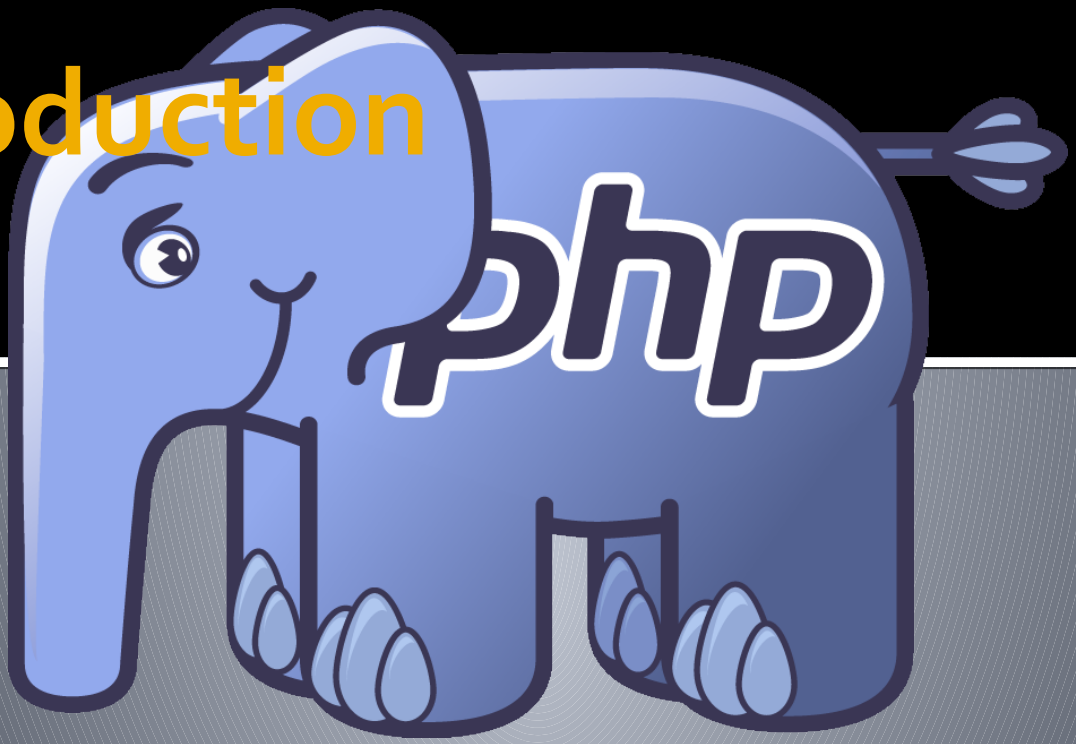
Ismail HADJADJ
logismail@gmail.com

Programmation Web Coté Serveur : PHP

Introduction



PHP: Langage de script pour le Web

- Qu'est-ce que PHP ?
  - Langage de script. Principalement utilisé côté serveur
  - Créé en 1994-1995 par Rasmus Lerdorf
  - Acronyme initial : **P**ersonal **H**ome **P**age
  - Acronyme récursif : **P**HP: **H**ypertext **P**reprocessor
  - Extension utilisée sur ≈80% des serveurs Web
  - Langage multi plate-forme (UNIX / Windows…)
  - Open Source
  - Versions actuelles (*source w3techs.com, 03/14*) :
    - PHP5 >97%
    - PHP4 <3% - PHP3 <0,1%

Langage de script ?

- Langage impératif
- Langage interprété
- Pas de compilation
- Le code source « est » le programme
- Typage faible
- Programmation orientée objet possible

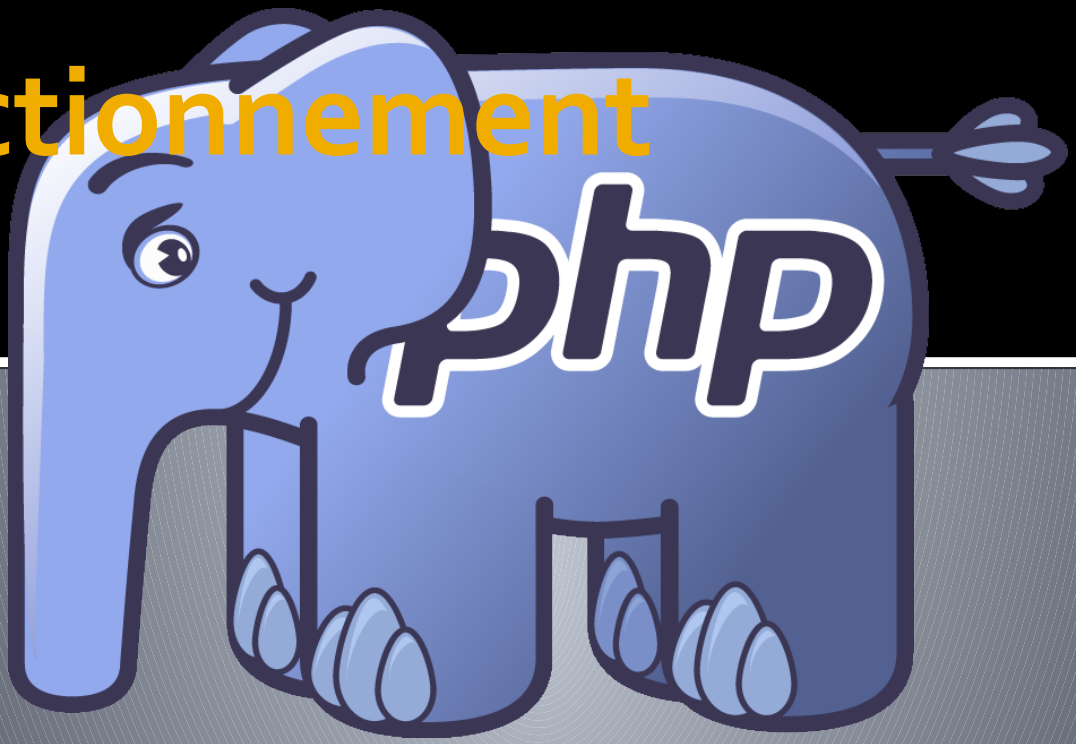
Utilité et utilisation de PHP

- Création de pages Web « dynamiques », fabriquées à la volée, construite à la demande
- Interface entre un serveur Web et des bases de données
- Création d'applications Web

Principales fonctionnalités de PHP

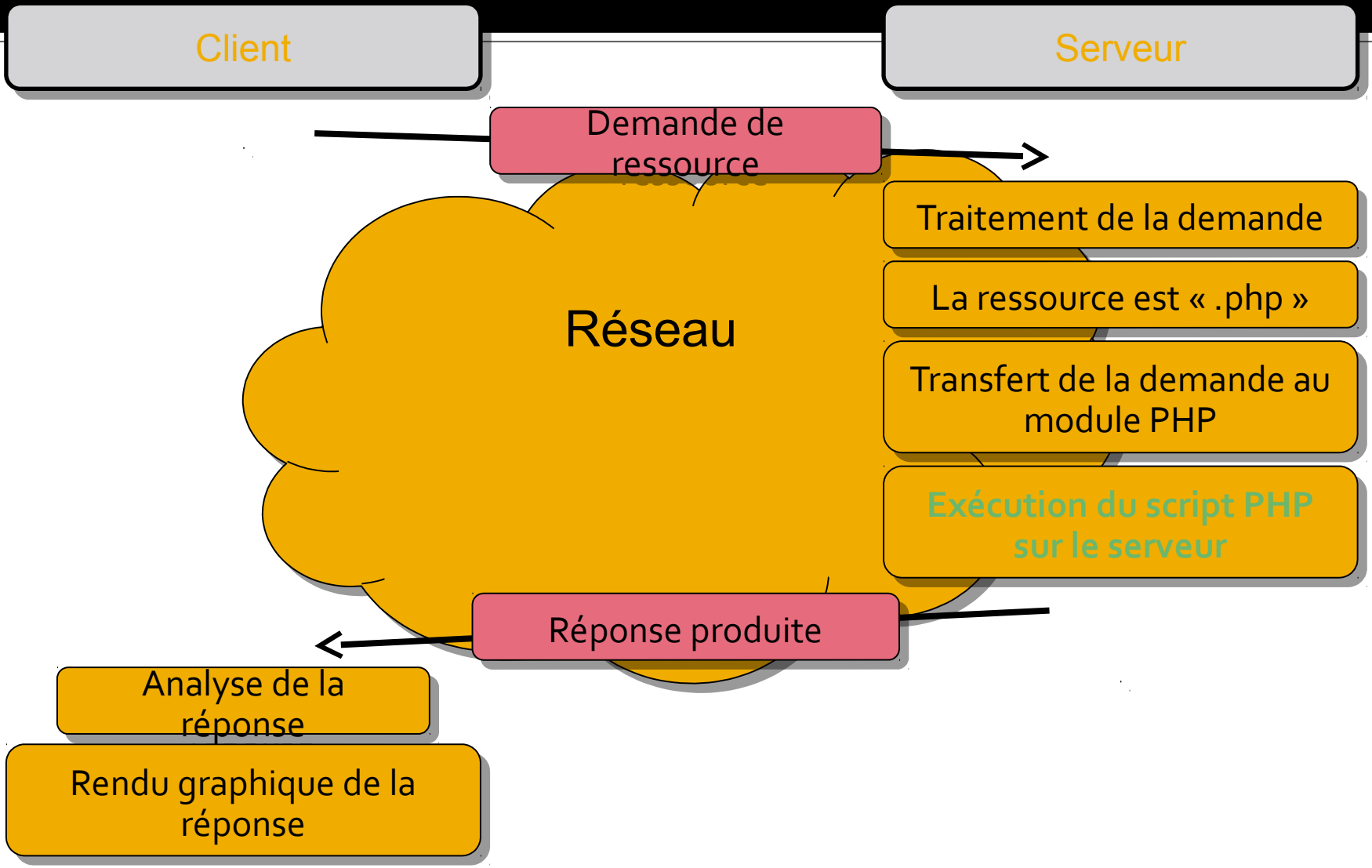
- Manipulation de chaînes et tableaux
- Calendrier / dates / heures
- Fonctions mathématiques
- Accès au système de fichiers
- Manipulation d'images
- HTTP / FTP / IMAP
- Bases de données (Oracle, MySQL, …)
- XML
- …

Fonctionnement



Fonctionnement de PHP

Fonctionnement de PHP



Programme en PHP

Délimitation du code PHP dans le fichier **.php** :

- `<?php Code PHP ?>`
- `<script language="PHP">`
  `Code PHP`
  `</script>`
- `<? Code PHP ?>`
- `<% Code PHP %>`

Fermeture optionnelle et déconseillée

Confusion avec JavaScript
➔ à bannir !!

