

# M107 Développer des sites web dynamiques

# Principe Client/serveur

Le principe du client-serveur est un modèle de communication informatique dans lequel un programme, appelé client, envoie des demandes à un autre programme, appelé serveur, via un réseau. Ce modèle est largement utilisé dans le domaine de l'informatique pour permettre à des systèmes différents de communiquer et de partager des ressources.



Fig. : Un client, un serveur

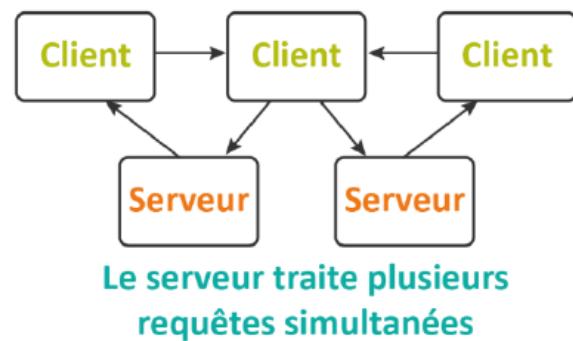


Fig. : Plusieurs clients, un serveur



Fig. : Un client, plusieurs serveurs

# Principe Client/serveur

C'est quoi un Client ?

-  Un client est un consommateur de services.
-  Le client déclenche la demande de service.
-  Une requête est un appel de fonction, la réponse éventuelle pouvant être synchrone ou asynchrone (le client peut émettre d'autres requêtes sans attendre)
-  Les arguments et les réponses sont énoncés dans un protocole
-  Le respect du protocole entre les deux processus communicants est obligatoire. Ce protocole étant décrit dans un RFC (Request For Comment).

# Principe Client/serveur

## Qui sont les clients ?

Les navigateurs web jouent le rôle de clients :



**Brave**

- Respect et protection de la vie privée
- Interface et ergonomie soignée
- Le plus rapide du marché



**Chrome**

- Très bonnes performances
- Simple et agréable à utiliser
- Un navigateur bien sécurisé



**Firefox**

- Un des plus innovants du marché
- Fonctionnalités pour optimiser l'UX
- Gère les derniers standards vidéo AV1



**UC Browser**



**Next**



**Edge**

- Performances correctes
- Plus léger, rapide et moderne
- L'intégration à l'écosystème Windows/Microsoft...



**Opéra**

- Stable et performant
- Débits rapides
- Sécurité accrue avec outils de chiffrement efficaces



**Vivaldi**

- Bonne alternative aux navigateurs du marché
- Niveau de personnalisation
- Fonctionnalités intuitives et originales



**Safari**



**Maxthon**



**UR Browser**

# Principe Client/serveur

## C'est quoi un Serveur ?

un serveur est un composant essentiel de l'infrastructure informatique moderne, fournissant des services, des ressources et des fonctionnalités aux clients sur un réseau, tout en offrant des avantages tels que la centralisation des ressources, la fiabilité, la sécurité et l'évolutivité.

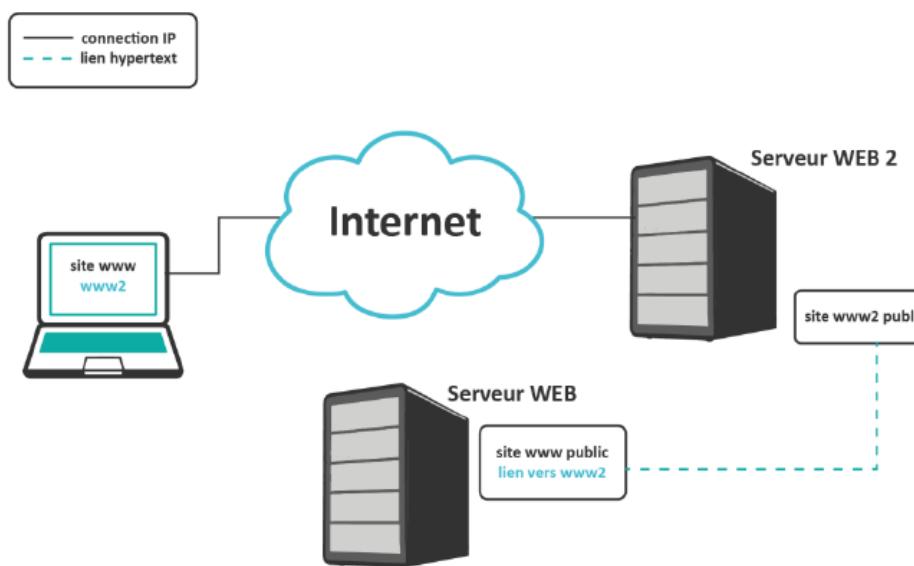


Fig : Internet connexion client serveur. Source: <https://www.ipgp.fr/>

# Principe Client/serveur

## C'est quoi un Middleware ?

- La liaison entre le client et le serveur qui se charge de toutes les communications entre les processus est appelée Middleware.
- Un middleware est un logiciel médiateur ou intergiciel qui se charge de la liaison.

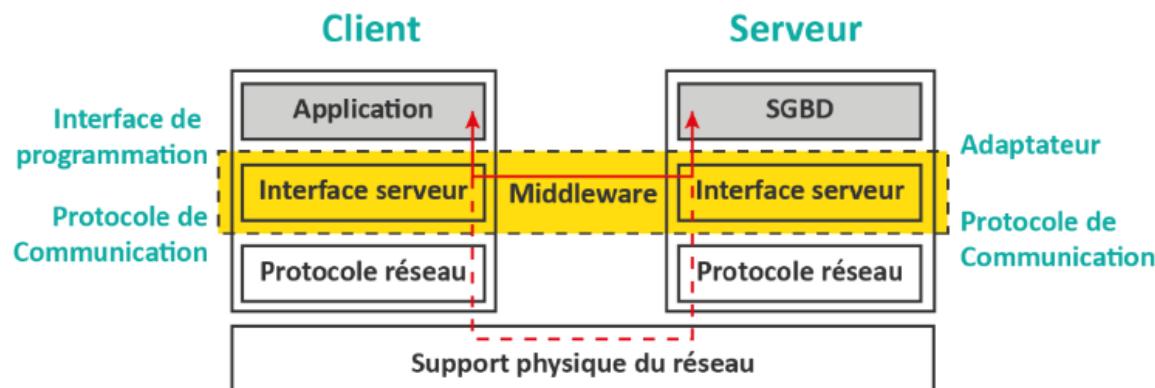


Fig : Représentation Middleware selon Gartner Group

### Les fonctions d'un Middleware

- Procédure d'établissement de connexion
- Exécution des requêtes
- Récupération des résultats
- Procédure de fermeture de connexion
- Initiation des processus sur différents sites
- Services de répertoire (nommage)
- Accès aux données à distance
- Gestion des accès concurrents
- Sécurité et intégrité
- Monitoring
- Terminaison des processus
- Mise en cache des résultats et des requêtes

### Les services d'un Middleware

- Conversion
- Adressage
- Sécurité
- Communication

# Principe Client/serveur

## Avantages et Inconvénients

AVANTAGES	INCONVÉNIENTS
<ul style="list-style-type: none"><li>• Des ressources centralisées, tout en garantissant la cohérence des données.</li><li>• Une meilleure sécurité, puisque le nombre de points d'entrée permettant l'accès aux données est connus.</li><li>• Une administration au niveau serveur.</li><li>• Un réseau évolutif, on peut supprimer ou rajouter des clients sans perturber le fonctionnement du réseau et sans modifications majeures.</li><li>• Une interface utilisateur riche.</li><li>• Une appropriation des applications par l'utilisateur.</li><li>• Une notion d'interopérabilité.</li></ul>	<ul style="list-style-type: none"><li>• Déploiement coûteux : un coût élevé dû à la technicité du serveur</li><li>• Trafic réseau important : tout le réseau est architecturé autour du serveur</li><li>• Déploiement difficile : le poste client doit constamment être mis à jour pour répondre au besoin</li></ul>

# Types d'architectures

- 1. Architecture 1-tiers :** Aussi appelée architecture centralisée ou monolithique, toute l'application est regroupée en une seule entité, souvent sur un seul ordinateur. Simple à développer et déployer, mais peut devenir difficile à maintenir et à faire évoluer.
- 2. Architecture 2-tiers :** Connue sous le nom d'architecture client-serveur, elle sépare les responsabilités entre les clients (qui envoient des demandes) et les serveurs (qui répondent en fournissant des services ou des données). Utilisée couramment dans les applications web et les bases de données.
- 3. Architecture 3-tiers :** Étend le modèle client-serveur en ajoutant une couche intermédiaire, généralement la couche de présentation, d'application et de données. Offre une meilleure modularité et une plus grande flexibilité dans le développement et la maintenance des applications.
- 4. Architecture n-tiers :** Une extension de l'architecture 3-tiers, avec plusieurs couches intermédiaires ajoutées, telles que des services de traitement des transactions, de sécurité, etc. Offre une haute disponibilité et une meilleure gestion des ressources, mais peut être plus complexe à gérer.

# Types de serveurs web

## Définition

### Un serveur web

- Un serveur Web (aussi appelé serveur http), est tout type de serveur qui permet de diffuser des contenus Web sur Internet ou Intranet. C'est un service logiciel utilisé pour communiquer entre deux appareils sur un réseau.
- Un serveur web sert à rendre accessible des pages web sur internet via le protocole HTTP.
- Un serveur web répond par défaut sur le port 80.
- Pour qu'un site Web soit accessible à tout moment, le serveur Web sur lequel il est hébergé doit être connecté à Internet en permanence
- Un serveur Web en architecture 3 tiers est composé d'un système d'exploitation, un serveur HTTP, un langage serveur et un système de gestion de base de données (SGBD), cela constituant une plate-forme.

# Types de serveurs web

## Définition

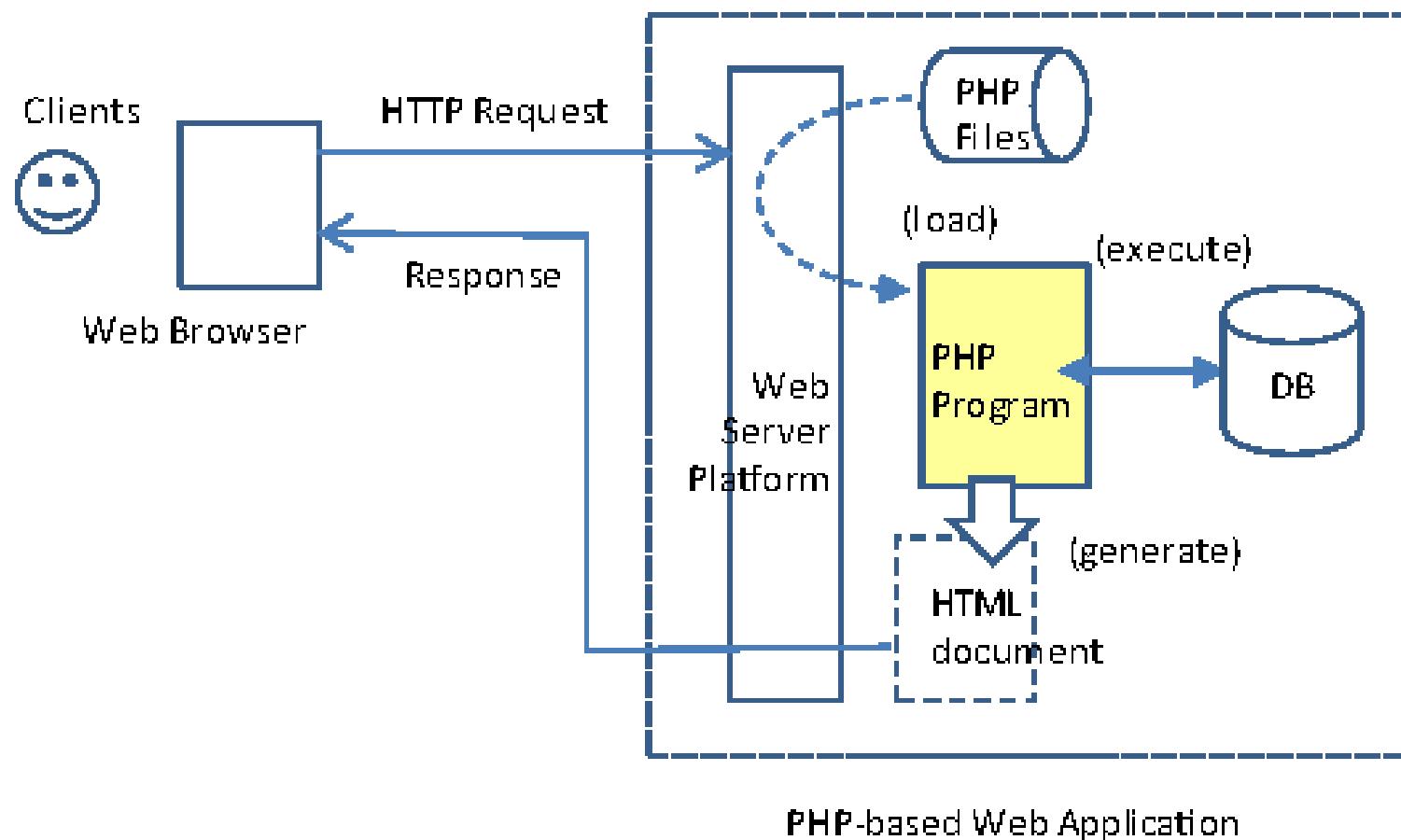
### Un serveur web statique

- (aussi appelé une pile) est composé d'un ordinateur (matériel) et d'un serveur HTTP (logiciel).
- Il est appelé « statique » car le serveur envoie les fichiers hébergés « tels quels » vers le navigateur.

