br>-1<br>"FALSE" |

- **Les fonctions de dossiers**

nombre = chroot($chaine);
définit la chaîne de caractères comme la nouvelle racine

nombre = chdir($chaine);
définit le dossier en cours comme celui précisé par la chaîne de caractères.

$objet = dir($chaine);
crée un objet à partir du dossier spécifié.

closedir($identificateur_dossier);
ferme le dossier à partir d'un identificateur retourné par opendir.

$chaine = getcwd();
retourne le nom du dossier en cours.

$identificateur_dossier = opendir($chaine);
ouvre un dossier et retourne un identificateur de l'objet.

$chaine = readdir($identificateur_dossier);
retourne le fichier suivant dans le dossier spécifié par l'entremise de son identificateur.

## Création d'une classe et d'un objet

- **Une classe est composée de deux parties:**
  - Les attributs: il s'agit des données représentant l'état de l'objet
  - Les méthodes : il s'agit des opérations applicables aux objets

```php
class client
{
var $nom;
var $ville;
var $naiss;
function age()
{
        $jour = getdate();
        $an = $jour["year"];
        $age = $an - $this->naiss;
 echo "Il a $age ans cette année <br />";
}
}
```

```php
//création d'un objet
$client1 = new client();
//affectation des propriétés de l'objet
$client1->nom = "Dupont";
$client1->naiss = "1961";
$client1->ville = "Angers";
//utilisation des propriétés
echo "le nom du client1 est ",
 $client1->nom, "<br />";
echo "la ville du client1 est ",
 $client1->ville, "<br />";
echo "le client1 est né en ",
$client1->naiss, "<br />";
//appel de la méthode age()
$client1->age();
```

## Manipulation des classes et des objets

- Instanciation de la classe
  - $Nom_de_l_objet = new Nom_de_la_classe;
- Accéder aux propriétés d'un objet
  - $Nom_de_l_objet->Nom_de_la_donnee_membre = Valeur;
- Accéder aux méthodes d'un objet
  - $Nom_de_l_objet->Nom_de_la_fonction_membre(parametre1,parametre2,...);
- La variable $this :La pseudo-variable $this est disponible lorsqu'une méthode est appelée depuis un contexte objet. $this est la valeur de l'objet appelant.
  - $this->age = $Age;

### POO, visibilité d'une classe

La visibilité permet de définir le champ d'utilisation des propriétés et des méthodes d'une classe. Il existe trois niveaux de visibilité :

**public** : propriétés ou méthodes accessibles au niveau de l'objet, de la classe et des classes étendues (extends).

```
class Bouteille{
    public $marque = "coca-cola";
}
$mabouteille = new Bouteille();
echo $mabouteille->marque;
```

**POO, visibilité d'une classe**

**private** : propriétés ou méthodes accessibles seulement au niveau de la classe qui les a défini mais pas au niveau des objets, ni au niveau des classes étendues (extends).

```
class Bouteille{
    private $marque = "coca-cola";
    public function info(){
        echo $this->marque;
    }
}
$mabouteille = new Bouteille();
echo $mabouteille->marque; // affiche une erreur
$mabouteille->info(); // affiche 'coca-cola'
```

POO, visibilité d'une classe

**protected** : propriétés ou méthodes accessibles au niveau de la classe qui les a défini, au niveau des classes étendues (extends), mais pas au niveau des objets.

```
class Bouteille{
    private $marque = "coca-cola";
    protected $volume = "1.5L";
    public function info(){ echo $this->marque; }
}
class canette extends Bouteille{
    public function info(){echo $this->volume; } // propriété accéssible dans la classe enfant
 }
$unecanette= new canette();
echo $unecanette->volume; // affiche une erreur
$unecanette->info();
```

**POO, visibilité d'une classe**

**final** : si une méthode est définie comme *final*, elle ne pourra pas être surchargée par les classes filles. Si une classe est définie comme *final*, elle ne pourra pas être étendue.

```
final class Bouteille{
    private $marque = "coca-cola";
    protected $volume = "1.5L";
    public function info(){
        echo $this->marque;
    }
}
class canette extends Bouteille{ // erreur fatale
    }
```

### POO, notion d'héritage

L'héritage consiste à créer un autre objet (objet enfant) à partir d'une classe existante (parent). La classe enfant hérite, de ce fait, des propriétés et des méthodes de la classe parente.

```
class Animaux{
    public $classification = "être vivant<br>";
    public function info(){
        echo "Je suis un animal";
    }
}
class Chien extends Animaux{
    public function caracteristique(){
        echo "J'aboie <br>";
    }
}
$milou = new Chien();
$milou->classification;
echo $milou->classification;
$milou->info();
$milou->caracteristique();
```