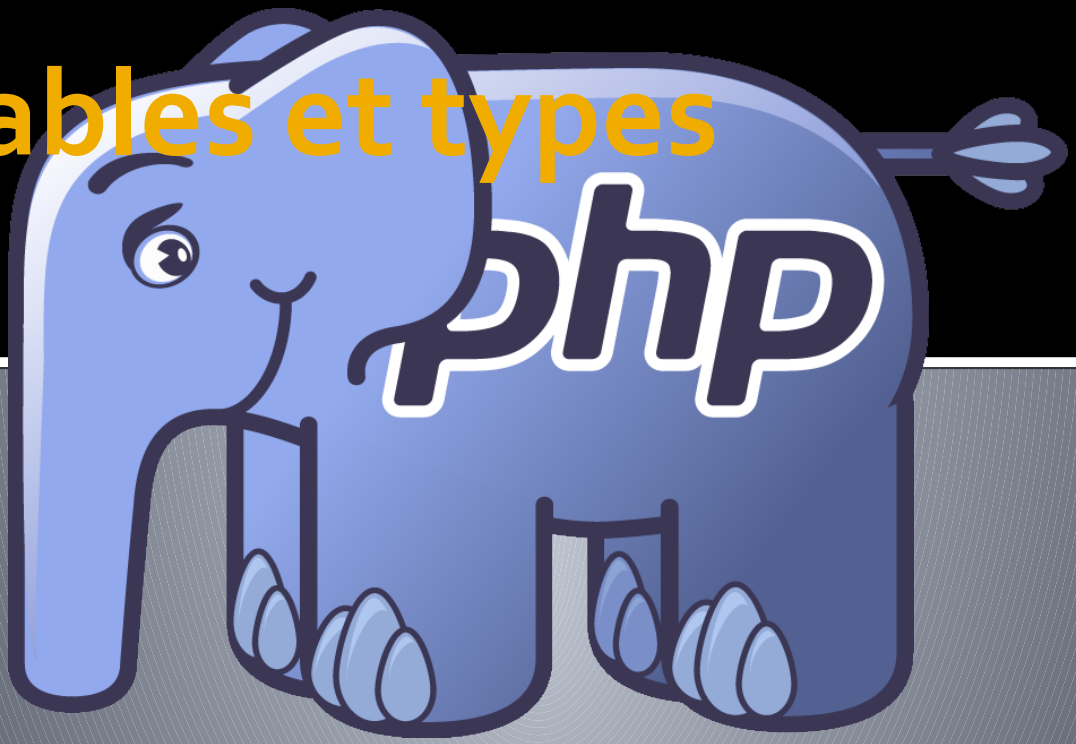
Dépend de la configuration du serveur
➔ à bannir !!

Eléments de syntaxe 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
```

Variables et types

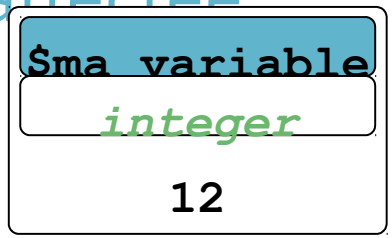


Les variables et les types de données

- Tout identificateur commence par « `$` »
- Les affectations sont réalisées grâce à « `=` »
- Les types sont :
  - Numérique entier : `12` ou réel : `1.54`
  - Chaîne: `"Hello"` ou `'Bonjour'`
  - Booléen: `true, false` (PHP 4)
  - Tableau: `$tab[2]=12`
  - Objet (PHP4, PHP5)
  - Ressource
  - `NULL`
- Les variables ne sont pas explicitement déclarées

Les variables et les types de données

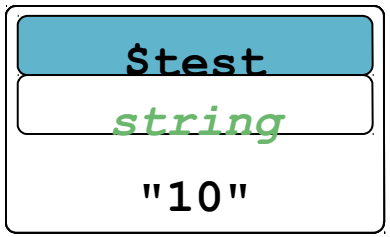
- La « déclaration » d'une variable correspond à sa première affectation
- Le type d'une variable est dynamique et est déterminé par la valeur qui lui est affectée
- Une variable possède donc :
  - Un nom (commençant par $)
  - Un type dynamique
  - Une valeur



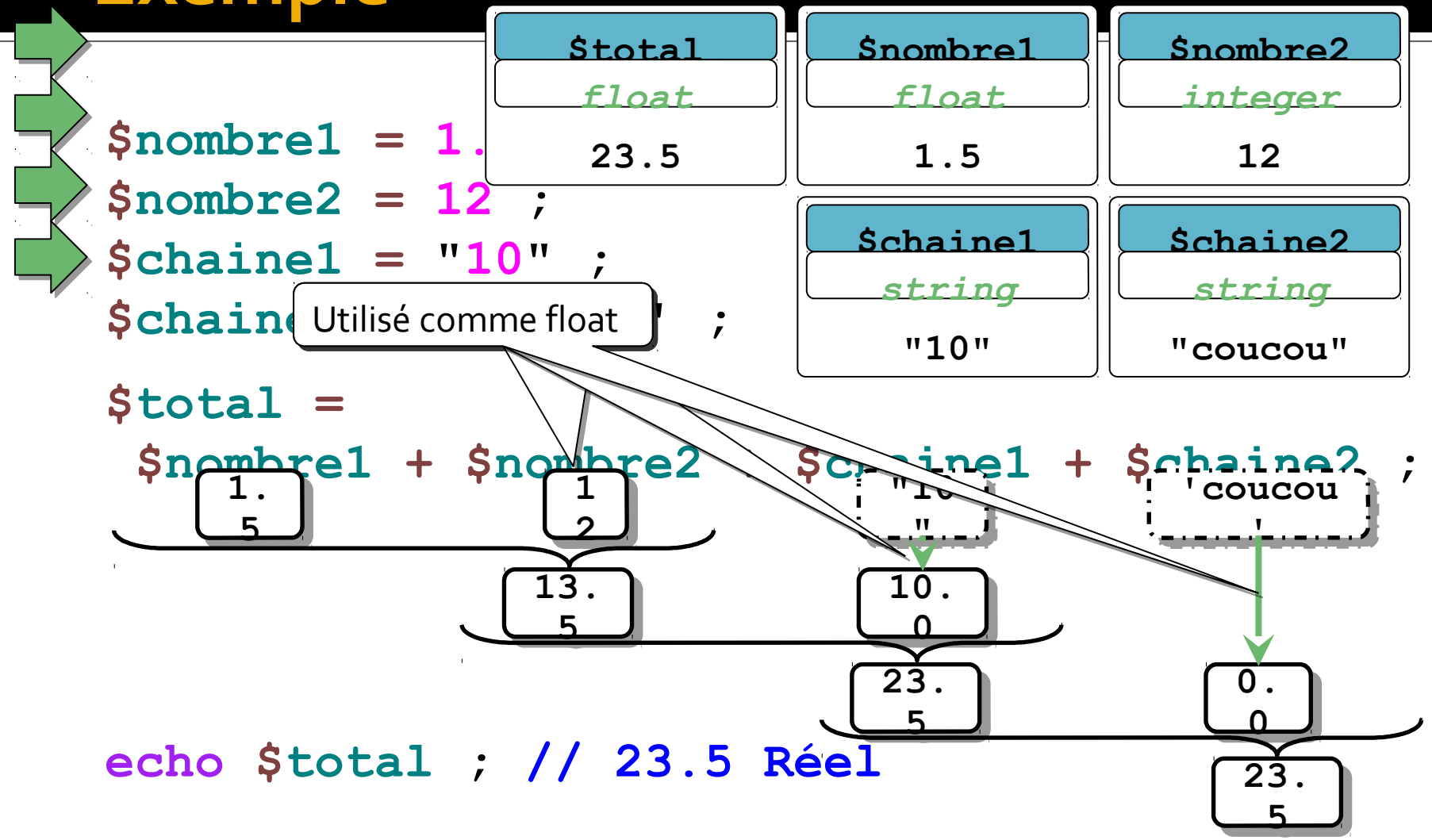
- Adaptation possible du type selon le contexte sans modification du type intrinsèque

Typage à l'affectation. Exemple

```
// Pas de déclaration préalable de variable

$test = 1.5 ;

$test = 12 ;

$test = array() ;

$test = "10" ;
```



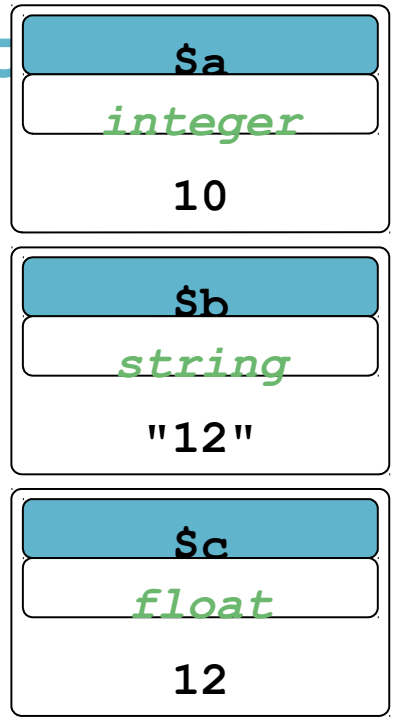
Typage en fonction du contexte. Exemple



Modification de types

- Le type désiré peut être précisé : cast
- **$variable = (nom_du_t**
  valeur

```
$a = (integer) "10" ;

$b = (string) 12 ;