### Un serveur web dynamique

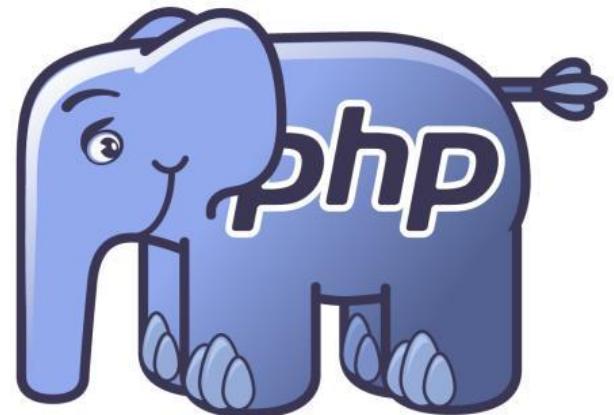
- Possède d'autres composants logiciels, certains qu'on retrouve fréquemment dont un serveur d'applications et une base de données.
- Il est appelé « dynamique » car le serveur d'applications met à jour les fichiers hébergés avant de les envoyer au navigateur via HTTP.

# Architecture web - PHP

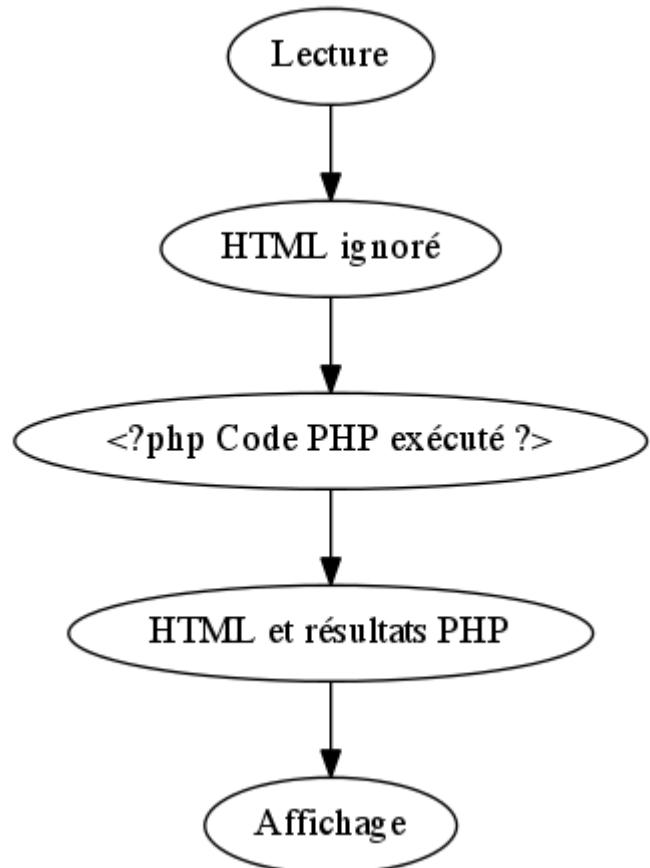


# Le langage PHP

- Langage de script (interprété) . Principalement utilisé côté serveur
- Langage impératif avec programmation orientée objet possible
- Crée en 1994-1995 par RasmusLerdorf
- Acronyme initial : Personal Home Page
- Acronyme récursif : PHP: HypertextPreprocessor
- Langage multi plate-forme (UNIX / Windows...)
- Open Source
- Versions actuelles : 8



# Fonctionnement



# Choix d'un serveur web

Le choix d'un serveur web pour le développement de sites et d'applications PHP dépend du système d'exploitation utilisé.

Pour **MacOS**, **MAMP** est recommandé, tandis que pour **Linux**, **LAMP** est préférable. Les utilisateurs de **Windows** peuvent opter pour **WAMP**. Une alternative **multi-plateforme** est également disponible avec **XAMPP**.



# Syntaxe de PHP

- La syntaxe de PHP est celle de la famille «C»(C, C++, Java, ...)
- Chaque instruction se termine par «;»
- Commentaires:

```
/*jusqu'au prochain */  
//jusqu'à la fin de la ligne  
#jusqu'à la fin de la ligne
```

- Délimitation du code PHP dans le fichier .php

```
<?phpCode PHP ?>4
```

# Les variables

Tout identificateur commence par « **\$** »

Les affectations sont réalisées grâce à « **=** »

Le **typage** est **dynamique**. Les types sont :

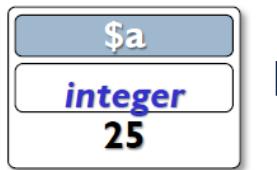
- ▶ Numérique entier : **12** ou réel : **1.54**
- ▶ Chaîne: "**Hello**" ou '**Bonjour**'
- ▶ Booléen: **true, false**
- ▶ Tableau: **\$tab[2]=12**
- ▶ Objet
- ▶ Ressource
- ▶ **NULL**
- ▶ Callable

Les variables ne sont pas explicitement déclarées: La « déclaration » d'une variable correspond à sa première affectation: **\$test = 12**

# Les variables

- ▶ Le cast :`$variable = (nom_du_type) valeur;`

`$a = (integer) "25" ; =>`



- ▶ Définition d'une constante:

`define (« nom_constante », "valeur");`

- ▶ Chaine de caractère:

guillemets doubles -> interprétation,

guillemets simples -> pas interprétation

Guillemets doubles

`$a="chaîne" ;  
$b="voici une $a";` **voici une chaîne**

Guillemets simples

`$a='chaîne' ;  
$b='voici une $a' ;` **voici une \$a**

`$d="voici une $a";  
$a="chaîne" ;`  
Guillemets simples

**voici une \$a**

# Les variables prédéfinies

**\$GLOBALS** : Contient le nom et la valeur de toutes les variables globales du script. Les noms des variables sont les clés de ce tableau.

**\$\_COOKIE**: Contient le nom et la valeur des cookies enregistrés sur le poste client. Les noms des cookies sont les clés de ce tableau .

**\$\_ENV**: Contient le nom et la valeur des variables d'environnement qui sont changeantes selon les serveurs.

**\$\_FILES** :Contient le nom des fichiers téléchargés à partir du poste client.

**\$\_GET**: Contient le nom et la valeur des données issues d'un formulaire envoyé par la méthode GET. Les noms des champs du formulaire sont les clés de ce tableau.

**\$\_POST**: Contient le nom et la valeur des données issues d'un formulaire envoyé par la méthode POST. Les noms des champs du formulaire sont les clés de ce tableau.

**\$\_REQUEST**: Contient l'ensemble des variables superglobales\$\_GET, \$\_POST, \$\_COOKIE et \$\_FILES.

**\$\_SESSION** :Contientl'ensemble des noms des variables de session et leurs valeurs.

**\$\_SERVER**: Contient les informations liées au serveur Web, tel le contenu des en-têtes HTTP ou le nom du script en cours d'exécution. Retenons les variables suivantes :

**\$\_SERVER["HTTP\_ACCEPT\_LANGUAGE"]**: qui contient le code de langue du navigateur client.

**\$\_SERVER["HTTP\_COOKIE"]**: qui contient le nom et la valeur des cookies lus sur le poste client.

**\$\_SERVER["HTTP\_HOST"]**: qui donne le nom de domaine.

**\$\_SERVER["SERVER\_ADDR"]**:qui indique l'adresse IP du serveur.

**\$\_SERVER["PHP\_SELF"]**: qui contient le nom du script en cours. Nous l'utiliserons souvent dans les formulaires.

**\$\_SERVER['REQUEST\_METHOD']** :détermine si la requête utilise la méthode POST ou GET.

# Exemples fonctions de gestion des variables

- int intval(variable), //Convertit une chaîne en entier
- float floatval(variable),// Convertit une chaîne en nombre à virgule flottante
- string strval(variable). // Convertit une variable en chaine
- empty // Détermine si une variable est vide
- var\_dump // Affiche les informations d'une variable
- print\_r // Affiche des informations lisibles pour une variable

# Existence de variables

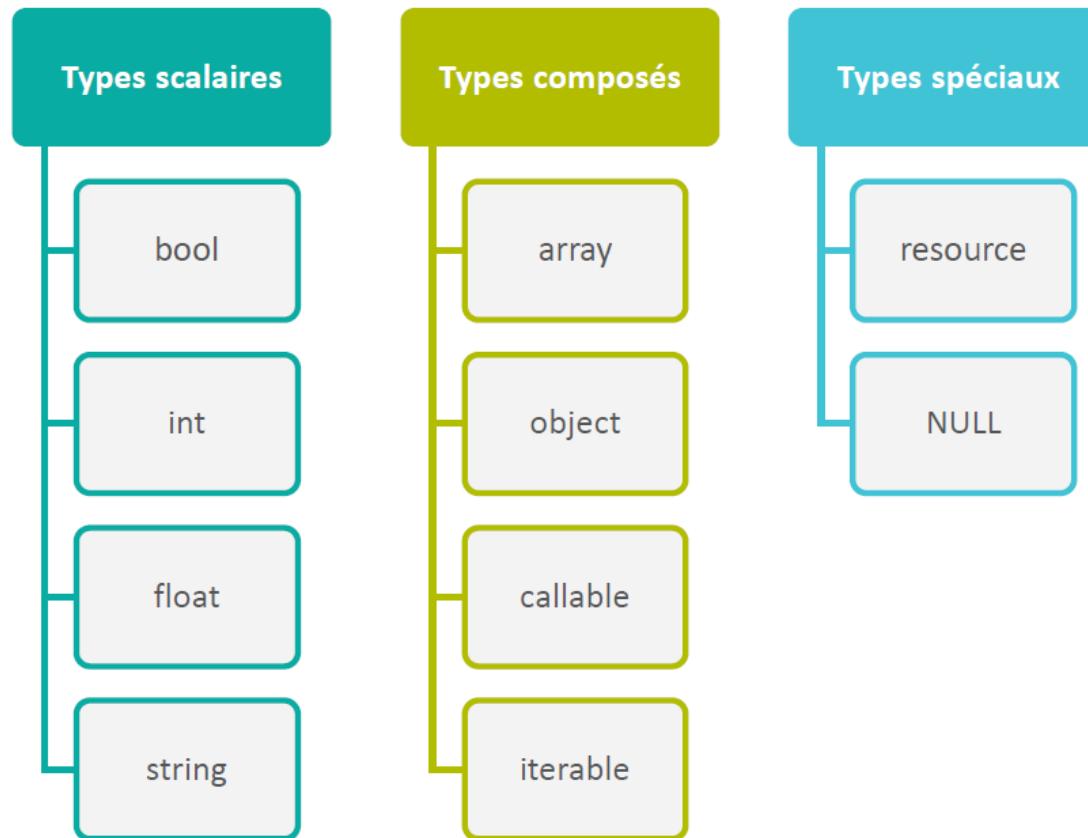
**isset()**: vérifier si une variable est définie et si elle n'est pas nulle. Elle renvoie true si la variable existe et a une valeur autre que null, sinon elle renvoie false

**unset()** : détruire une variable spécifiée

```
$a = "une variable en PHP";
if (isset($a)) echo "la variable a existe";
unset($a);
echo "la variable a a été supprimée ...";
```

# Types de données en php

PHP supporte 10 types basiques :



# Manipulation des types de données

Afin de connaître le type des variables, un certain nombre de fonctions sont disponibles et retournent true ou false :

- **is\_array()**,
- **is\_double(), is\_float(), is\_real()**
- **is\_long(), is\_int(), is\_integer()**
- **is\_string()**,
- **is\_object()**,
- **is\_null()**,
- **is\_scalar()** - si la variable est scalaire, c'est à dire si c'est un entier, une chaîne, ou un double.
- **is\_numeric()** - si la variable est un nombre ou une chaîne numérique,

```
$variable = 123;
if (is_int($variable)) {
    echo "La variable est un entier.";
} elseif (is_float($variable)) {
    echo "La variable est un nombre à virgule flottante.";
} elseif (is_string($variable)) {
    echo "La variable est une chaîne de caractères.";
} else {
    echo "La variable n'est pas un type de données courant.";
}
```

# Les fonctions gettype( ) et settype( )

gettype( ) renvoie l'un des résultats suivants:

- integer
- double
- string
- array
- object
- class
- « unknown type »

settype( ) change le type d'un élément

```
$a=3.5;  
settype($a,"integer");  
echo "le contenu de la variable a est ".$a;
```



le contenu de la variable a est 3

# Instructions de sortie

Trois fonctions permettent d'envoyer du texte vers le navigateur :

**echo(string ...\$expressions)** : void Affiche une chaîne de caractères.

**print(string \$expression)** : int → Affiche une chaîne de caractères.

**printf(string \$format, mixed ...\$values)** : int → Affiche une chaîne de caractères formatée.

**Echo :**

- N'a pas de valeur de retour.
- Peut prendre plusieurs arguments (bien que cette utilisation soit rare).
- Est légèrement plus rapide que print.