**POO, notion d'héritage**

En étant dans la classe étendue, vous pouvez récupérer des informations de la classe de base avec le mot clé **parent**.

```
class Animaux{
    public $classification = "être vivant<br>";
    public function info(){
        echo "Je suis un animal";
    }
}
class Chien extends Animaux{
    public function caracteristique(){
        echo "J'aboie <br> ";
         parent::info(); // référence à la méthode info() de la classe mère(operateur
de resolution de portee)
    }
}

$milou = new Chien();
$milou->caracteristique();
```

### Constructeur de classe

Le constructeur est une méthode de la classe qui est automatique appelée à la création d'un objet (instance de la classe). Cette méthode est pratique pour initialiser les paramètres d'un objet au moment de le créer.

```
class Voiture{
    const NOMBREAIRBAG = 3;
    function __construct() { // constructeur
        echo "Une nouvelle voiture vient d'être fabriquée";
    }
    public function demarrage(){
        echo "La voiture de marque";
    }
}$mavoiture  = new Voiture();
```

### Destructeur de classe

Le destructeur est une méthode de la classe qui est automatique appelée à la destruction d'un objet (instance de la classe), c'est à dire lorsque la référence de cet objet est détruite.

```php
class Voiture{
    const NOMBREAIRBAG = 3;
    function __construct() {
        echo "Une nouvelle voiture vient d'être fabriquée";
    }
    public function __destruct() { // destructeur
        echo 'Voiture detruite';
    }
    public function demarrage(){
        echo "La voiture de marque";
    }
}
$mavoiture  = new Voiture(); // appel de __construct
$mavoiture = null; // on détruit la référence : appel de __destruct
```

**Opérateur de résolution de portée ::**

Vous pouvez accéder aux propriétés et aux méthodes d'une classe sans recourir aux objets avec l'opérateur de résolution de portée **::**

Il peut être utilisé avec le nom de la classe et non pas avec le nom de l'instance. Cet opérateur fait donc référence à la classe et non pas à l'objet (instance de classe).

```
class Voiture{
    const NOMBREAIRBAG = 3;
    public function demarrage(){
        echo "La voiture de marque";
    }
}
echo Voiture::NOMBREAIRBAG;  // les :: s'applique à la classe Voiture
```

### Les méthodes magiques

Les méthodes magiques sont des méthodes qui sont appelées automatiquement suite à un événement particulier.

Surcharge : **__set()**. Cette méthode est appelée au moment de la création d'une propriété dynamique de l'objet.

Surcharge : **__get()**. Cette méthode est appelée au moment de la consultation d'une propriété créée dynamiquement sur l'objet.

Conversion : **__toString()**. Cette méthode est appelée lorsque l'objet est traitée comme une chaîne (echo $maVoiture, par exemple).

Méthodes inaccessibles : **__call()**. Cette méthode est appelée lorsque vous appelez une méthode qui n'existe pas sur un objet.

### Les méthodes magiques

Surcharge : **__set()**. Cette méthode est appelée au moment de la création d'une propriété dynamique de l'objet.

```
class Soda{
    public $marque;
    private $prop = [];
    function __construct($marque) {
        $this->marque = $marque;
    }
    function __set($name, $value) {
        echo "Définition de '$name' à la valeur '$value'\n";
        $this->prop[$name] = $value;
    }
}
$mabouteille = new Soda('coca');
echo $mabouteille->volume = '2L';
```

### Les méthodes magiques

Surcharge : **__get()**. Cette méthode est appelée au moment de la consultation d'une propriété créée dynamiquement sur l'objet.

```php
class Soda
{
    public $marque;
    private $prop = [];
    function __construct($marque)
    {
        $this->marque = $marque;
    }
    function __get($name)
    {
        echo "Récupération de la proriété $name";
        if (array_key_exists($name, $this->prop)) {
            return $this->prop[$name];
        } else {echo "<br>propriété non existante";}
    }
}
$mabouteille = new Soda('coca');
echo $mabouteille->volume = '2L'; echo $mabouteille->volume;
```

### Les méthodes magiques

La méthode **__toString()** détermine comment l'objet doit réagir lorsqu'il est traité comme une chaîne de caractères.

```
  class ClasseTest
{
    public $foo;

    public function __construct($foo)
    {
        $this->foo = $foo;
    }

    public function __toString()
    {
        return $this->foo;
    }
}

$class = new ClasseTest('Bonjour');
echo $class;
```

### Les méthodes magiques

**La méthode __call()** est une méthode magique qui intercepte les appels à des méthodes non définies

```
    class Soda{
        public $marque;
        private $prop = [];
        function __construct($marque) {
            $this->marque = $marque;
        }
        public function __call($name, $arguments){
            echo "Appel de la méthode '$name' - arguments : "
                    . implode(', ', $arguments);// implode rassemble les éléments d'un
tableau en une chaîne.
        }

    }
    $mabouteille = new Soda('coca');
    echo $mabouteille->donneVolume('litre');
```