$c = (float) $b ;
```



Définition de constantes

```
<?php
define("ma_constante", "Bonjour à tous") ;

echo ma_constante ;
?>
```

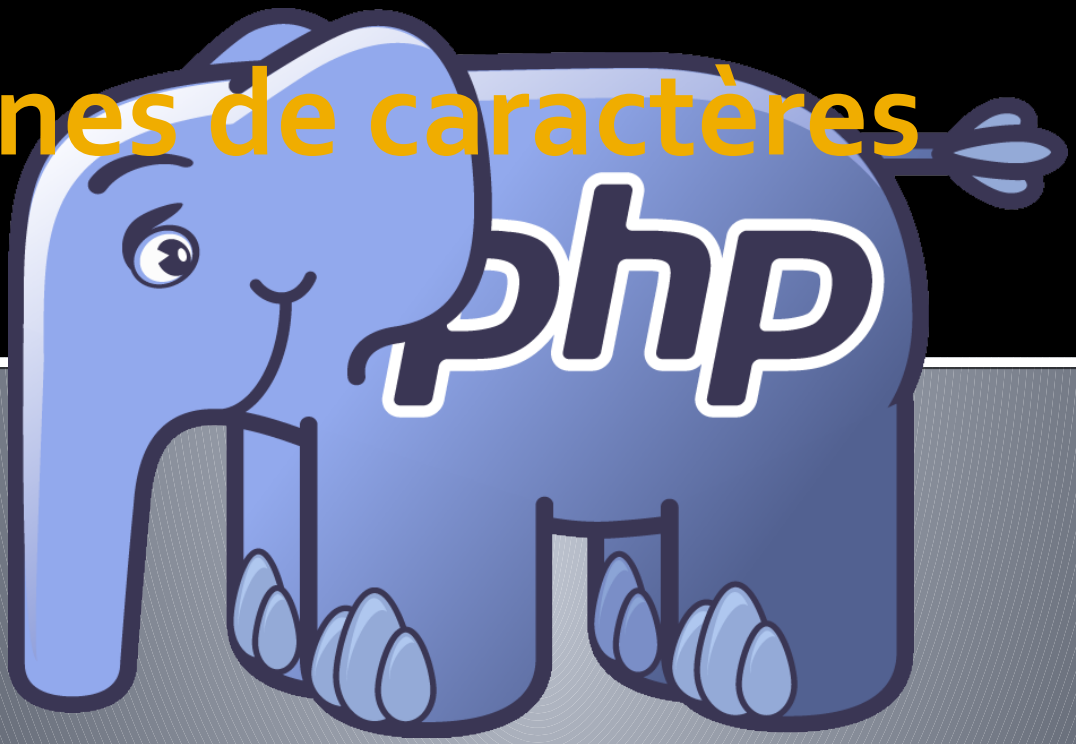
nom

valeur

Définition d'une constante

Utilisation de la constante

Chaînes de caractères



Substitution de variables dans les chaînes

- Guillemets doubles

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

```
chaîne
voici une chaîne
```

- Guillemets simples

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

```
chaîne
voici une $a
```

Substitution de variables dans les chaînes

- Syntaxe *HereDoc*

```
$a="chaîne" ;
$b=<<<MARQUEDEFIN
voici une $a
sur deux lignes ;-)
MARQUEDEFIN;
```

```
chaîne
```

```
voici une chaîne
sur deux lignes ;-)
```

- Syntaxe *NowDoc* (PHP 5.3)

```
$a="chaîne" ;
$b=<<<'MARQUEDEFIN'
voici une $a
sur deux lignes ;-)
MARQUEDEFIN;
```

```
chaîne
```

```
voici une $a
sur deux lignes ;-)
```

- MARQUEDEFIN au choix, seule sur la ligne

La commande echo

- Naïvement, permet d'envoyer du texte au navigateur du client (« écrire » la page au format HTML résultant de l'interprétation de PHP)
  - `echo "Bonjour" ;`
  - `$nom="Marcel" ; echo "Bonjour $nom" ;`
- Plus exactement, permet d'envoyer des octets au navigateur du client
  - Contenu HTML, XML, CSS, JavaScript, …
  - Données d'une image
  - Contenu d'un ficher PDF, Flash, etc.

Hello world !

Interprétation du code PHP sur le serveur et transmission du résultat au client

Serveur

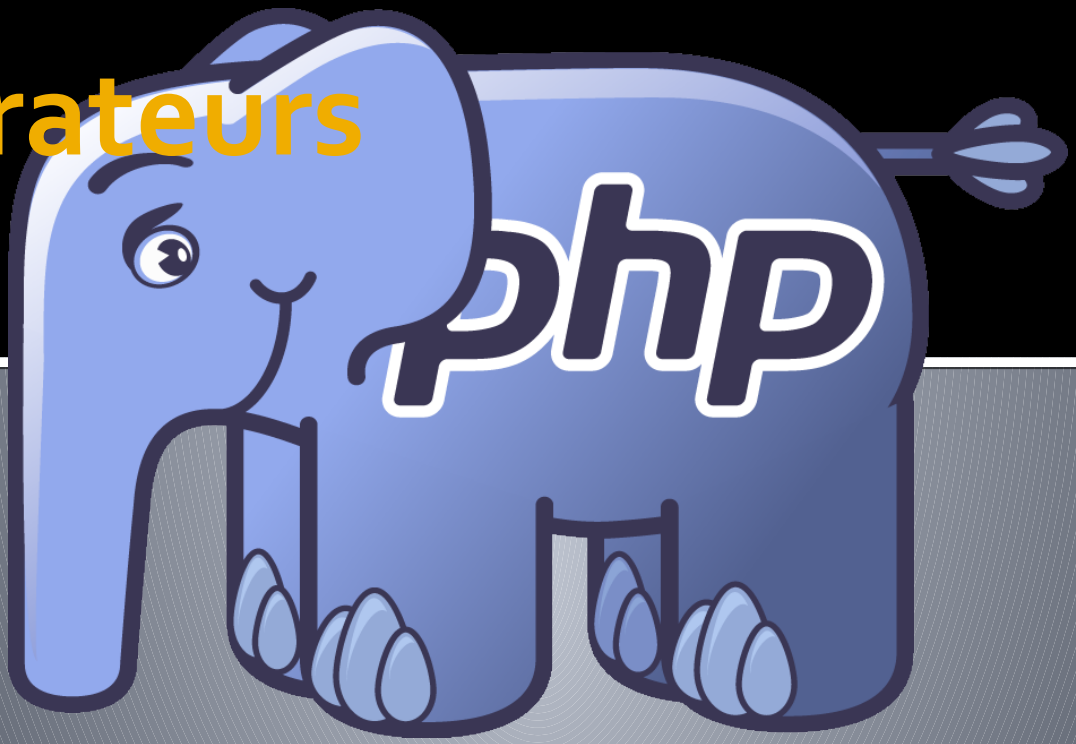
```
<?php
$debut = <<<HTML
<html>
  <head>
    <title>hello</title>
  </head>
  <body>\n
HTML;
$corps = "Hello world!\n";
$fin   = <<<HTML
 </body>
</html>
HTML;
/* Envoi au client */
echo  $debut.$corps.$fin ;
```

Navigateur

```
<html>
  <head>
    <title>hello</title>
  </head>
  <body>
Hello world!
 </body>
</html>
```

Impossible de voir le code PHP depuis le navigateur !!

Opérateurs



Concaténation de chaînes

- Permet d'assembler plusieurs chaînes
- Réalisé grâce à l'opérateur point : « . »

```
"Bonjour " . "Marcel"
```

→ vaut "Bonjour Marcel"

```
$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 d'in- et de dé-crémentation pré- et post-fixés

`$a++`	Retourne la valeur de `$a` puis augmente la valeur de `$a` de 1
`++$a`	Augmente la valeur de `$a` de 1 puis retourne la nouvelle valeur de `$a`
`$a--`	Retourne la valeur de `$a` puis diminue la valeur de `$a` de 1
`--$a`	Diminue la valeur de `$a` de 1 puis retourne la nouvelle valeur de `$a`

Les opérateurs de comparaison

`$a == $b`	Vrai si égalité entre les valeurs de `$a` et `$b`
`$a != $b`	Vrai si différence entre les valeurs de `$a` et `$b`
`$a < $b`	Vrai si `$a` inférieur à `$b`
`$a > $b`	Vrai si `$a` supérieur à `$b`
`$a <= $b`	Vrai si `$a` inférieur ou égal à `$b`
`$a >= $b`	Vrai si `$a` supérieur ou égal à `$b`
`$a === $b`	Vrai si `$a` et `$b` identiques (valeur et type)
`$a !== $b`	Vrai si `$a` et `$b` différents (valeur ou type)

Comparaison large / stricte

```
$a = 12 ; $b = "12" ; $c = 12.0 ;

var_dump($a ==  $b) ;
var_dump($a ==  $c) ;
var_dump($c ==  $b) ;
var_dump($a !=  $b) ;
var_dump($a !=  $c) ;
var_dump($c !=  $b) ;
var_dump($a === $b) ;
var_dump($a === $c) ;
var_dump($c === $b) ;
var_dump($a !== $b) ;
var_dump($a !== $c) ;
var_dump($c !== $b) ;
```

```
boolean true
boolean false
boolean false
boolean false
boolean true
boolean true
boolean false
boolean false
boolean false
boolean true
boolean true
boolean true
```

$a
integer
12

$b
string
"12"

$c
float
12

Les opérateurs logiques

`[Expr1]` **`and`** `[Expr2]`	Vrai si `[Expr1]` et `[Expr2]` sont vraies
`[Expr1]` **`&&`** `[Expr2]`	idem
`[Expr1]` **`or`** `[Expr2]`	Vrai si `[Expr1]` ou `[Expr2]` sont vraies
`[Expr1]` **`||`** `[Expr2]`	idem
`[Expr1]` **`xor`** `[Expr2]`	Vrai si `[Expr1]` ou `[Expr2]` sont vraies mais pas les deux
`!` `[Expr1]`	Vrai si `[Expr1]` est non vraie

Les opérateurs sur bits

`$a & $b`	ET binaire
`$a | $b`	OU binaire
`$a ^ $b`	XOR binaire
`~ $a`	Inversion bit à bit
`$a << $b`	`$a` décalé à gauche de `$b` rangs
`$a >> $b`	`$a` décalé à droite de `$b` rangs