**Print :**

- Renvoie la valeur 1, il peut donc être utilisé dans des expressions.
- Peut prendre un seul argument.

**Le type mixed :**

- La valeur peut être de n'importe quelle valeur.

```
// Utilisation de echo avec et sans parenthèses
echo "Bonjour, ";
echo ("monde!");
echo "<br>"; // Saut de ligne en HTML
```

```
// Utilisation de print avec et sans parenthèses
print "Bonjour, ";
print("monde!");
```

```
$name = "sara";
$age = 30;
printf("Bonjour, je m'appelle %s et j'ai %d ans.", $name, $age);
// Affiche Bonjour, je m'appelle sara et j'ai 30 ans.
```

# Les opérateurs: Concaténation de chaînes

Permet d'assembler plusieurs chaînes

Réalisé grâce à l'opérateur point : « . »

```
"Bonjour" . "HAMED"  
                    → vaut "Bonjour AHMED"
```

```
$nb = 6*2;  
"Acheter" . $nb . " oeufs"  
                    → vaut "Acheter 12 oeufs"
```

# Les opérateurs arithmétiques

$\$a + \$b$	Somme
$\$a - \$b$	Différence
$\$a * \$b$	Multiplication
$\$a / \$b$	Division
$\$a \% \$b$	Modulo (Reste de la division entière)

# Les opérateurs de comparaison

<code>\$a == \$b</code>	Vrai si égalité entre les valeurs de <code>\$a</code> et <code>\$b</code>
<code>\$a != \$b</code>	Vrai si différence entre les valeurs de <code>\$a</code> et <code>\$b</code>
<code>\$a &lt; \$b</code>	Vrai si <code>\$a</code> inférieur à <code>\$b</code>
<code>\$a &gt; \$b</code>	Vrai si <code>\$a</code> supérieur à <code>\$b</code>
<code>\$a &lt;= \$b</code>	Vrai si <code>\$a</code> inférieur ou égal à <code>\$b</code>
<code>\$a &gt;= \$b</code>	Vrai si <code>\$a</code> supérieur ou égal à <code>\$b</code>
<code>\$a === \$b</code>	Vrai si <code>\$a</code> et <code>\$b</code> identiques (valeur et type)
<code>\$a !== \$b</code>	Vrai si <code>\$a</code> et <code>\$b</code> différents (valeur ou type)

# Les opérateurs logiques

[Expr1] <b>and</b> [Expr2]	Vrai si [Expr1] et [Expr2] sont vraies
[Expr1] <b>&amp;&amp;</b> [Expr2]	idem
[Expr1] <b>Or</b> [Expr2]	Vrai si [Expr1] ou [Expr2] sont vraies
[Expr1] <b>  </b> [Expr2]	idem
[Expr1] <b>XOr</b> [Expr2]	Vrai si [Expr1] ou [Expr2] sont vraies mais pas les deux
! [Expr1]	Vrai si [Expr1] est non vraie

# structures alternative

## L'instruction if else

- Si l'expression vaut true, PHP exécutera l'instruction et si elle vaut false, l'instruction sera ignorée.
- Les instructions après le else ne sont exécutées que si l'expression du if est false.

```
$prix = 55;
if ($prix > 100) {
    echo "<b> Pour un montant d'achat de $prix &euro;, la remise est de 10 % </b><br>";
    echo "Le prix net est de " . $prix * 0.90;
} else {
    echo "<b>Pour un montant d'achat de $prix &euro;, la remise est de 5 % </b>";
    echo "<h3> Le prix net est de " . $prix * 0.95 . "</h3>";
}
```

```
if (expression)
    commandes
elseif (expression)
    commandes
elseif (expression)
    commandes
elseif (expression)
    commandes
else
    commandes
```

## L'instruction elseif

- L'expression elseif est exécutée seulement si le if précédent et tout autre elseif précédent sont évalués comme false, et que votre elseif est évalué à true.
- Le « elseif » et « else if » sont traités de la même façon seulement quand des accolades sont utilisées. (Exemple : `else if ($a == $b){ echo $a." égal ".$b;}` est une synthèse correcte, tandis que `else if ($a == $b) : echo $a." égal ".$b;` est une synthèse qui génère une erreur PHP. De la même façon, les deux syntaxes `elseif ($a == $b){ echo $a." égal ".$b;}` et `elseif ($a == $b) : echo $a." égal ".$b;` sont toutes les deux correctes.

# structures alternative

## L'instruction switch

- L'instruction switch équivaut à une série d'instructions if.
- Un cas spécial est default. Ce cas est utilisé lorsque tous les autres cas ont échoué.
- PHP continue d'exécuter les instructions jusqu'à la fin du bloc d'instructions du switch, ou bien dès qu'il trouve l'instruction break.
- Dans une commande switch, une condition n'est évaluée qu'une fois, et le résultat est comparé à chaque case.

Exemples :

```
$i = 0;
switch ($i) {
    case 0:
        echo "i égal 0";
        break;
    case 1:
        echo "i égal 1";
        break;
    case 2:
        echo "i égal 2";
        break;
    default:
        echo "i n'est ni égal à 2, ni à 1, ni 0";
}
```

# structures alternative

## L'instruction match

- De la même manière qu'une instruction switch, l'instruction match a une expression de sujet qui est comparée à plusieurs alternatives.
- Différences entre match et switch :
  - match évaluera une valeur un peu comme les expressions ternaires.
  - la comparaison match est un contrôle d'identité (==>) plutôt qu'un contrôle d'égalité faible (==) comme switch.
  - match renvoie une valeur.

```
$fruit = "pomme";
$result = match ($fruit) {
    'pomme' => "C'est une pomme.",
    'orange', 'banane' => "C'est une orange ou une banane.",
    default => "C'est autre chose."
};
echo $result;
// Affiche C'est une pomme.
```

# structures répétitives

## L'instruction while

- PHP exécute l'instruction tant que l'expression de la boucle while est évaluée comme true. Si l'expression du while est false avant la première itération, l'instruction ne sera jamais exécutée.

```
while (expression) :  
    commandes  
endwhile;
```

## L'instruction do-while

- La principale différence par rapport à la boucle while est que la première itération de la boucle do-while est toujours exécutée.

```
do  
    commandes  
while (expression);
```

# structures répétitives

## L'instruction for

- expr1 : est évaluée (exécutée), au début de la boucle.
- expr2 : est évaluée, au début de chaque itération. Si l'évaluation vaut true, la boucle continue et les commandes sont exécutées. Si l'évaluation vaut false, l'exécution de la boucle s'arrête.
- expr3 : est évaluée (exécutée), à la fin de chaque itération.
- Les expressions peuvent éventuellement être laissées vides ou peuvent contenir plusieurs expressions séparées par des virgules.

```
for (expr1; expr2; expr3)
    commandes
```

## L'instruction foreach

- foreach ne fonctionne que pour les tableaux et les objets.
- La forme suivante passe en revue le tableau iterable\_expression. À chaque itération, la valeur de l'élément courant est assignée à \$value.

```
foreach (iterable_expression as $value){
    //commandes
}
```

- La forme suivante assignera en plus la clé de l'élément courant à la variable \$key à chaque itération.

```
foreach (iterable_expression as $key =>
    $value){
    //commandes
}
```

# Ruptures de séquence (goto)

**goto** : permet de transférer le contrôle d'exécution du programme à une autre partie du code marquée par une étiquette spécifique

```
<?php
$i = 0;
start: // Étiquette
echo $i . "\n";
$i++;

if ($i < 5) {
    goto start; // Transfère le contrôle à l'étiquette 'start' si $i est inférieur à 5
}
// Affiche : 0 1 2 3 4
```

# Ruptures de séquence (break)

L'instruction **break** est utilisée pour sortir immédiatement de la boucle la plus proche. Une fois que break est rencontré, l'exécution de la boucle est terminée, et le contrôle passe à l'instruction suivant la boucle.

```
<?php
// Utilisation de break dans une boucle for
for ($i = 0; $i < 10; $i++) {
    echo $i . " ";
    if ($i === 5) {
        |   break; // Sort de la boucle lorsque $i atteint 5
    }
}
// Affiche : 0 1 2 3 4 5
?>
```

# Ruptures de séquence (continue)

L'instruction **continue** est utilisée pour passer à l'itération suivante de la boucle sans exécuter le reste du code dans la boucle pour cette itération particulière. Cela signifie que le code après continue dans la boucle ne sera pas exécuté pour cette itération, mais la boucle continue avec l'itération suivante.

```
<?php
// Utilisation de continue dans une boucle for
for ($i = 0; $i < 5; $i++) {
    if ($i === 2) {
        continue; // Passe à l'itération suivante lorsque $i est égal à 2
    }
    echo $i . " ";
}
// Affiche : 0 1 3 4
?>
```

# Tableaux «classiques» : indexés

Création / initialisation:

```
$tab1 = array(12, "fraise", 2.5) ;  
  
$tab1_bis = [ 12, "fraise", 2.5 ] ;  
  
$tab2[] = 12 ;  
$tab2[] = "fraise" ;  
$tab2[] = 2.5 ;  
  
$tab3[0] = 12 ;  
$tab3[1] = "fraise" ;  
$tab3[2] = 2.5 ;
```

Clé	Valeur
0	12
1	"fraise"
2	2.5

# Tableaux associatifs : syntaxe

```
$tab5['un'] = 12 ;
$tab5['trois'] = "fraise" ;
$tab5["deux"] = 2.5 ;
$tab5[42] = "el5" ;

$tab6 = array( 'un' => 12,
              'trois' => "fraise",
              "deux" => 2.5,
              42      => "el5") ;

$tab7 = [ 'un'    => 12, 'trois' => "fraise",
          "deux" => 2.5, 42      => "el5" ] ;
```

Clé	Valeur
"un"	12
"trois"	"fraise"
"deux"	2.5
42	"el5"

# Parcours de tableau: foreach

- Des tableaux associatifs ne peuvent PAS être parcourus grâce aux indices des cases contenant les éléments...

```
foreach ($tableau as $element)
{
    /* Bloc d'instructions répété pour chaque élément de
    $tableau */
    /* Chaque élément de $tableau est accessible grâce à
    $element */
}
```

# Parcours de tableau : foreach

```
...
$tab4[0] = 12 ;
$tab4[6] =
    "fraise" ;
$tab4[2] = 2.5 ;
$tab4[5] = "el5" ;
foreach($tab4 as
    $v)
{
    echo "<p>Val:
$v\n";
}
```



```
Val: 12
Val: fraise
Val: 2.5
Val: el5
```

# Parcours de tableau : foreach

```
foreach($tableau as $cle=>$element)
{
    /* Bloc d'instructions répété pour
    chaque élément de $tableau */
    /* Chaque élément de $tableau est
    accessible grâce à $element*/
    /* La clé d'accès à chaque élément est
    donnée par $cle*/
}
```

```
$tab4[0] = 12;
$tab4[6] = "fraise";
$tab4[2] = 2.5;
$tab4[5] = "el5";
foreach ($tab4 as $cle => $v) {
    echo " cle: $cle Val: $v";
}
```



cle: 0 Val: 12

cle: 6 Val: fraise

cle: 2 Val: 2.5

cle: 5 Val: el5

# Opérations tableau

**sort(\$tab)** : Trier d'un tableau

```
$tab = [3, 1, 2];
sort($tab);
print_r($tab); // Affichera : [1, 2, 3]
```

**void asort(array \$tab)** : Trier un tableau associatif par ses valeurs

```
$tab = ['b' => 3, 'k' => 1, 'c' => 2];
asort($tab);
print_r($tab); // Affichera : ['K' => 1, 'c' => 2, 'b' => 3]
```

**ksort():** trier un tableau associatif par ses clés

```
$tab = array("d" => 4, "a" => 1, "b" => 2, "c" => 3);
ksort($tab);
print_r($tab); // Affichera : Array ( [a] => 1 [b] => 2 [c] => 3 [d] => 4 )
```

**array\_reverse(\$tab)** : Inverser l'ordre des éléments d'un tableau

```
$tab = [1, 2, 3];
$reversed_tab = array_reverse($tab);
print_r($reversed_tab); // Affichera : [3, 2, 1]
```

# Opérations tableau

**array\_count\_values(array \$tab)** : Compter les occurrences des valeurs dans un tableau

```
$tab = ['a', 'b', 'a', 'c', 'a'];
$counted_values = array_count_values($tab);
print_r($counted_values); // Affichera : ['a' => 3, 'b' => 1, 'c' => 1]
```

**array\_filter(array \$tab, string nom\_fonction)** : Filtrer un tableau

```
$tab = [1, 2, 3, 4, 5];
$filtered_tab = array_filter($tab, function ($element) {
    return $element % 2 == 0; // Retourne true pour les nombres pairs
});
print_r($filtered_tab); // Affichera : [2, 4]
```

**array\_keys(array \$input)** : Retourner toutes les clés d'un tableau

```
$input = ['a' => 1, 'b' => 2, 'c' => 3];
$keys = array_keys($input);
print_r($keys); // Affichera : ['a', 'b', 'c']
```

# Opérations tableau

**array\_values(array \$input)** : Retourner toutes les valeurs d'un tableau

```
$input = [ 'a' => 1, 'b' => 2, 'c' => 3];
$values = array_values($input);
print_r($values); // Affichera : [1, 2, 3]
```