## clonage

Il est possible de cloner un objet avec la syntaxe **clone**. Le clonage consiste à créer un autre objet identique à partir d'un objet existant. Même si les clones sont identiques, ce ne sont pas les mêmes. Si vous modifiez la valeur d'une propriété d'un objet cloné, la valeur de propriété du clone ne sera pas modifiée, et inversement.

```php
class Bouteille{
    public $marque = "coca-cola";
    protected $volume = "1.5L";
    public function info(){
        echo $this->marque;
    }
}
$unebouteille1 = new Bouteille();
$unebouteille2 = clone $unebouteille1; // clonage de l'objet
$unebouteille1->marque = "orangina"; // changement de marque sur l'objet cloné
$unebouteille1->info(); // information sur l'objet cloné
echo "<br>";
$unebouteille2->info();  // information sur le clone
```

### classe abstraite

Une classe abstraite est une classe qui ne peut pas être instanciée, et toute classe qui contient une méthode abstraite doit être abstraite.

Cependant, une classe abstraite n'est pas obligée d'avoir que des méthodes abstraites. Elle peut contenir par exemple un constructeur. Une méthode abstraite est une méthode vide (qui n'est donc pas codée), comme la méthode *seDeplacer()* ci-dessous.

```
abstract class Animaux {
    public function __construct() {
        echo "C'est un animal";
    }
    abstract public function seDeplacer();
}
```

### classe abstraite

Une classe abstraite force les classes qui la dérivent à implémenter des fonctions aux modes d'accès public ou protégé. Autrement dit, dans l'exemple ci-dessus, les classes qui héritent de la classe mère devront implémenter obligatoirement la méthode *seDeplacer()*.

```
abstract class Animaux {
    public function __construct() {
        echo "C'est un animal";
    }
    abstract public function seDeplacer();
}

class Chien extends Animaux {
    public function seDeplacer() {
        echo 'le chien se déplace à quatres pattes';
    }
}
$milou = new Chien;
echo '<br>';
$milou->seDeplacer();
```

### Interface

L'interface diffère de la classe abstraite puisque qu'en soit ce n'est pas une classe : on ne peut donc ni l'instancier, ni l'hériter. Mais elle ressemble à la classe abstraite parce qu'on retrouve cette notion de contrainte : obligation d'implémenter ses méthodes.

La création d'une interface se fait avec la syntaxe **interface**

Il est possible de créer plusieurs autres fonctions à une interface.  Note : les méthodes d'une interface sont toujours en mode d'accès *public*.

L'utilisation d'une interface se fait avec la syntaxe **implements**

Interface

```php
interface ActionDeVoler {
    public function voler();
}
// class abstraite
abstract class Animaux {
    public function __construct() {echo "C'est un animal";}
    abstract public function seDeplacer();
}
// héritage avec implémentation de l'interface
class Oiseau extends Animaux implements ActionDeVoler {
    public function seDeplacer() {// visibilité SUPERIEUR qui génère une erreur fatale
        echo 'Un oiseau se déplace en volant'; }
    public function voler(){
        echo "Un oiseau vole comme un avion";  }
}
$pigeon = new Oiseau;
echo '<br>';
$pigeon->seDeplacer();
echo '<br>';
$pigeon->voler();
```

## Manipulation des classes et des objets

- Les informations méta sur les classes et les objets

| | |
|---|---|
| Get_class() | Détermination de la classe d'un objet |
| Get_parent_class() | Détermination des super-classes d'un objet |
| Method_exists() | Détermination de la présence d'une méthode dans un objet |
| Class_exists() | Détermination de la présence d'une définition de classe |
| Is_subclass_of() | Vérifie si une classe est une sous classe d'une autre |
| Get_class_methods() | Retourne les méthodes d'une classe dans un tableau |
| Get_declared_classes() | Retourne les classes déclarées dans un tableau |
| Get_class_vars() | Retourne les variables de classe dans un tableau |
| Get_object_vars() | Retourne les variables d'un objet dans un tableau |

- Les dates et les heures

## Les fonctions de date et d'heure

true | false = checkdate(mois, jour, année);

**vérifie la validité d'une date.**

$chaine = date(format [, nombre]);

**retourne une chaîne de caractères date/heure selon le format spécifié et représentant la date courante par défaut.**

$tableau = getdate([nombre]);

**retourne les éléments de date et d'heure dans un tableau associatif.**

$tableau = gettimeofday();

**retourne l'heure courante dans un tableau associatif.**

$chaine = gmdate(format [, nombre]);

**retourne une chaîne de caractères date/heure GMT/CUT selon le format spécifié et représentant la date courante par défaut.**

$nombre = gmmktime(heure, minute, seconde, mois, jour, année [, 1/0]);

**retourne l'instant UNIX d'une date GMT spécifiée et avec éventuellement une heure d'hiver**