Précédence des opérateurs

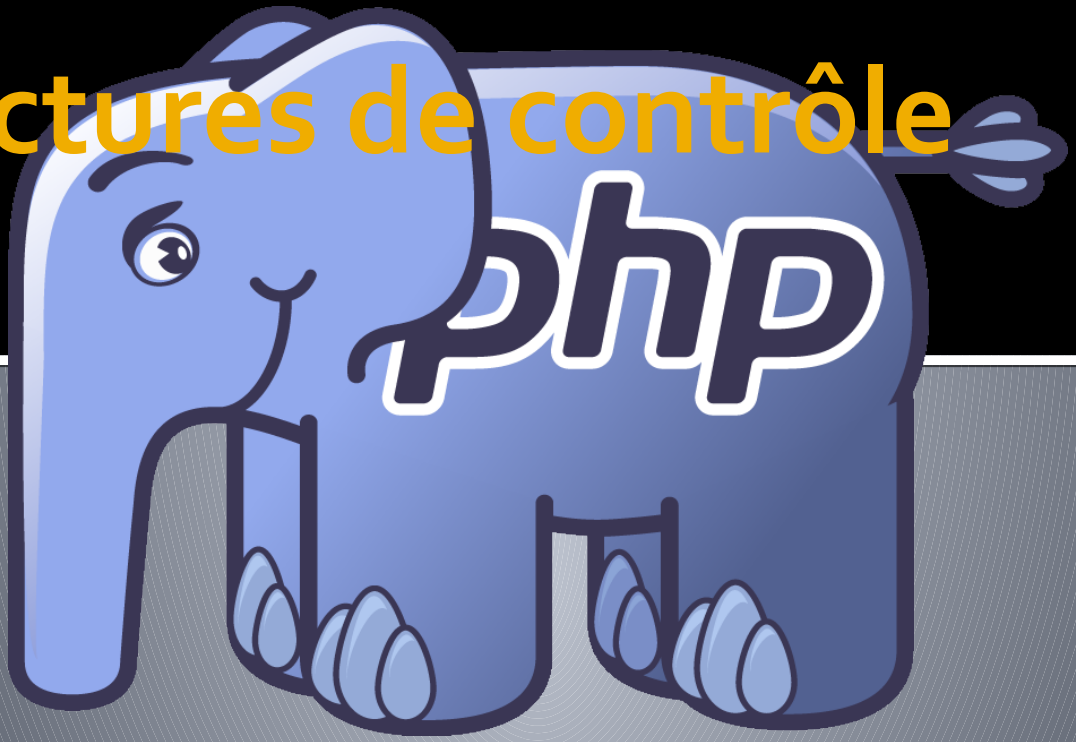
Opérateurs
new
[
++ --
! ~ - (int) (float) (string) (array) (object) @
* / %
+ - .
<< >>
< <= > >=
== != === !==
&
...

Précédence des opérateurs

Opérateurs
...
^
|
&&
||
? :
= += -= *=
and
xor
or

En cas de doute,
utilisez les parenthèses ;-)

Structures de contrôle



Structure de contrôle Si…Alors… Sinon…

```
if (condition)
{
  /* Bloc d'instructions exécuté si la
  condition est vraie */
}
else
{
  /* Bloc d'instructions exécuté si la
  condition est fausse */
}
```

Optionnel

Structure de contrôle Tant que… faire…

```
while (condition)
{
  /* Bloc d'instructions répété tant que la
  condition est vraie */
}
```

```
do {
  /* Bloc d'instructions exécuté une fois
  puis répété tant que la condition est
  vraie */
} while (condition) ;
```

Structure de contrôle Tant que… faire…

```
for (avant; condition; fin_chaque_itération)
{ /* Bloc d'instructions répété tant que la
  condition est vraie */
}
```

Équivalent à :

```
avant ;
while (condition)
{ /* Bloc d'instructions répété tant que la
  condition est vraie */
  fin_chaque_itération ;
}
```

Structure de contrôle selon…

```
switch (val)
{
  case v1:
    instructions exécutées si val==v1
  case v2:
  case v3:
    instructions exécutées si val==v2
      ou si val==v3
      ou si val==v1
  …
  default:
    instructions dans tous les cas
}
```

- Fonctionne exactement comme une série de **`if/else`** avec comparaison large
- **`val`**, **`v1`**, **`v2`**, … peut être de type **`integer`**, **`float`**, **`string`**, **`NULL`**

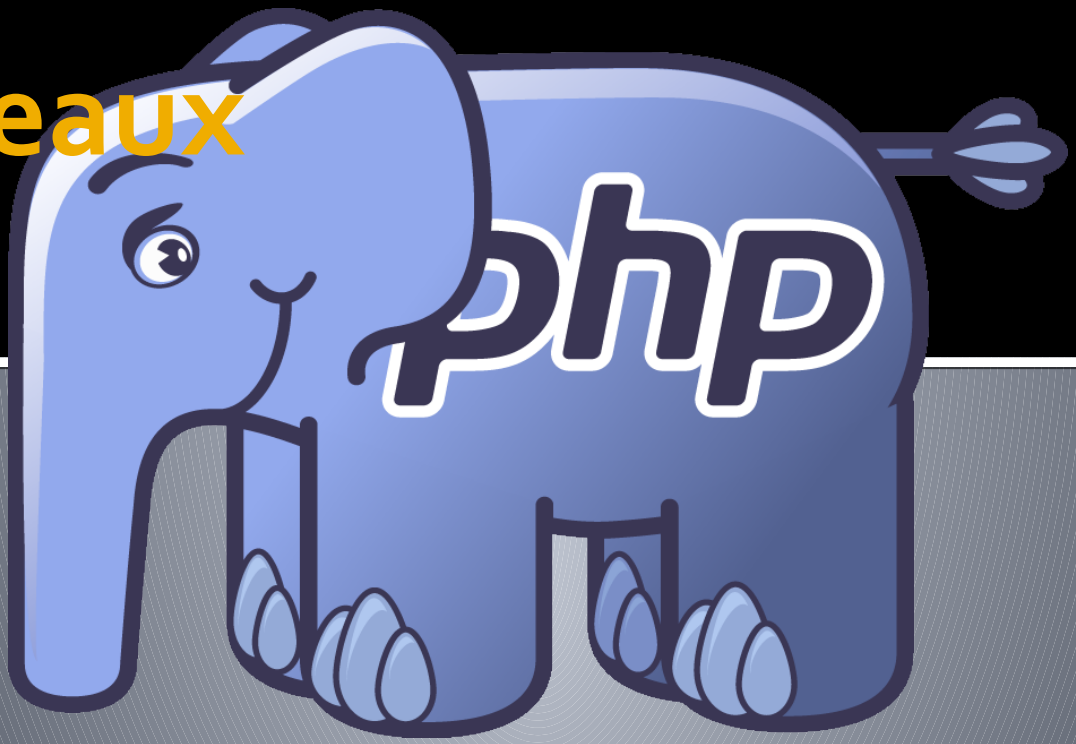
L'instruction break

Permet de sortir d'une structure de contrôle

```
switch (val)
{
  case v1:
    instructions exécutées si val==v1
    break ; /* Sortie du switch si val==v1 */
  case v2:
  case v3:
    instructions exécutées si val==v2
  ou si val==v3
  ou si val==v1
    break ; /* Sortie du switch si val==v2 ou val==v3*/
  …
  default:
    instructions exécutées dans tous les cas
                        si val!=v1 et val!=v2 et val!=v3
}
```

Cas rendus exclusifs si **break** présent pour chaque **case**

Tableaux



Principes généraux

- Un tableau PHP est en réalité une cartdonnée
- Une carte associe des valeurs à des clés
- Un tableau PHP peut être vu comme :
  - Une série de valeurs (peu importe leur type)
  - Chaque valeur est étiquetée, repérée par une clé (entier ou chaîne)

Clé	0	1	2	3
Valeur	12	15.2	"42"	NULL

  - Ils sont dits associatifs

Clé	"début"	12	"a"	"valeur"
Valeur	false	3	16	"bonjour"

Tableaux « classiques » : indexés

- Création / initialisation:

```
$tab1=array(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" ;
```

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

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

Structure de contrôle « Pour chaque… »

Des tableaux associatifs ne peuvent 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 */
}
```

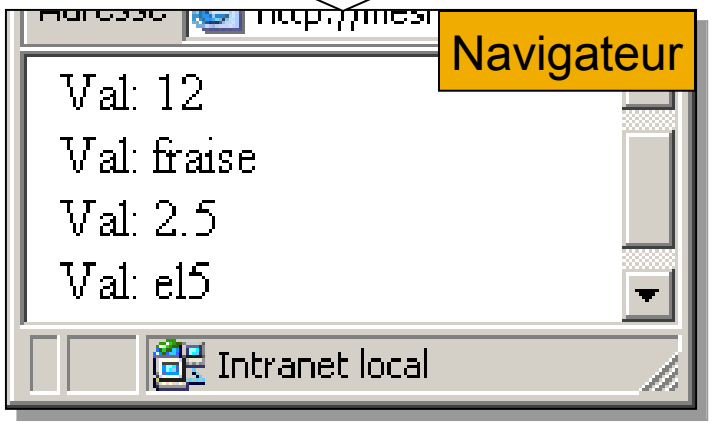
Parcours de tableau : foreach

PHP

```
…
$tab4[0] = 12 ;
$tab4[1] = "fraise" ;
$tab4[2] = 2.5 ;
$tab4[5] = "el5" ;
foreach($tab4 as $v)
{
   echo "Val:
   $v<br>\n";
}
…
```

HTML

```
…
Val:12<br>\n
Val:fraise<br>\n
Val:2.5<br>\n
Val:el5<br>\n
…
```



Tableaux associatifs

- Tableaux dont l'accès aux éléments n'est plus réalisé grâce à un index (0,1, …) mais grâce à une clé de type entier ou chaîne.
- Exemples de clés:

```
$tab['un']       = 12 ;
$tab[205]        = "bonjour" ;
$tab["la valeur"] = 3.0 ;
```

- Création

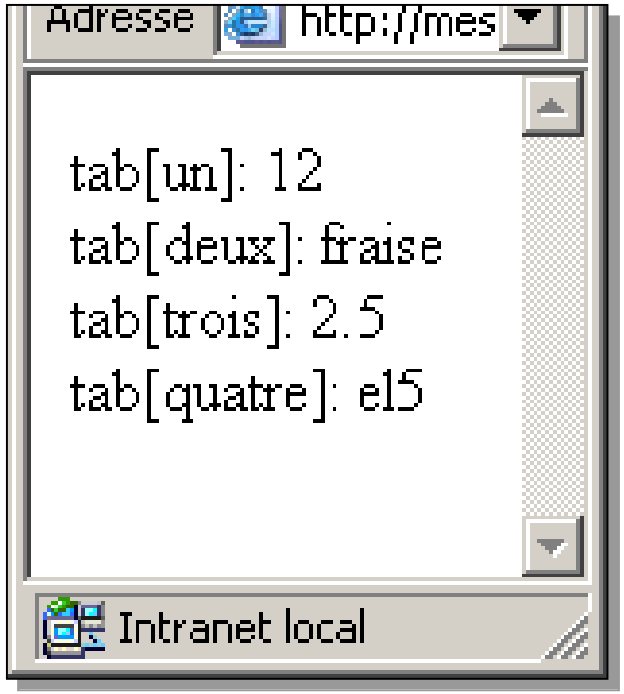
```
$tab = array(cle1 => val1,
             cle2 => val2,
             …);
```

Structure de contrôle « Pour chaque… »

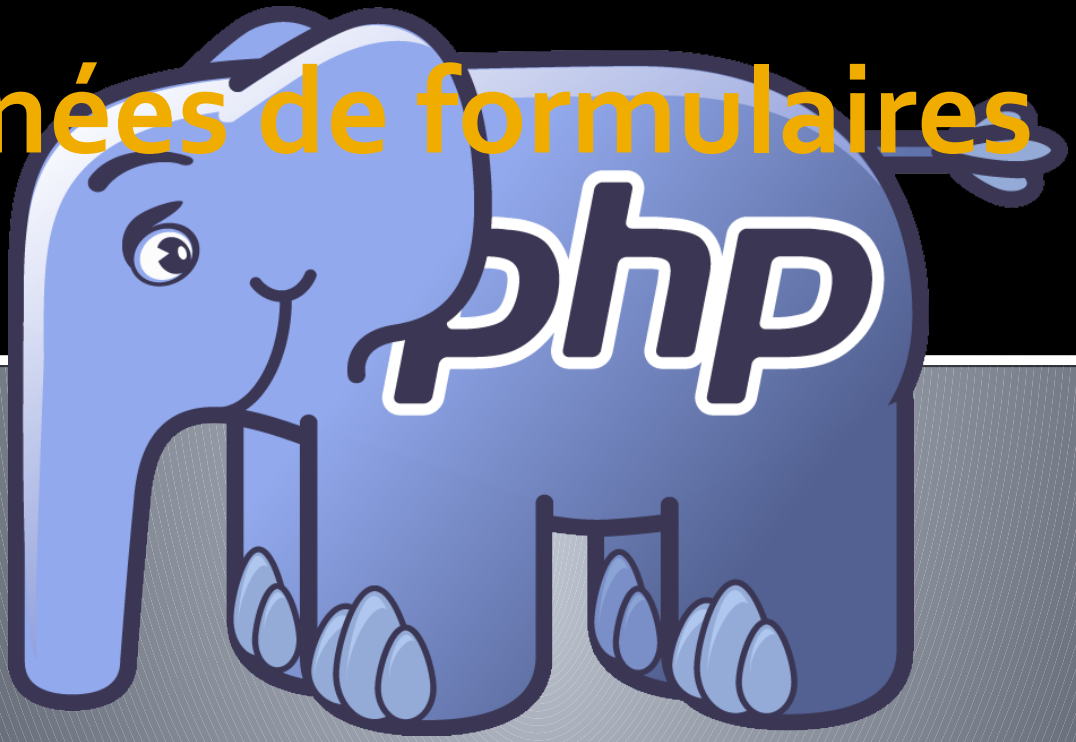
```
foreach($tableau as $cle => $element)
{
  /* Bloc d'instructions répété pour
     chaque élément de $tableau */
  /* Chaque élément de $tableau est
     accessible grâce à $element */
  /* La clé d'accès à chaque élément est
     donnée par $cle */
}
```

Parcours de tableau

```php
<?php
$html = <<<HTML
<html>
  <head><title>foreach clé</title>
  </head>
<body>
HTML;
$tab6 = array('un'      => 12,
              'deux'    => "fraise",
              "trois"   => 2.5,
              "quatre" => "el5") ;
foreach ($tab6 as $cle => $val)
{
  $html .= "tab[$cle]:
   $val<br>\n" ;
}
echo $html . "</body>\n</html>" ;
```



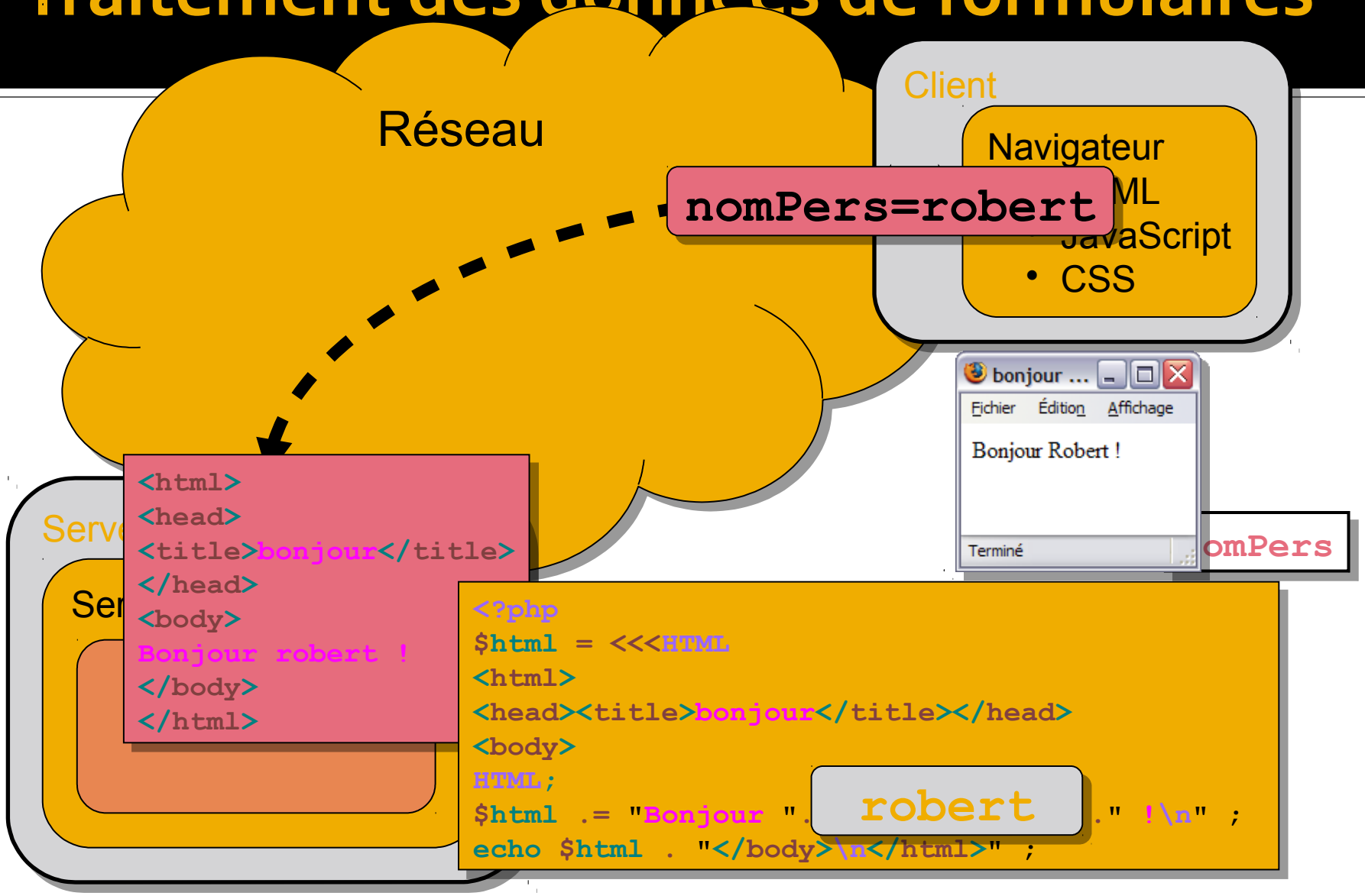
Données de formulaires



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`** (ou **`$_REQUEST`**).
- La valeur peut être trouvée grâce à une clé qui porte le même nom que le champs du formulaire de la page HTML de saisie.

Traitement des données de formulaires



Exemple – Formulaire HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>formulaire</title>
</head>
<body>
 <form action="valide1.php" method="get">
    Nom: <input type="text" name="nomPers">
 <input type="submit" value="Envoyer">
 </form>
</body>
</html>
```

Exemple – Traitement en PHP

```php
<?php
$html = <<<HTML
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>bonjour</title>
</head>
<body> HTML;
  if (isset($_GET['nomPers']))
  {
      if (!empty($_GET['nomPers']))
      {
      $html .= "Vous avez saisi '"
                    .$_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 ?

Formulaires contenant des champs « SELECT »

Formulaires contenant des champs « SELECT unique»

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<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>
```

Envoyer

valide3.php?
sel=Pomme

Formulaires contenant des champs « SELECT multiple»

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<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>
```

Envoyer

valide3.php?
sel=Pomme&sel=Poire

??

Formulaires contenant des champs « SELECT multiple»