**string implode(string \$glue, array \$pieces)** : Rassembler les éléments d'un tableau

```
$tab = ['apple', 'banana', 'orange'];
$res = implode(':', $tab);
echo $res; // Affichera : "apple:banana:orange"
```

**explode()**: diviser une chaîne de caractères en un tableau en utilisant un séparateur spécifié

```
$string = "pomme banane orange";
$tab = explode(" ", $string);
print_r($tab); // Affichera : Array ( [0] => pomme [1] => banane [2] => orange )
```

**bool in\_array(\$v, \$tab)** : Vérifier si une valeur \$v existe dans le tableau \$tab.

```
$tab = ['apple', 'banana', 'orange'];
$is_present = in_array('banana', $tab);
var_dump($is_present); // Affichera : bool(true)
```

# Opérations tableau

**array\_slice()** : extraire une portion d'un tableau

```
$tab = array("a", "b", "c", "d", "e");
$slice = array_slice($tab, 2, 2);
print_r($slice); // Affichera : Array ( [0] => c [1] => d )
```

**array\_push()**: ajouter un ou plusieurs éléments à la fin d'un tableau

```
$tab = array("a", "b", "c");
array_push($tab, "d", "e");
print_r($tab); // Affichera : Array ( [0] => a [1] => b [2] => c [3] => d [4] => e )
```

**array\_unshift()** :ajouter un ou plusieurs éléments au début d'un tableau.

```
$tab = array("c", "d", "e");
array_unshift($tab, "a", "b");
print_r($tab); // Affichera : Array ( [0] => a [1] => b [2] => c [3] => d [4] => e )
```

**array\_pop()**: extraire et retourner le dernier élément d'un tableau

```
$tab = array("a", "b", "c", "d");
$last_element = array_pop($tab);
print_r($tab); // Affichera : Array ( [0] => a [1] => b [2] => c )
echo $last_element; // Affichera : d
```

```
array array_intersect($tab1,$tab2)
```

Cette fonction retourne un tableau contenant tous les éléments communs aux tableaux \$tab1 et \$tab2.

```
array_diff($tab1,$tab2)
```

retourne un tableau contenant les éléments présents dans le premier paramètre mais pas dans le second

# Opérations tableau

## **array\_chunk**

Sépare un tableau en tableaux de taille inférieure

## **array\_diff\_key**

Calcule la différence de deux tableaux en utilisant les clés pour comparaison

## **array\_diff**

Calcule la différence entre des tableaux

## **array\_fill\_keys**

Remplit un tableau avec des valeurs, en spécifiant les clés

## **array\_key\_first**

Récupère la première clé d'un tableau

## **array\_merge**

Fusionne plusieurs tableaux en un seul

## **array\_search**

Recherche dans un tableau la première clé associée à la valeur

# Constantes pré-définies

## CASE\_LOWER (int)

est utilisée avec `array_change_key_case()` et sert à convertir tous les index d'un tableau en minuscules. C'est aussi le comportement par défaut de `array_change_key_case()`

## CASE\_UPPER (int)

est utilisée avec `array_change_key_case()` et sert à convertir tous les index d'un tableau en majuscules

## SORT\_ASC (int)

est utilisée avec `array_multisort()` pour trier en ordre ascendant

## SORT\_DESC (int)

est utilisée avec `array_multisort()` pour trier en ordre descendant

## ARRAY\_FILTER\_USE\_KEY (int)

est utilisé avec `array_filter()` pour passer chaque clé comme premier argument à la fonction de rappel fournie

## ARRAY\_FILTER\_USE\_BOTH (int)

est utilisé avec `array_filter()` pour passer la valeur et la clé à la fonction de rappel fournie

# Opérateurs de tableaux

Exemple	Nom	Résultat
<code>\$a + \$b</code>	Union	Union de <code>\$a</code> et <code>\$b</code> .
<code>\$a == \$b</code>	Égalité	<b>true</b> si <code>\$a</code> et <code>\$b</code> contiennent les mêmes paires clés/valeurs.
<code>\$a === \$b</code>	Identique	<b>true</b> si <code>\$a</code> et <code>\$b</code> contiennent les mêmes paires clés/valeurs dans le même ordre et du même type.
<code>\$a != \$b</code>	Inégalité	<b>true</b> si <code>\$a</code> n'est pas égal à <code>\$b</code> .
<code>\$a &lt;&gt; \$b</code>	Inégalité	<b>true</b> si <code>\$a</code> n'est pas égal à <code>\$b</code> .
<code>\$a !== \$b</code>	Non-identique	<b>true</b> si <code>\$a</code> n'est pas identique à <code>\$b</code> .

# Traitement des données de formulaires

PHP permet de **traiter les données** saisies grâce à un formulaire HTML si le champ ACTION du formulaire désigne une page PHP du serveur.

Après récupération par le serveur Web, les données sont contenues dans l'une des variables superglobales de type tableau associatif **\$ GET** ou **\$ POST** selon la méthode de la requête (ou **\$\_REQUEST** qui reçoit le contenu de **\$\_GET** et **\$\_POST**).

La valeur peut être trouvée grâce à une Clé **qui porte le même nom** que le champ du formulaire de la page HTML de saisie.

# Exemple –Formulaire HTML

```
<!doctype html>
<html ">
  <head>
    <title>formulaire</title>
  </head>
  <body>
    <form action="valide1.php"
method="get">
      Nom: <input type="text"
name="nomPers">
      <input type="submit"
value="Valider">
    </form>
  </body>
</html>
```

# Exemple – Traitement en PHP

```
<?php
<!doctype html>
<html>
<head><title>bonjour</title></head>
<body>
if (isset($_GET['nomPers']))
    if (!empty($_GET['nomPers']))
        $html .= "Bonjour\n".
$_GET['nomPers'] . "\n";
    else
        $html .= "Aucune valeur saisie\n";
else
    $html .= "Utilisation incorrecte\n";
echo $html . "</body>\n</html>" ;
```

`$_GET['nomPers']`  
est-il défini ?

`$_GET['nomPers']`  
est-il vide ?

# Récupération des valeurs multiples

Formulaires contenant des champs « SELECT »

Choisissez des fruits:

- Fraise
- Pomme
- Poire
- Banane
- Cerise

« SELECT unique »

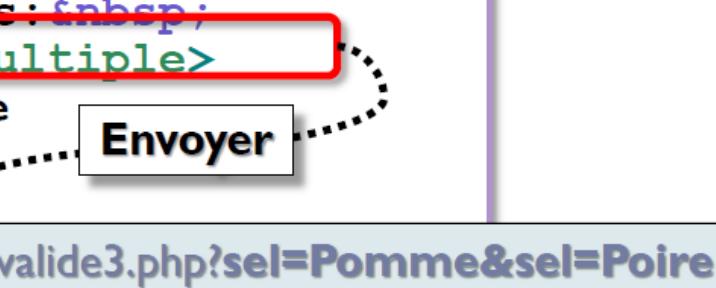
```
<!doctype html>
<html lang="fr">
<head>
  <title>Formulaire de saisie des
  fruits</title>
</head>
<body>
  <form action="valide3.php" method="get">
    Choisissez des fruits: 
    <select name="sel">
      <option>Fraise
      <option>Pomme
      <option>Poire
      <option>Banane
      <option>Cerise
    </select>
    <input type="submit" value="envoyer">
  </form>
</body>
</html>
```

valide3.php?sel=Pomme

# Récupération des valeurs multiples

```
<!doctype html>
<html lang="fr">
<head>
    <title>Formulaire de saisie des
        fruits</title>
</head>
<body>
    <form action="valide3.php" method="get">
        Choisissez des fruits: 
        <select name="sel" multiple>
            <option>Fraise
            <option>Pomme
            <option>Poire
            <option>Banane
            <option>Cerise
        </select>
        <input type="submit" value="envoyer">
    </form>
</body>
</html>
```

« SELECT multiple »



???

# Récupération des valeurs multiples

```
<!doctype html>
<html lang="fr">
<head>
    <title>Formulaire de saisie des
    fruits</title>
</head>
<body>
    <form action="valide3.php" method="get">
        Choisissez des fruits:&nbsp;
        <select name="sel[]" multiple>
            <option>Fraise
            <option>Pomme
            <option>Poire
            <option>Banane
            <option>Cerise
        </select>
        <input type="submit" value="envoyer">
    </form>
</body>
</html>
```

**SELECT multiple**

Envoyer

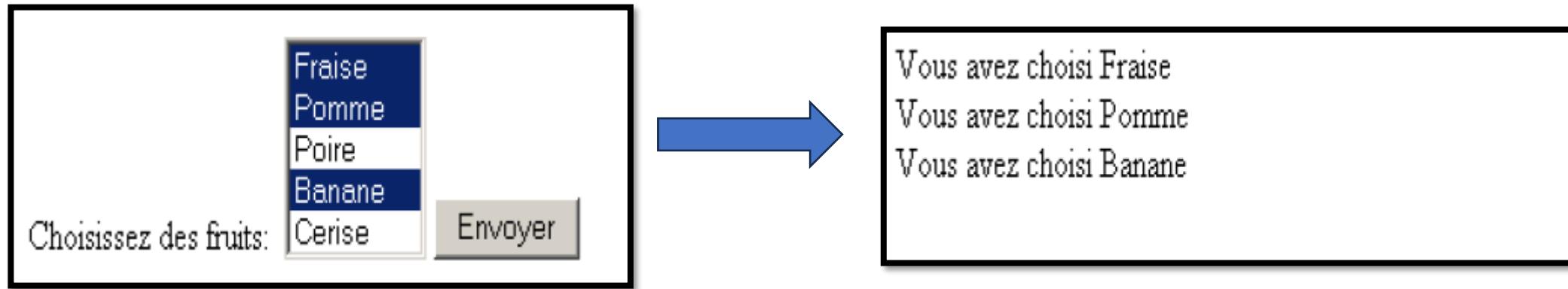
valide3.php?sel%5B%5D=Pomme&sel%5B%5D=Poire

# Récupération des valeurs multiples

```
<?php
<!doctype html>
<html>
<head>
    <title>Liste de fruits</title>
</head>
<body>
    if (isset($_GET['sel']) && is_array($_GET['sel']))
    { /* La variable $_GET['sel'] est définie et elle
        est un tableau */
        foreach($_GET['sel'] as $fruit)
            $html .= "<p>Vous avez choisi $fruit\n" ;
    }
    else
        $html .= "<p>Vous n'avez pas choisi de
        fruit\n" ;
$html .= "</body>\n</html>" ;
echo $html ;
```

\$\_GET['sel'] est un tableau

# Récupération des valeurs multiples



# Récupération des valeurs multiples

Formulaires contenant des champs «CHECKBOX»

Choisissez des fruits:

Fraise  
 Pomme  
 Poire  
 Banane  
 Cerise

Résultat

Vous avez choisi Fraise  
Vous avez choisi Pomme  
Vous avez choisi Banane

```
<!doctype html>
<html>
  <head>
    <title>Formulaire de saisie des fruits</title>
  </head>
  <body>
    <form name="formu" action="valide3.php" method="get">
      Choisissez des fruits ;
      <label><input type="checkbox" name="sel[]" value="Fraise">Fraise</label>
      <label><input type="checkbox" name="sel[]" value="Pomme">Pomme</label>
      <label><input type="checkbox" name="sel[]" value="Poire">Poire</label>
      <label><input type="checkbox" name="sel[]" value="Banane">Banane</label>
      <label><input type="checkbox" name="sel[]" value="Cerise">Cerise</label>
      <input type="submit" value="Envoyer">
    </form>
  </body>
</html>
```

# Variables d'environnement

## \$LSERVER

\$ SERVER est un tableau contenant des informations comme les en-têtes, dossiers et chemins du script.