```
<html>
<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>
```

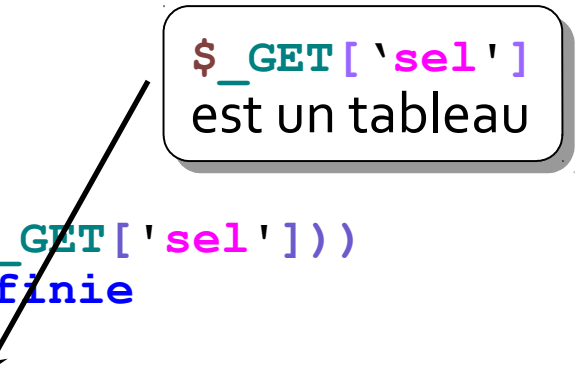
Envoyer

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

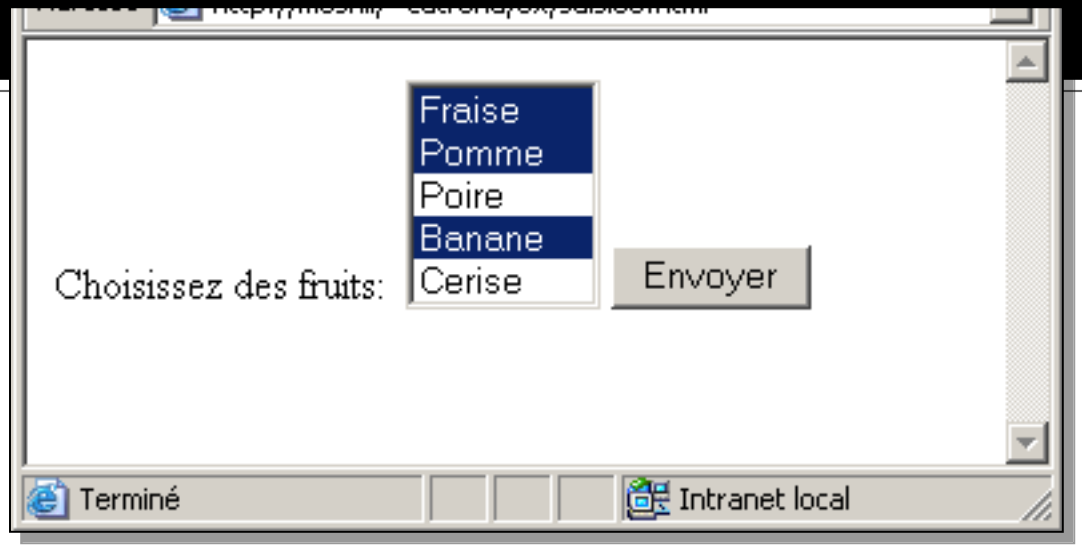
valide3.php?sel[]=Pomme&sel[]=Poire

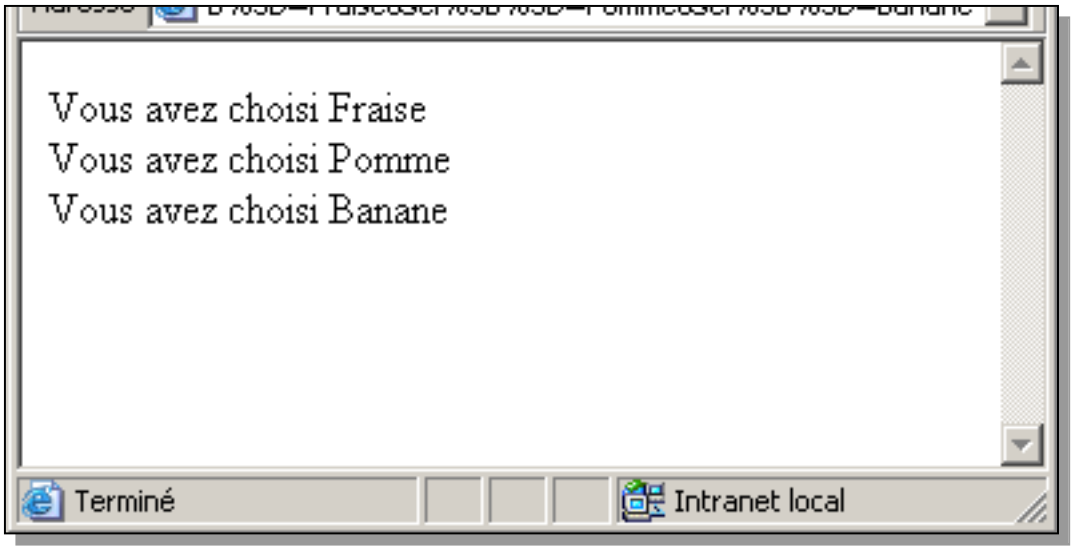
Traitement des données des champs « SELECT »

```php
<?php
$html = <<<HTML
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
   <title>Liste de fruits</title>
</head>
<body>
HTML;
   if (isset($_GET['sel']) && !empty($_GET['sel']))
   {/* La variable $_GET['sel'] est définie
       et elle n'est pas vide */
      foreach($_GET['sel'] as $fruit)
          $html .= "Vous avez choisi $fruit<br>\n" ;
   }
   else
       $html .= "Vous n'avez pas choisi de fruit\n" ;
echo $html . "</body>\n</html>" ;
```

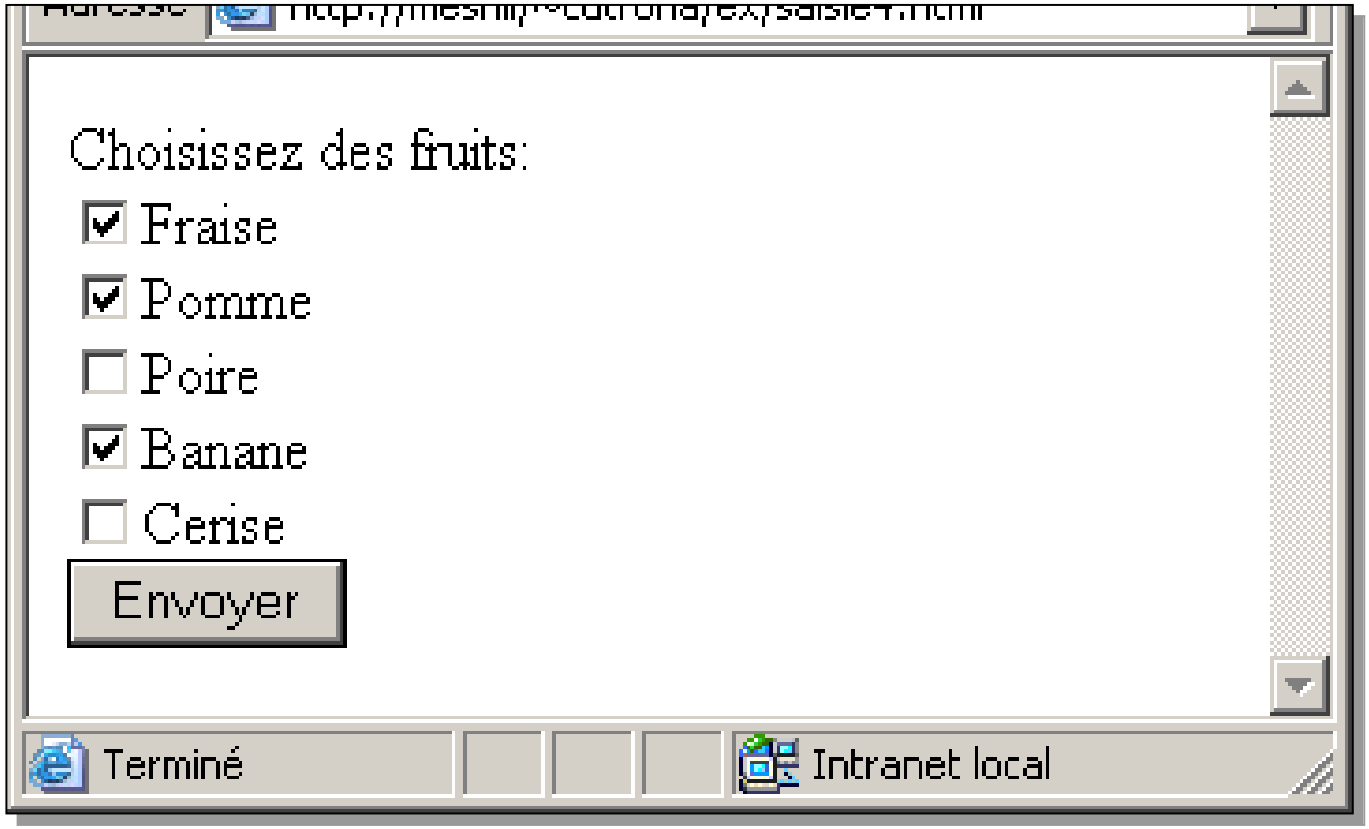


Résultat


Fraise
Pomme
Poire
Banane
Cerise
Choisissez des fruits:
Envoyer
Terminé
Intranet local


Vous avez choisi Fraise
Vous avez choisi Pomme
Vous avez choisi Banane
Terminé
Intranet local

Formulaires contenant des champs « CHECKBOX »

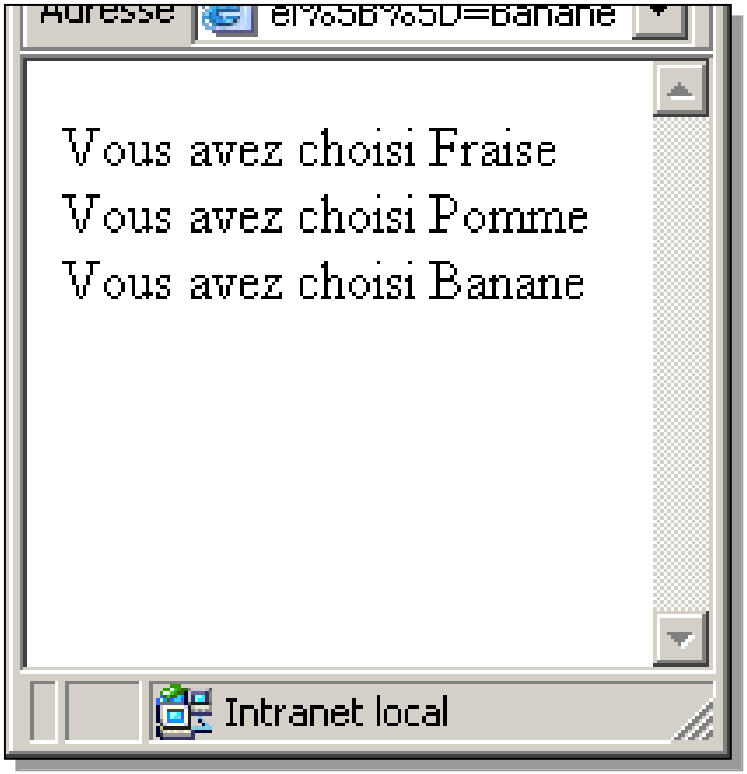


Formulaires contenant des champs « CHECKBOX »

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
   <title>Formulaire de saisie des fruits</title>
</head>
<body>
   <form name="formu" action="valide3.php" method="get">
   Choisissez des fruits :<br>
   <input type="checkbox" name="sel[]" value="Fraise">Fraise<br>
   <input type="checkbox" name="sel[]" value="Pomme" >Pomme <br>
   <input type="checkbox" name="sel[]" value="Poire" >Poire <br>
   <input type="checkbox" name="sel[]" value="Banane">Banane<br>
   <input type="checkbox" name="sel[]" value="Cerise">Cerise<br>
   <input type="submit" value="Envoyer">
   </form>
</body>
</html>
```

Résultat


Choisissez des fruits:
Fraise
Pomme
Poire
Banane
Cerise
Envoyer
Intranet local

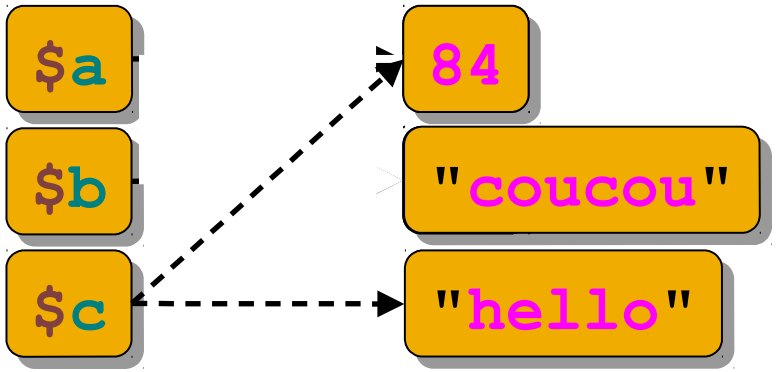

Vous avez choisi Fraise
Vous avez choisi Pomme
Vous avez choisi Banane
Intranet local