### **PHP\_SELF**

Le nom du fichier du script en cours d'exécution, par rapport à la racine web

### **GATEWAY\_INTERFACE**

Numéro de révision de l'interface CGI du serveur

### **SERVER\_ADDR**

L'adresse IP du serveur sous lequel le script courant est en train d'être exécuté

### **SERVER\_NAME**

Le nom du serveur hôte qui exécute le script suivant

### **SERVER\_SOFTWARE**

Chaîne d'identification du serveur, qui est donnée dans les en-têtes lors de la réponse aux requêtes

### **REQUEST\_METHOD**

Méthode de requête utilisée pour accéder à la page

### **HTTP\_ACCEPT\_LANGUAGE**

Contenu de l'en-tête Accept-Language: de la requête courante, si elle existe

### **HTTP\_CONNECTION**

Contenu de l'en-tête Connection: de la requête courante, si elle existe

### **HTTP\_HOST**

Contenu de l'en-tête Host: de la requête courante, si elle existe

### **HTTPS**

Défini à une valeur non-vide si le script a été appelé via le protocole HTTPS

### **REMOTE\_PORT**

Le port utilisé par la machine cliente pour communiquer avec le serveur web

### **REMOTE\_USER**

L'utilisateur authentifié

# Variable prédéfinies

`$_FILES` - Variable de téléchargement de fichier via HTTP :

- `$_FILES['nom_de_la_variable']['name']` : Le nom original du fichier qui provient de la machine du client
- `$_FILES['nom_de_la_variable']['type']` : Le type MIME du fichier
- `$_FILES['nom_de_la_variable']['size']` : La taille du fichier en bytes (soit 8 bits ou un octet)
- `$_FILES['nom_de_la_variable']['tmp_name']` : Le nom temporaire du fichier stocké sur le serveur
- `$_FILES['nom_de_la_variable']['error']` : Le code erreur associé à l'upload

`$_SESSION` - Variables de session. Liste des fonctions dans <https://www.php.net/manual/fr/ref.session.php>

`$_ENV` - Variables d'environnement

`$_COOKIE` - Cookies HTTP

# Transfert de fichier vers le serveur

```
<form action="demo1.php" method="post" enctype="multipart/form-data">
    <fieldset>
        |   |   |
        |   |   |   | <legend><B> Transfert de fichier </B></legend>
        |   |   |
        |   |   |   | <table>
        |   |   |   |       <tr>
        |   |   |   |           <th> Fichier </th>
        |   |   |   |           <td> <input type="file" name="fich" accept="image/gif" size="50" /></td>
        |   |   |   |       </tr>
        |   |   |   |       <tr>
        |   |   |   |           <th> envoyer ! </th>
        |   |   |   |           <td> <input type="submit" value="Envoi" />
        |   |   |   |           </td>
        |   |   |   |       </tr>
        |   |   |
        |   |   | </table>
    </fieldset>
</form>
```

Transfert de fichier

Fichier  doc.gif

envoyer

# Transfert de fichier vers le serveur

```
if (isset($_FILES['fich'])) {
    foreach ($_FILES["fich"] as $cle => $valeur) {
        echo "clé: $cle valeur: $valeur <br />";
    }
    echo "<br />";
    // Enregistrement et renommage du fichier
    $result = move_uploaded_file(
        $_FILES["fich"]["tmp_name"],
        "imagephp.gif"
    );
    if ($result == true) {
        echo "<br /><big>Transfert est réalisé!</big>";
    } else {
        echo "<br /> Erreur de transfert n°" . $_FILES["fich"]["error"];
    }
}
```

clé: name valeur: doc.gif  
clé: type valeur: image/gif  
clé: tmp\_name valeur: C:\xampp\tmp\php7BB6.tmp  
clé: error valeur: 0  
clé: size valeur: 86133

Transfert est réalisé!

# Gérer les boutons d'envoi multiples

L'utilisation de plusieurs boutons submit dans un même formulaire permet de déclencher des actions différentes en fonction du bouton activé par l'utilisateur. Il est nécessaire pour cela que les boutons aient le même nom et que la sélection de l'action se fasse en fonction de la valeur associée à chaque bouton via l'attribut value

# Gérer les boutons d'envoi multiples

```
<form action="" method="post">
    <table>
        <tr>
            <td>Nombre X:</td>
            <td><input type="text" name="nb1" value=<?php if (isset($_POST['nb1'])) echo $_POST['nb1']; ?>"></td>
        </tr>
        <tr>
            <td>Nombre Y:</td>
            <td><input type="text" name="nb2" value=<?php if (isset($_POST['nb2'])) echo $_POST['nb2']; ?>"></td>
        </tr>

        <tr>
            <td colspan="2">
                <input type="submit" name="calcul" value="Addition" />
                <input type="submit" name="calcul" value="Soustraction" />
                <input type="submit" name="calcul" value="Division" />
            </td>
        </tr>
    </table>
</form>
```

# Gérer les boutons d'envoi multiples

```
if (isset($_POST["calcul"]) && isset($_POST["nb1"]) && isset($_POST["nb2"])) {
    switch ($_POST["calcul"]) {
        case "Addition":
            $resultat = $_POST["nb1"] + $_POST["nb2"];
            break;
        case "Soustraction":
            $resultat = $_POST["nb1"] - $_POST["nb2"];
            break;
        case "Division":
            if ($_POST["nb2"] != 0) {
                $resultat = $_POST["nb1"] / $_POST["nb2"];
            } else {
                $resultat = "Erreur : Division par zéro";
            }
            break;
        default:
            $resultat = "Opération inconnue";
            break;
    }
    echo "<h3>Le résultat est : {$resultat}</h3>";
} else {
    echo "<h3>Entrez deux nombres et choisissez une opération.</h3>";
}
```

# Fonctions de math

**abs(\$va/)**: retourne la valeur absolue de \$val

**acos(\$va/), cos(\$va/), sin(\$val) ...**

**exp(\$va/), log(\$va/)**

**floor(\$va/)**: retourne l'entier inférieur du nombre \$val.

**round(va/, precision)**: Retourne la valeur arrondie de num à la précision precision (nombre de chiffres après la virgule). Le paramètre precision peut être négatif ou null (sa valeur par défaut).

**ceil(val)**: Retourne l'entier supérieur du nombre num.

**max(va/l, va12), min(va/l, va12)**

**pow(val, puiss), sqrt(va/)**

**rand(), rand(min, max)**: entier aléatoire entre max et min

**intdiv** — Division entière

**Is\_finite** — Indique si un nombre est fini

**Is\_infinite** — Indique si un nombre est infini

**Is\_nan** — Indique si une valeur n'est pas un nombre

## Exemple

```
$number = 4.3;

// Arrondir à l'entier inférieur
$floorValue = floor($number);
echo "Valeur arrondie à l'entier inférieur : $floorValue<br>";

// Arrondir à l'entier supérieur
$ceilValue = ceil($number);
echo "Valeur arrondie à l'entier supérieur : $ceilValue<br>";

// Calcul de la puissance
$powerValue = pow($number, 2);
echo "Valeur de $number à la puissance 2 : $powerValue<br>";

// Valeur absolue
$absoluteValue = abs(-7.8);
echo "Valeur absolue de -7.8 : $absoluteValue<br>";

// Valeur maximale
$maxValue = max(10, 20, 5, 30);
echo "Valeur maximale : $maxValue<br>";

// Arrondir à l'entier le plus proche
$roundedValue = round(3.7);
echo "Valeur arrondie de 3.7 : $roundedValue<br>";

// Génération d'un nombre aléatoire entre 1 et 100
$randomNumber = rand(1, 100);
echo "Nombre aléatoire : $randomNumber";
```

Valeur arrondie à l'entier inférieur : 4

Valeur arrondie à l'entier supérieur : 5

Valeur de 4.3 à la puissance 2 : 18.49

Valeur absolue de -7.8 : 7.8

Valeur maximale : 30

Valeur arrondie de 3.7 : 4

Nombre aléatoire : 99

# Les chaînes de caractères

**Strlen:** Calculer la taille d'une chaîne

**trim:** Supprimer les espaces (ou d'autres caractères) en début et fin de chaîne

**ltrim:** Supprimer à gauche

**rtrim:** Supprimer à droite

```
$str = " Ceci est une chaîne avec des espaces ";
$str = trim($str);

echo $str; // Affiche "Ceci est une chaîne avec des espaces"
```

**str\_word\_count:** Compter le nombre de mots utilisés dans une chaîne

```
$string = "Ceci est un exemple de phrase";
$word_count = str_word_count($string);
echo "Nombre de mots dans la phrase : " . $word_count;
// Résultat : Nombre de mots dans la phrase : 6
```

**strtoupper/ strtolower:** Renvoyer la chaîne en majuscule / en minuscule

**ucfirst/ lcfirst:** Mettre le premier caractère en majuscule / en minuscule

```
$string = "exemples de phrases";
$ucfirst = ucfirst($string);
$lcfirst = lcfirst($string);
echo "Première lettre en majuscule : " . $ucfirst . "<br>";
echo "Première lettre en minuscule : " . $lcfirst;
// Résultat :
// Première lettre en majuscule : Exemples de phrases
// Première lettre en minuscule : exemples de phrases
```

# Les chaînes de caractères

**ucwords:** Mettre en majuscule la première lettre de tous les mots

```
$string = "exemples de phrases";
$ucwords = ucwords($string);
echo "Première lettre de tous les mots en majuscule : " . $ucwords;
// Résultat : Première lettre de tous les mots en majuscule : Exemples De Phrases
```

**strpos:** Chercher la position de la première occurrence dans une chaîne

```
$string = "Bonjour tout le monde";
$position = strpos($string, "tout");
echo "Position de 'tout' : " . $position;
// Résultat : Position de 'tout' : 8
```

**stripos:** chercher la position de la première occurrence dans une chaîne, sans tenir compte de la casse

```
$string = "Bonjour tout le monde";
$position = stripos($string, "TOUT");
echo "Position de 'TOUT' (sans tenir compte de la casse) : " . $position;
// Résultat : Position de 'TOUT' (sans tenir compte de la casse) : 8
```

**substr(\$string, \$indiceDebut, \$taille):** Retourner un segment de chaîne à partir de l'indice de début donné jusqu'à la fin de la chaîne.

\$taille: (optionnel) le nombre de caractères à découper

```
$string = "Exemple de texte à extraire";
$substring = substr($string, 8); // à partir de l'index 8 jusqu'à la fin
echo "Extrait : " . $substring;
// Résultat : Extrait : texte à extraire
```

# Les chaînes de caractères

**str\_replace(\$chaine\_recherchée, \$remplacement, \$chaine)** Remplacer toutes les occurrences de \$chaine\_recherchée dans une chaîne \$chaine par \$remplacement

```
$chaine = "Bonjour tout le monde";
$nouvelle_chaine = str_replace("tout", "chacun", $chaine);
echo "Nouvelle chaîne : " . $nouvelle_chaine;
// Résultat : Nouvelle chaîne : Bonjour chacun le monde
```

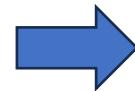
**str\_ireplace:** Version insensible à la casse de str\_replace()

**preg\_match(\$pattern, \$subject):** Effectuer une recherche de correspondance avec une expression rationnelle standard

**preg\_match\_all():** chercher toutes les occurrences correspondantes dans la

```
$texte = " numéro1 est 123-456-7890 et numéro2 est 987-654-3210.";
$pattern = "/\d{3}-\d{3}-\d{4}/";
if (preg_match_all($pattern, $texte, $resultats)) {
    echo "Numéros de téléphone trouvés : <br>";
    foreach ($resultats[0] as $match) {
        echo $match . "<br>";
    }
} else {
    echo "Aucun numéro de téléphone trouvé.";
}
```

```
$subject = "Bonjour tout le monde";
$pattern = "/tout/";
if (preg_match($pattern, $subject)) {
    echo "Correspondance trouvée !";
} else {
    echo "Aucune correspondance trouvée.";
}
// Résultat : Correspondance trouvée !
```



```
Numéros de téléphone trouvés
123-456-7890
987-654-3210
```

# Les chaînes de caractères

**str\_pad:** Compléter une chaîne jusqu'à une taille donnée

```
$chaine = "Bonjour";
$caractere_special = "*";
$nouvelle_chaine = str_pad($chaine, 15, $caractere_special, STR_PAD_RIGHT);
echo "Nouvelle chaîne : " . $nouvelle_chaine;
// Résultat : Nouvelle chaîne : Bonjour*****
```

**Nl2br:** Insérer un retour à la ligne HTML à chaque nouvelle ligne

```
$texte = "Je suis\nun retour à la ligne.";
$texte_corrigé = nl2br($texte);
echo $texte_corrigé;
```



Je suis  
un retour à la ligne.

**str\_repeat:** Répéter une chaîne

```
$chaine = "Hello ";
$repetitions = 3;
$nouvelle_chaine = str_repeat($chaine, $repetitions);
echo "Nouvelle chaîne : " . $nouvelle_chaine;
// Résultat : Nouvelle chaîne : Hello Hello Hello
```

# Fonctions sur les dates

**Format complet de la date et de l'heure :**

**Y-m-d H:i:s** : Année-mois-jour Heures:minutes:secondes

```
$format = date("Y-m-d H:i:s");
echo "Format Y-m-d H:i:s : " . $format; // Exemple: 2022-04-14 15:30:00
```

**Formats de date :**

**Y-m-d** : Année-mois-jour

**d/m/Y** : Jour/mois/année

**M j, Y** : Abréviation du mois, jour, année

**j F Y** : Jour, nom complet du mois, année

```
echo "Format Y-m-d : " . date("Y-m-d"); // Exemple: 2022-04-14
echo "Format d/m/Y : " . date("d/m/Y"); // Exemple: 14/04/2022
echo "Format M j, Y : " . date("M j, Y"); // Exemple: Apr 14, 2022
echo "Format j F Y : " . date("j F Y"); // Exemple: 14 April 2022
```

**Formats d'heure :**

**H:i:s** : Heures:minutes:secondes au format 24 heures g:i A : Heures:minutes AM/PM

```
echo "Format H:i:s : " . date("H:i:s"); // Exemple: 15:30:00
echo "Format g:i A : " . date("g:i A"); // Exemple: 3:30 PM
```

**Autres formats :**

**D, d M Y H:i:s T** : Jour de la semaine, jour mois année Heures:minutes:secondes fuseau horaire

# Fonctions sur les dates

**date(format,\$timestamp)**: formater une valeur timestamp en une date et une heure plus lisibles

```
$timestamp = time(); // Récupère le timestamp actuel  
$date_formattee = date("d.m.Y H:i:s", $timestamp);  
echo $date_formattee; // Affiche quelque chose comme "14.03.2022 14:01:49"
```

**mktime(hour, minute, second, month, day, year)**: Retourner la Valeur timestamp de la date en paramètre

```
// Crée un timestamp pour le 12 août 2022 à 11:14:54  
$timestamp = mktime(11, 14, 54, 8, 12, 2022);  
  
// Formate le timestamp en une date lisible  
$date_formattee = date("d m Y h:i:sa", $timestamp);  
  
// Affiche la date créée  
echo "La date créée : " . $date_formattee;  
// Affiche "12 08 2022 11:14:54am"
```

**Strtotime** : Créer une date à partir d'une chaîne

```
$date_chaine = "2022-03-14 14:01:49";  
$timestamp = strtotime($date_chaine);  
echo "Timestamp créé à partir de la chaîne : " . $timestamp;
```

# Fonctions sur les dates

Exemple avec **date()** et **mkttime()**

```
<?php  
    // Crée un timestamp pour demain  
    $tomorrow = mkttime(0, 0, 0, date("m"), date("d") + 1, date("Y"));  
  
    // Crée un timestamp pour le mois dernier  
    $lastmonth = mkttime(0, 0, 0, date("m") - 1, date("d"), date("Y"));  
  
    // Crée un timestamp pour l'année prochaine  
    $nextyear = mkttime(0, 0, 0, date("m"), date("d"), date("Y") + 1);  
?>
```

**Checkdate** :envoyer true si la date est valide et false sinon.

```
$month = 2; // avril  
$day = 29;  
$year = 2923;  
  
if (checkdate($month, $day, $year)) {  
    echo "La date est valide.";  
} else {  
    echo "La date n'est pas valide.";  
}
```

# Fonctions sur les dates

**cal\_days\_in\_month**:obtenir le nombre de jours dans un mois donné

```
$month = 4; // Avril  
$year = 2024;  
  
$days_in_month = cal_days_in_month(CAL_GREGORIAN, $month, $year);  
echo "Il y a $days_in_month jours en avril $year.";
```

# Fonctions utilisateur

Description d'une fonctionnalité dépendant éventuellement de paramètres et retournant éventuellement un résultat

## Définition:

```
Function moyenne($a,$b)
{
    return ($a+$b)/2. ;
}
```

## Utilisation

```
$resultat= moyenne(2,4) ;
echo $resultat; // vaut 3
```

# Fonctions utilisateur

**Passage par valeur :** copier la valeur de l'argument et l'utilisée dans la fonction, mais toute modification de la valeur à l'intérieur de la fonction n'affecte pas la valeur de l'argument à l'extérieur de la fonction.

```
function incrementer($x) {  
    $x++;  
    return $x;  
}  
  
$valeur = 10;  
echo incrementer($valeur); // Affiche 11  
echo $valeur; // Affiche 10
```

**Passage par référence :** Passer l'adresse mémoire de l'argument à la fonction. Toute modification de la valeur à l'intérieur de la fonction affecte directement la valeur de l'argument à l'extérieur de la fonction.



```
function incrementer(&$x) {  
    $x++;  
    return $x;  
}  
  
$valeur = 10;  
echo incrementer($valeur); // Affiche 11  
echo $valeur; // Affiche 10
```

# Fonctions utilisateur

**Passage par valeur par défaut** : Assigner une valeur par défaut à l'argument si aucun argument correspondant n'est passé lors de l'appel de la fonction.

```
function incrementer($x = 0) {
    return $x + 1;
}

echo incrementer(); // Affiche 1
echo incrementer(10); // Affiche 11
```

# Définition de fonctions fréquemment utilisées

- ❑ Certaines fonctions sont utilisées dans plusieurs scripts PHP
- ❑ Comment faire pour ne pas les définir dans chacune des pages ?

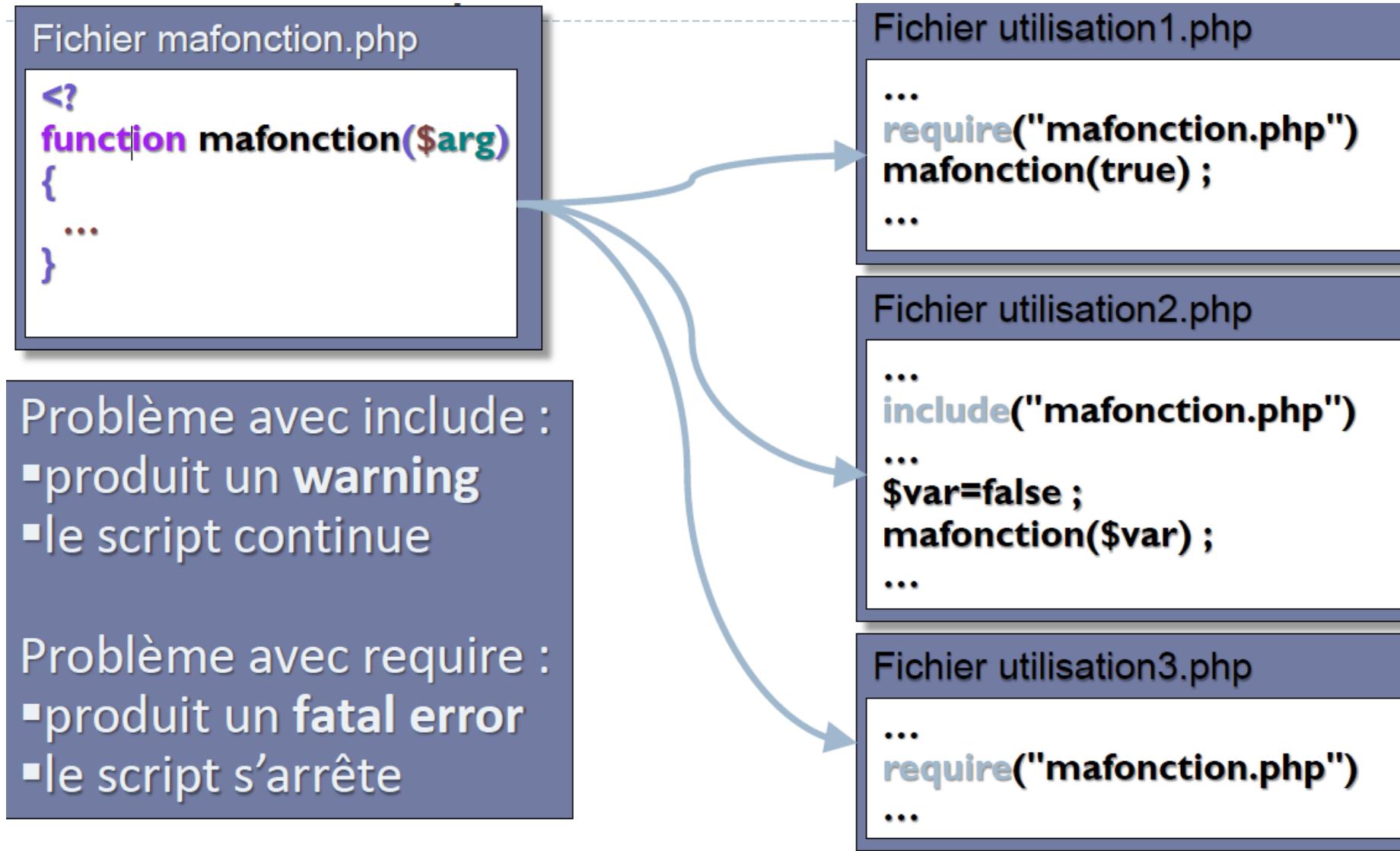
**Utilisation de :**

```
►include("fichier");  
►require("fichier");  
►include_once("fichier");  
►require_once("fichier");
```

Permet d'inclure le contenu de fichier dans le script courant **Require** erreur bloquante, **include** erreur non bloquante ...**\_once** pour les inclusions uniques

# Définition de fonctions fréquemment utilisées

## Include et require



# Gestion des erreurs

- Dans certains cas, il n'est ni possible ni utile de poursuivre l'exécution du code PHP (variables non définies, valeurs erronées, échec de connexion)
- Arrêt brutal de l'exécution du code:

`die` (message)

`exit` (message)

Envoie **message** au navigateur et termine l'exécution du script courant

```
$age = 15;  
if ($age < 18) {  
    die("Désolé, vous devez avoir au moins 18 ans pour accéder à cette page.");  
}  
echo "Bienvenue sur la page réservée aux adultes.";
```



si la variable \$age est inférieure à 18, le script s'arrête immédiatement et affiche le message "Désolé, vous devez avoir au moins 18 ans pour accéder à cette page.". Si \$age est supérieure ou égale à 18, le script continue et affiche "Bienvenue sur la page réservée aux adultes."

# Manipulation des fichiers

`fopen("filename", "mode");` Pour ouvrir un fichier en spécifiant le chemin du fichier et le type d'accès désiré au flux

```
$fichier = fopen("exemple.txt", "r");
```

mode	Description
'r'	Ouvre en lecture seule et place le pointeur de fichier au début du fichier.
'r+'	Ouvre en lecture et écriture et place le pointeur de fichier au début du fichier.
'w'	Ouvre en écriture seule ; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
'w+'	Ouvre en lecture et écriture ; le comportement est le même que pour 'w'.
'a'	Ouvre en écriture seule ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer. Dans ce mode, la fonction fseek() n'a aucun effet, les écritures surviennent toujours.
'a+'	Ouvre en lecture et écriture ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer. Dans ce mode, la fonction fseek() n'affecte que la position de lecture, les écritures surviennent toujours.

# Manipulation des fichiers: Fonctions

## Lire un fichier

### fread()

Lecture du fichier en mode binaire

### file()

Lit le fichier et renvoie le résultat dans un tableau

### file\_get\_contents

Lit tout un fichier dans une chaîne

### fstat

Lit les informations sur un fichier à partir d'un pointeur de fichier

### fgets()

Récupère la ligne courante à partir de l'emplacement du pointeur sur fichier

### fgetc

Lit un caractère dans un fichier

### readfile

Affiche un fichier

## Autre opérations sur un fichier

Liste : <https://www.php.net/manual/fr/ref.filesystem.php>

### fclose

Ferme un fichier

### fwrite

Écrit un fichier en mode binaire

### fputs

Alias de fwrite

### file\_put\_contents

Écrit des données dans un fichier

# Exemples des fonctions de lecture

- lire une ligne à partir du pointeur de fichier. Elle retourne la ligne lue sous forme de chaîne de caractères, ou FALSE si la fin du fichier est atteinte
- lire un nombre spécifié d'octets à partir du pointeur de fichier. Elle retourne les données lues sous forme de chaîne de caractères, ou FALSE si la fin du fichier est atteinte
- lire un fichier et le place dans un tableau où chaque élément du tableau correspond à une ligne du fichier
- lire tout le contenu d'un fichier dans une chaîne de caractères

```
$fichier = fopen("exemple.txt", "r");
while (!feof($fichier)) {
    echo fgets($fichier) . "<br>";
}
fclose($fichier);
```

```
$fichier = fopen("exemple.txt", "r");
echo fread($fichier, filesize("exemple.txt"));
fclose($fichier);
```

```
$lignes = file("exemple.txt");
foreach ($lignes as $ligne) {
    echo $ligne . "<br>";
}
```

```
$contenu = file_get_contents("exemple.txt");
echo $contenu;
```

# Exemples d'écriture dans des fichiers

- Ecrire dans un fichier ouvert avec **fopen()**

```
$fichier = fopen("nouveau.txt", "w");
fwrite($fichier, "Contenu à écrire dans le fichier\n");
fclose($fichier);
```

- Ecrire une chaîne dans un fichier. Si le fichier n'existe pas, il sera créé. Si le fichier existe déjà, son contenu sera remplacé

```
file_put_contents("nouveau.txt", "Contenu à écrire dans le fichier\n");
```

- Ecrire dans un fichier formaté en utilisant un format similaire à la fonction **printf()**.

```
$fichier = fopen("nouveau.txt", "w");
$nom = "John";
$age = 30;
fprintf($fichier, "Nom: %s, Age: %d\n", $nom, $age);
fclose($fichier);
```

# Manipulation des fichiers: Fonctions

- file\_exists(): vérifier si un fichier existe

```
if (file_exists("exemple.txt")) {  
    echo "Le fichier existe.";  
} else {  
    echo "Le fichier n'existe pas.";  
}
```

unlink(): supprimer un fichier

```
unlink("fichier_a_supprimer.txt");
```

rename(): Pour déplacer ou renommer un fichier

```
rename("ancien.txt", "nouveau.txt");
```