Références

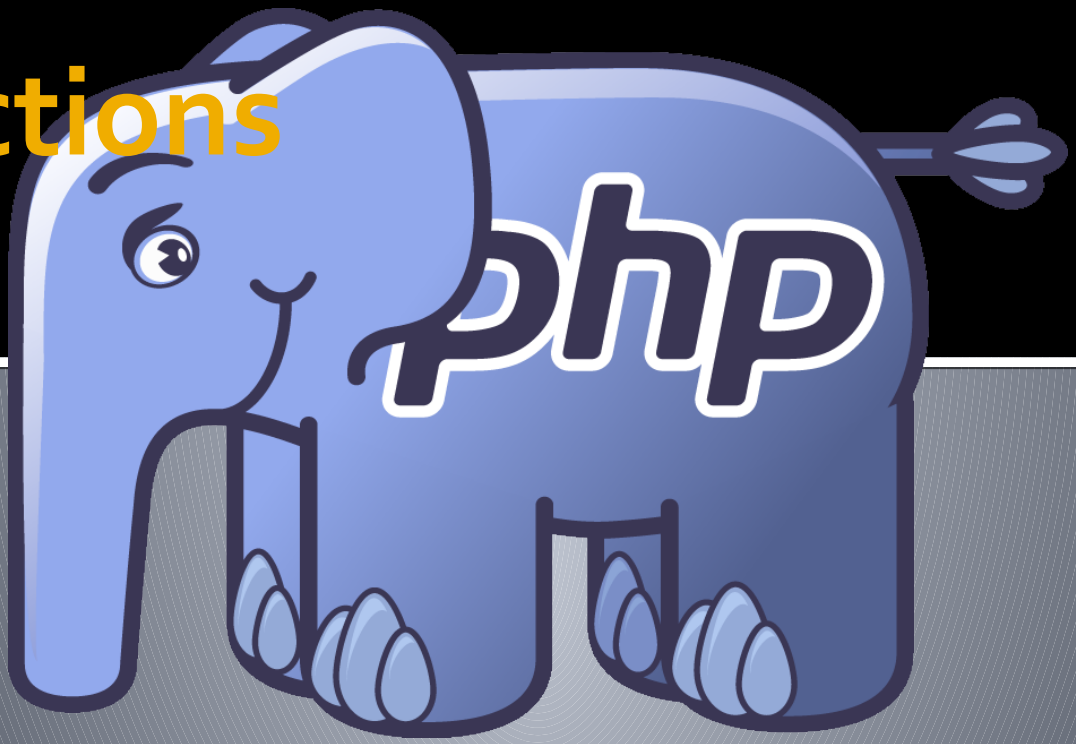
```php
$a = 12 ;
$b = $a ;
$c = &$a ;
$b = "coucou" ;
```

```
$a : 84
$b : coucou
$c : 84
```

```php
unset($c) ;
$c = "hello" ;
```



Fonctions



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

- Valeur de retour

```
function moyenne($a,$b)
{
```

Typage faible de PHP :
Aucune information

- Arguments

```
function moyenne( $a, $b)
{   …   }
```

- Variables défi… …on
- ➔ Visibles et ac… …ction

Typage faible de PHP :
Aucune information

Mode de passage des arguments
(types natifs)

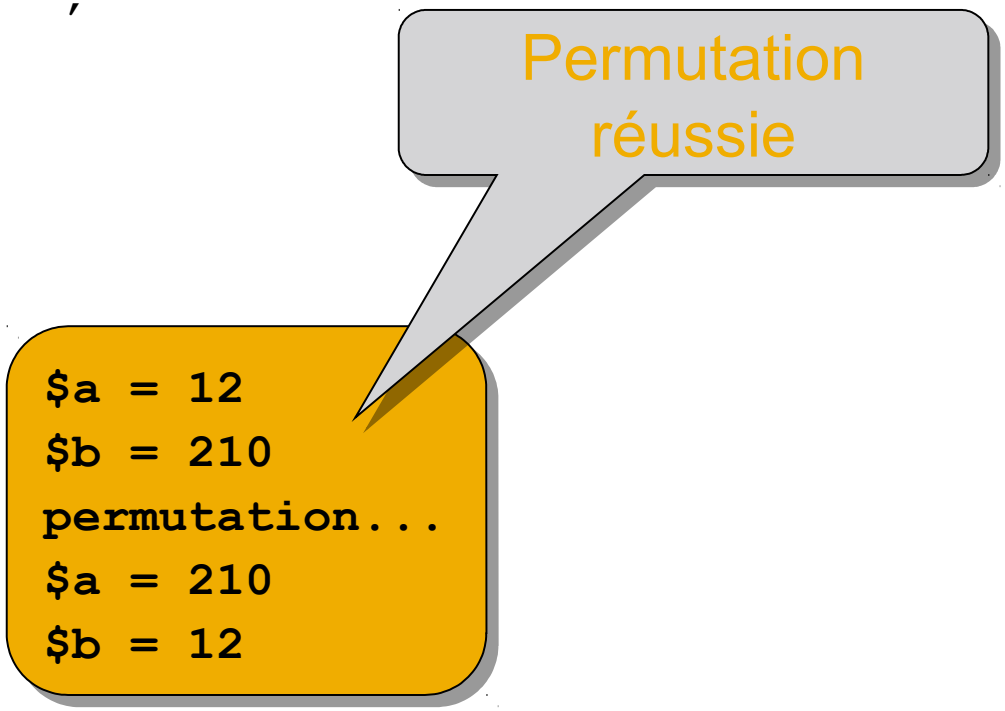
```php
<?php
function permutation($x, $y) {
   echo "permutation..." ;
   $t = $x ;
   $x = $y ;
   $y = $t ;
}
$a = 12 ;
$b = 210 ;
echo "\$a = $a" ;
echo "\$b = $b" ;
permutation($a, $b) ;
echo "\$a = $a" ;
echo "\$b = $b" ;
?>
```

Permutation impossible : Passage des arguments des fonctions par valeur

```
$a = 12
$b = 210
permutation...
$a = 12
$b = 210
```

Mode de passage des arguments
(types natifs)

```php
<?php
function permutation(&$x, &$y) {
   echo "permutation..." ;
   $t = $x ;
   $x = $y ;
   $y = $t ;
}
$a = 12 ;
$b = 210 ;
echo "\$a = $a" ;
echo "\$b = $b" ;
permutation($a, $b) ;
echo "\$a = $a" ;
echo "\$b = $b" ;
?>
```



Arguments par défaut des fonctions

- Valeur par défaut d'un argument s'il n'a pas été défini lors de l'appel de la fonction

```
function bonjour($nom="inconnu")
{ echo "Bonjour cher $nom" ; }
```

- Utilisation

```
bonjour();
```

Bonjour cher inconnu

```
bonjour("Marcel");
```

Bonjour cher Marcel

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

include et require

Fichier mafonction.php

```
<?
function mafonction($arg)
{
   …
}
```

Problème avec include :
- produit un warning
- le script continue

Problème avec require :
- produit un fatal error
- le script s'arrête

Fichier utilisation1.php

```
…
require("mafonction.php")
mafonction(true) ;
…
```

Fichier utilisation2.php

```
…
include("mafonction.php")
…
$var=false ;
mafonction($var) ;
…
```

Fichier utilisation3.php

```
…
require("mafonction.php")
…
```

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

Gestion des erreurs – (Mauvais) Exemple

PHP

```
<?php
$html = <<<HTML
<html>
<head>
<title>die-exit</title>
</head>
<body>
HTML;
if (!isset($val)) {
  die($html."problèm
  /* Au delà de ce point,
     fin du script */
}
$html .= <<<HTML
Choix: $val
</body>
</html>
```

HTML

```
<html>
<head>
<title>die-exit</title>
</head>
<body>
problème val
```

HTML non valide…

Navigateur

problème val

Gestion de l'affichage des erreurs

- `int error_reporting(`

Ancien niveau d'erreur

Sur un serveur en production, **toute erreur affichée** donne des indices sur les scripts et **rend le site vulnérable**

php.ini

`display_errors boolean`

Constante
E_ERROR
E_WARNING
E_PARSE
E_NOTICE
E_CORE_ERROR
E_CORE_WARNING
E_COMPILE_ERROR
E_COMPILE_WARNING
E_USER_ERROR
E_USER_WARNING
E_USER_NOTICE
E_ALL
E_STRICT

Opérateur de contrôle d'erreur en phase de développement

Fichier absent

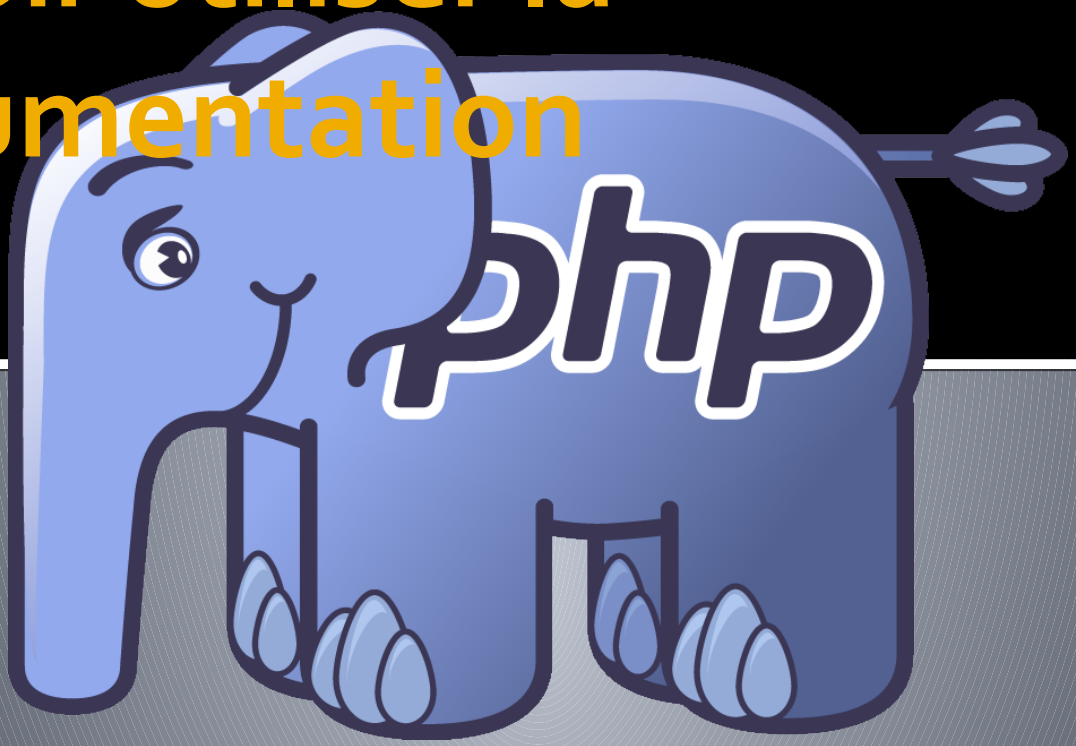
```
$v = file("dummy.txt")
```

```
Warning: file(dummy.txt): failed to open
stream: No such file or directory in
dummy.php on line 68
Problème de lecture
```

```
$v = @file("dummy.txt")
   or die("Problème de lecture") ;
```

```
Problème de lecture
```