# l'orientée objet en PHP

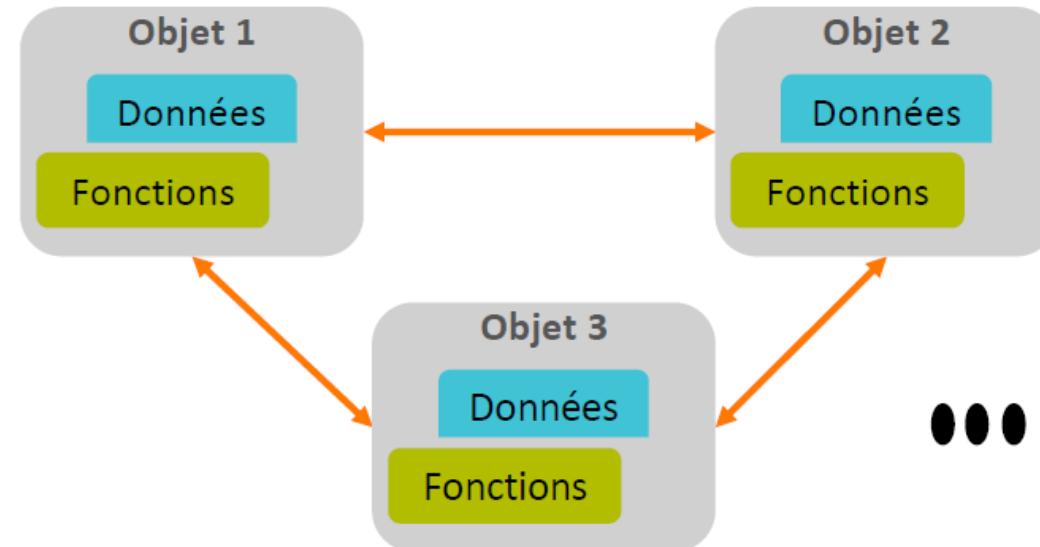
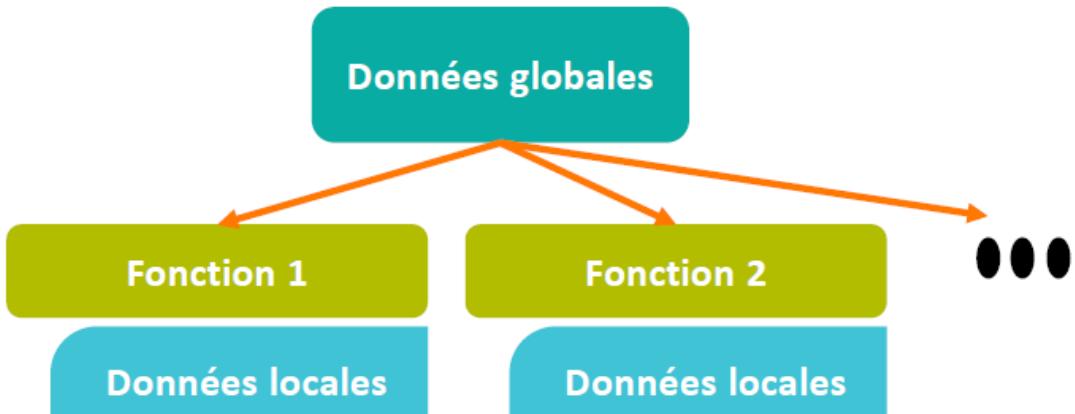
## Intérêt de programmer en Orienté Objet en PHP

### Programmation procédurale

- Le programme est divisé en procédures ou fonctions.
- Chaque fonction contient des données différentes.
- Une procédure permet d'effectuer des opérations sur les données généralement contenues dans des variables.
- Les opérations sont exécutées selon leur ordre d'écriture dans le script.

### Programmation orientée objet

- Le programme est divisé en objets.
- Les objets agissent comme une seule unité.
- Un objet est une entité qui va pouvoir contenir un ensemble de fonctions et de variables.
- Regrouper des tâches précises au sein d'objets pour obtenir une nouvelle organisation du code



# l'orientée objet en PHP

## Avantages d'une programmation orientée objet (POO)

Meilleur organisation du projet.

Facilité la maintenance du code.

Souplesse à évoluer le logiciel.

Factorisation des comportements, puisque les fonctions deviennent interdépendantes.

Masquage des données ou l'encapsulation consiste à masquer les détails d'implémentation d'un objet, en définissant une interface.

Spécificateur d'accès en utilisant un assesseur qui est une méthode d'accès pour connaître ou modifier la valeur d'un attribut d'un objet.

Transmission des propriétés grâce à l'héritage (ainsi que le polymorphisme), ainsi éviter la duplication de code.

L'agrégation qui permet de définir des objets composés d'autres objets.

## les classes

- ❑ Une classe définit un modèle, un moule, à partir duquel tous les objets de la classe seront créés
- ❑ La classe décrit les données internes ainsi que les fonctionnalités des objets
- ❑ La classe est une vision « inerte », une recette de cuisine, visant à décrire la structure et le comportement des objets qui seront créés
- ❑ La construction d'un objet à partir de la classe génératrice s'appelle instantiation
- ❑ Les objets, entités « vivantes » en mémoire, sont des instances de la classe

# Instanciation

- ❑ La classe est une description « inerte »
- ❑ Les objets doivent être instanciés à partir de la classe génératrice pour exister et devenir fonctionnels

Exemple : la classe Animal

```
$chat = new Animal();  
$chien = new Animal();  
$cheval = new Animal();
```

# Encapsulation

- ❑ Procédé consistant à rassembler les données et les traitements au sein des objets
- ❑ L'implémentation interne des objets est cachée
- ❑ les objets sont vus à travers leurs spécifications
- ❑ les données internes et les fonctionnalités possèdent un niveau de visibilité et peuvent éventuellement être masquées :

**Public**

**privé**

**Protégé**

# Encapsulation

données internes des objets sont appelées Les attributs (ou propriétés voire champs)

Les fonctionnalités des objets sont appelées méthodes

Méthodes habituelles :

- **Constructeur / destructeur**
- **Accesseurs / modificateurs (getters / setters)**

Référence a l'objet courant dans la description de la classe . \$this

# Visibilité

## Publique

- Les données internes et les méthodes sont accessibles par tous

## Protégé

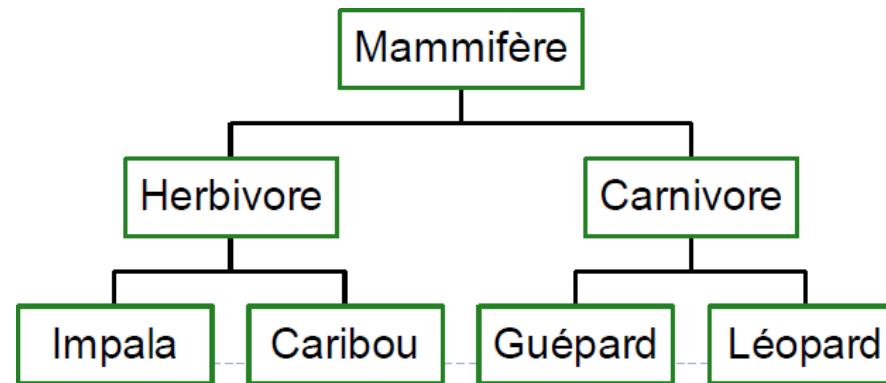
- Les données internes et les méthodes sont accessibles aux objets dérivés

## Privé:

- Les données internes et les méthodes ne sont accessibles qu'aux objets de la classe

# Héritage ou dérivation ou extension

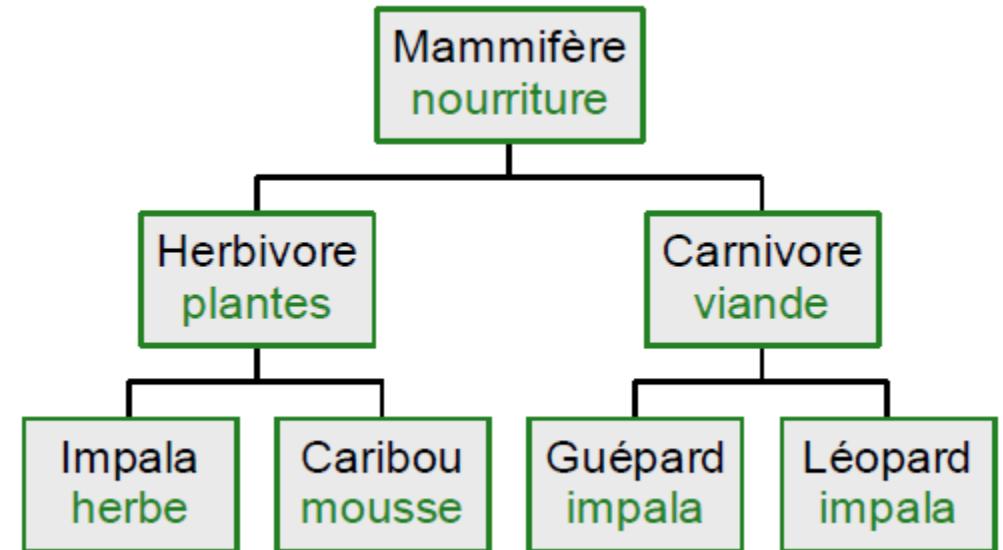
- ❑ Création de nouvelles classes à partir du modèle d'une classe existante
- ❑ La nouvelle classe possède **tous les attributs et méthodes de la classe mère**
- ❑ La nouvelle classe peut proposer de **nouveaux attributs et de nouvelles méthodes ou spécialiser des méthodes de la classe mère**



# Polymorphisme

- ❑ Choix dynamique de la méthode qui correspond au type réel de l'objet
- ❑ Méthode **mange()**

```
function nourriture(Mammifere $m) {  
    return $m->mange();  
}  
$i = new Impala(); nourriture($i);  
$c = new Carnivore(); nourriture($c);
```



herbe

viande

# Définition d'une classe

```
<?php  
class MaClasse {  
    private $madonnee ;  
  
    public function __construct($param) {  
        $this->madonnee = $param ;  
    }  
  
    public function __destruct() {  
        echo "Destruction..." ;  
    }  
  
    function affiche() {  
        echo "madonnee : "  
        . $this->madonnee  
    }  
}
```

Déclaration de classe

Attribut privé

Constructeur public

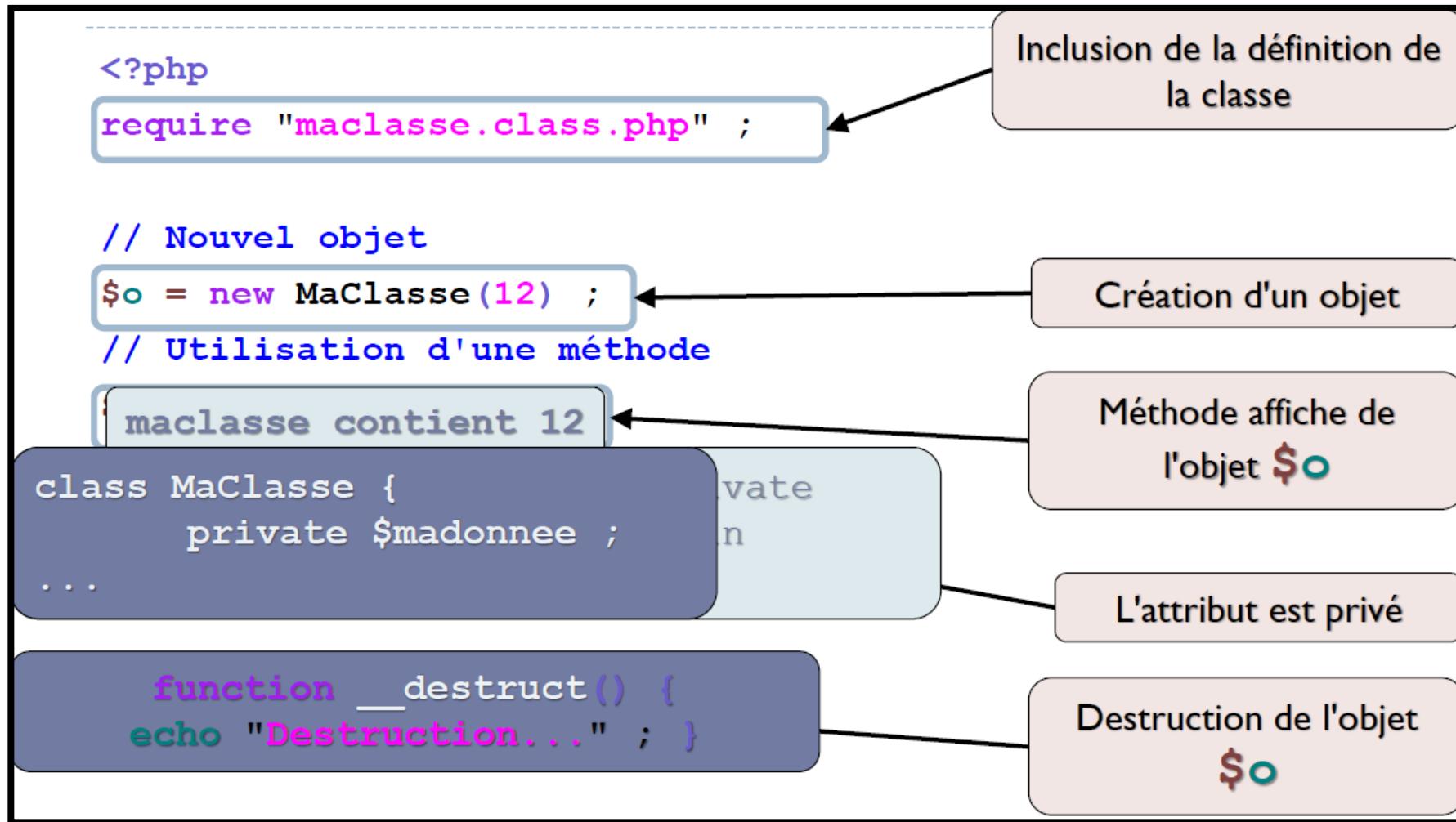
Référence à l'objet courant

Destructeur public

Méthode publique par défaut

Accès à un attribut

# Utilisation d'une classe



# Valeur par défaut des attributs

```
<?php  
class MaClasse {  
    private $madonnee = "Défaut" ;  
    public function affecte($val) {  
        $this->madonnee = $val ; }  
    public function affiche() {  
        echo "madonnee : ".$this->madonnee ; }  
}  
  
$o = new MaClasse()  
madonnee : Défaut  
{ madonnee : Nouvelle }  
$o->affiche();
```

Attribut avec valeur par défaut

Nouvel objet

Affichage

Affectation

Affichage

# Attributs et méthodes de classe

- ❑ Mot clé **Static**
- ❑ Attributs et méthodes utilisables sans instance de la classe (=attributs et méthode de classe)
- ❑ Attributs **NE** peuvent **PAS** être accédés depuis une instance (**\$objet->attribut**)
- ❑ Attributs partagés par toutes les instances de la classe
- ❑ Méthodes peuvent être accédés depuis une instance(**\$objet->methode ()** mais c'est mal !)
- ❑ Dans les méthodes, **\$this** n'est pas disponible

# Attributs statiques

```
class MaClasse {  
    private static $n = 0;  
    public function __construct() {  
        echo ++MaClasse::$n  
            . " instance(s)" ; }  
    public function __destruct() {  
        echo "destruction" ; self::$n-- ; }  
}
```

Attribut privé statique :  
ne peut être accédé que par  
des méthodes de la classe

Accès à l'attribut  
statique

1 instance(s)  
2 instance(s)  
destruction  
2 instance(s)  
3 instance(s)

Fatal error: Cannot access private property  
MaClasse::\$n in **dummy.php** on line 37

# Méthodes statiques

```
class MaClasse {  
    private static $n = 0 ;  
    public function __construct() {  
        echo ++MaClasse::$n." instance(s)\n" ; }  
    public function __destruct() {  
        MaClasse::$n-- ; }  
    public static function f($i) {  
        echo "Dans f() : ".($i*$i) ; }  
}
```

Méthode publique statique

1 instance(s)

Dans f() : 4

Dans f() : 9

Appel à partir d'une instance  
Toléré

Appel sans instance

# Constantes de classe

```
class MaClasse {  
    const constante = "Valeur";  
  
    public function montre() {  
        echo self::constante;  
    }  
}
```

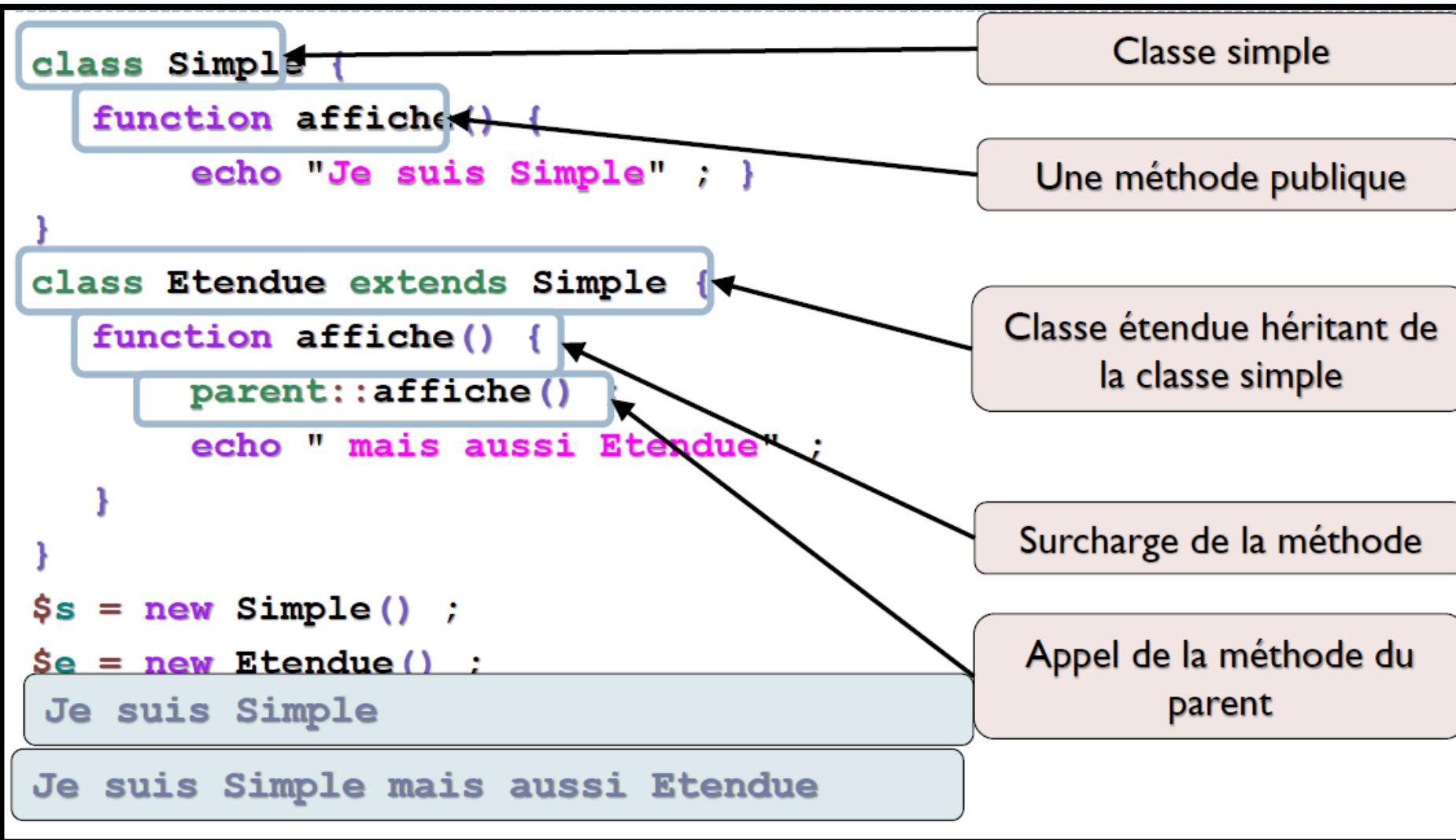
Constante publique de classe

```
$c = new MaClasse();  
$c->montre();  
echo MaClasse::constante;
```

Accès à la constante de classe depuis la classe

Accès à la constante de classe à l'extérieur de la classe

# Héritage

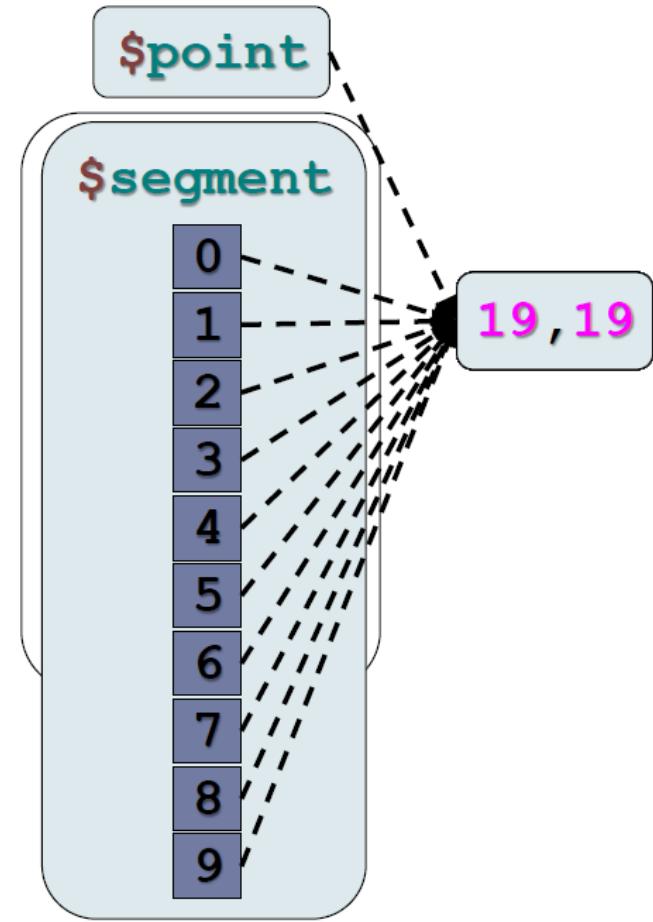


# Attribution d'objets

```
class Point {  
    private $_x ;  
    private $_y ;  
  
    public function __construct($x=0, $y=0) {  
        $this->_x = $x ;  
        $this->_y = $y ; }  
  
    public function set($x, $y) {  
        $this->_x = $x ;  
        $this->_y = $y ; }  
  
    public function toString() {  
        return "({$this->_x}, {$this->_y})" ; }  
}
```

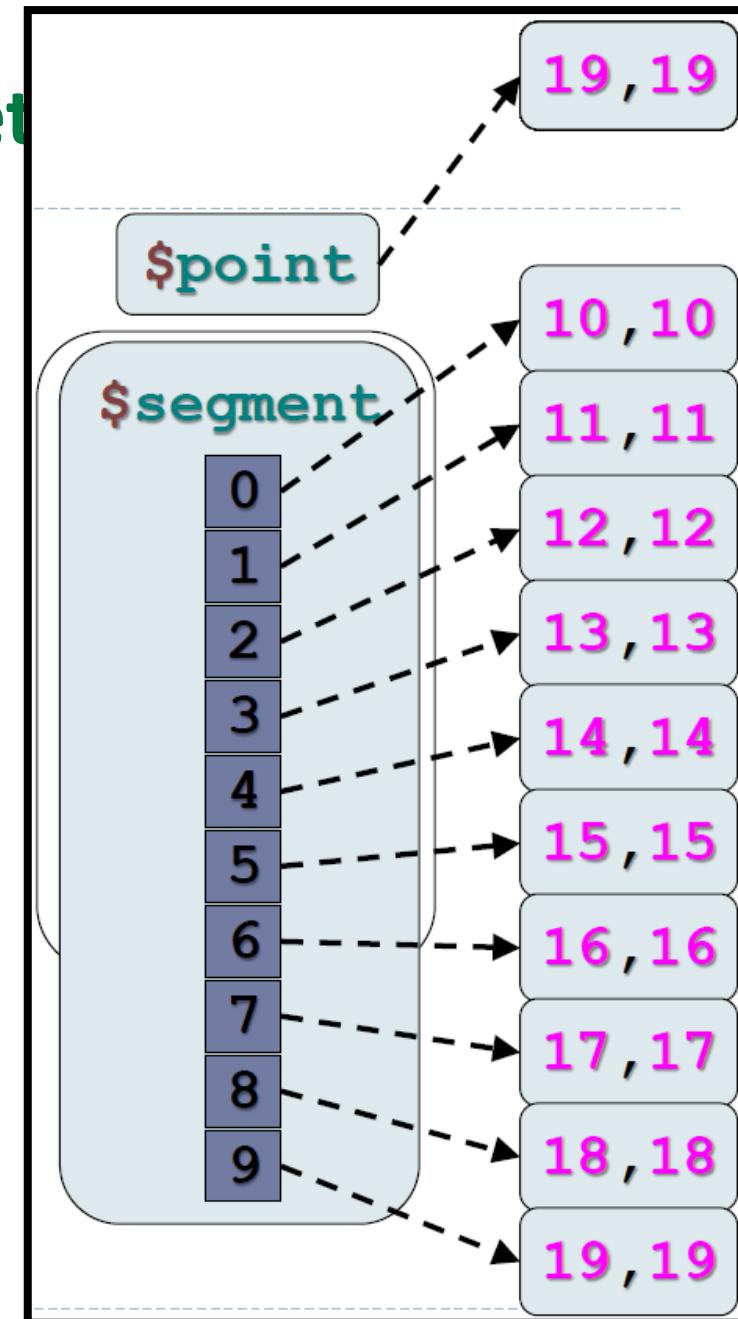
# Attribution d'objets

```
$segment = array() ;  
$point = new Point() ;  
for ($i=10; $i<20; $i++)  
{  
    $point->set($i, $i) ;  
    $segment[] = $point ;  
}  
  
foreach ($segment as $k => $p)  
    echo "$k: {$p->toString()} \n" ;
```



# Clonage d'objet

```
$segment = array() ;  
$point = new Point() ;  
for ($i=10; $i<20; $i++)  
{  
    $point->set($i, $i) ;  
    $segment[] = clone $point ;  
}  
  
foreach ($segment as $k => $p)  
    echo "$k: {$p->toString() }\n" ;
```



# Objets comme arguments de fonctions

```
function origine($p) {  
    $p->set(0, 0) ; }  
  
$point = new Point(10, 10) ;  
  
echo "avant: {$point->toString()}\n" ;  
origine($point) ;  
echo "apres: {$point->toString()}\n" ;
```

avant: (10, 10)  
apres: (0, 0)

Passage de l'objet  
Point par référence

## Gestion des erreurs : exceptions

Gestion des exception identiques à C++/Java

Exception peut être:

jetée : Throw

Essayée : try

Capturée : catch

Exception jetée : code après throw non exécuté

Capture : 1 ou plusieurs blocs (selon type)

Exception non capturée : erreur fatale

# Utilisation des exceptions

```
try {  
    $error = 'Toujours lancer cette erreur';  
    throw new Exception($error);  
    /* Le code après une exception n'est  
       jamais exécuté. */  
    echo 'Jamais exécuté'; }  
  
catch (Exception $e) {  
    echo "Capture Exception: "  
        . $e->getMessage() . "\n"; }  
  
// Poursuite de l'exécution  
echo 'Bonjour le monde';
```

Capturer