Savoir utiliser la documentation



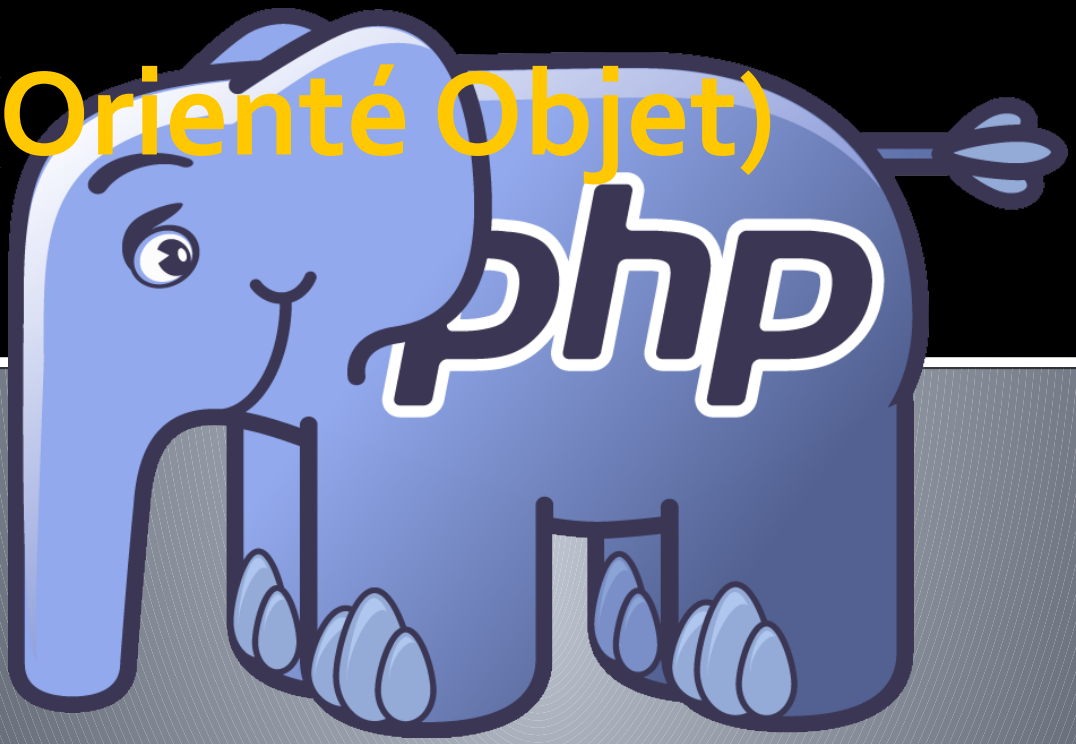
PROGRAMMATION MODULAIRES

- La programmation modulaire permet de la réutilisation de code, notamment par l'écriture de librairies.
- De ce fait, PHP permet cette modularité par la programmation de librairies classiques et de classes.
- **Librairies:** Les librairies sont des fichiers PHP traditionnels. Leur extension était .inc par convention, mais de plus en plus l'extension .PHP est utilisée.

PROGRAMMATION MODULAIRES

- On peut également inclure un fichier HTML ou d'autre type, cependant les éventuels tags PHP ne seront pas interprétés.
- On inclus un fichier en utilisant les deux instructions **include** ou **require**.

OO (Orienté Objet)



Programmation Orientée Objet

- PHP dispose des concepts de POO (Programmation Orientée Objet) au travers des classes.
- Rappelons d'abord qu'un objet possède des attributs et des méthodes, et doit utiliser les mécanismes d'héritage et de polymorphisme.
- Attribut caractéristique d'un objet.
- Méthode action qui s'applique à un objet

Programmation Orientée Objet

- Héritage définition d'un objet comme appartenant à la même famille qu'un autre objet plus général, dont il hérite des attributs et des méthodes.
- Polymorphisme capacité d'un ensemble d'objet à exécuter des méthodes de même nom, mais dont le comportement est propre à chacune des différentes versions.

Programmation Orientée Objet

- Les classes: Une classe est la description complète d'un objet. Elle comprend la déclaration des attributs ainsi que l'implémentation des méthodes de cet objet.
- La création d'un objet est déclenchée par celle d'une instance de la classe qui le décrit.

Programmation Orientée Objet

- Une bibliothèque de composants est un ensemble de fichiers contenant des définitions de classes, que l'on peut inclure en tête des programmes qui utilisent ces classes.
- Les classes peuvent être implémentées à l'aide d'autres classes. Elles sont alors définies selon le principe des couches successives, par empilage des classes de haut niveau sur des classes de bas niveau (cf. les fonctions).

Programmation Orientée Objet

- **Déclaration:** La déclaration d'une classe s'appuie sur le mot clé class. La syntaxe est comparable à celle de la déclaration des fonctions.

```
class ma_classe {
...
}
```

Programmation Orientée Objet

- **Affectation:** Pour exploiter les méthodes et les propriétés d'un objet, on utilise un accesseur dont la syntaxe est constituée des caractères « - » et « > » côte à côte : « -> »

```
$objet_test -> ma_méthode() ; // appelle la méthode
$objet_test -> ma_propriété ; // accède à la propriété
```

Programmation Orientée Objet

- **Opérateur de la classe courante**: $this-> est l'opérateur de self-référence. On peut utiliser un espace pour plus de lisibilité

```
$this->nb_roues = 4 ;
$this -> nb_roues = 4 ;
```

- Les méthodes se déclarent comme des fonctions.

Programmation Orientée Objet

- **Co**
  cor
  la c
  aut
  cla

```
class Véhicule
{
var $nb_roues;
function Véhicule( $nb_roues )
{
$this-> nb_roues= $nb_roues;
}
function NbRoues()
{
return $this-> nb_roues;
}
...
}
$moto= new Véhicule( 2 );
```

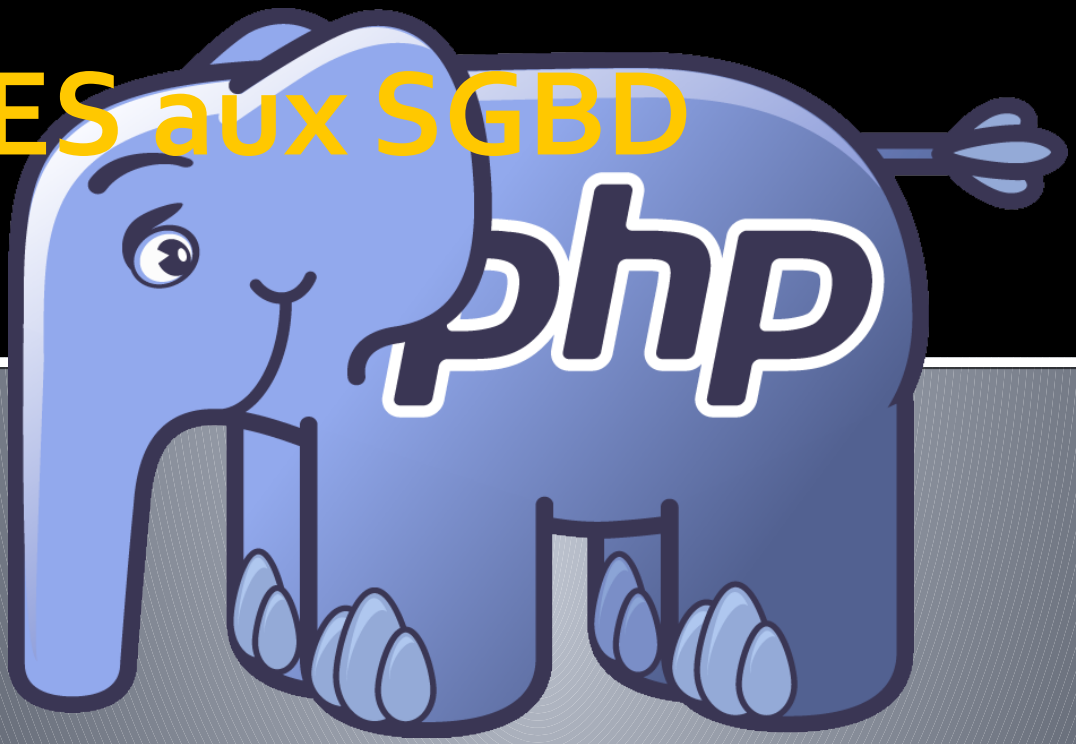
are
nom de
on de la

Programmation Orientée Objet

- **Hé**
- en
- ***Re***

```
class Automobile extends
Véhicule
{
var $marque= "";
function Automobile( $marque,
$nb_roues )
{
$this-> Véhicule( $nb_roues );
// appel constructeur classe
parente
$this-> marque= $marque;
// set de la marque
}
}
```

ACCES aux SGBD



Accès aux SGBD

- En général, la communication entre un programme et une base de données suit le schéma suivant :

Accès aux SGBD

- En programmation PHP, il existe 2 méthodes pour mettre en place cette architecture :
  - accéder nativement à la base par l'intermédiaire de l'API de son middleware associé,
  - 2.passer par ODBC, l'avantage d'ODBC est de proposer une API unifiée quelque soit le SGBD utilisé.
- En plus d'ODBC, PHP gère en accès natifs de nombreux SGBD :
  - Oracle, Sybase, Informix, MySQL, Adabas, Empress, FilePro, InterBase, mSQL, PostgreSQL, Solid, SQLServer, Unix Dbm.

Accès aux SGBD

- L'utilisation en général d'un SGBD (tel que MySQL) avec PHP s'effectue en 5 temps :
  - 1. Connexion au serveur de données
  - 2. Sélection de la base de données
  - 3. Requête
  - 4. Exploitation des requêtes
  - 5. Fermeture de la connexion

Accès aux SGBD

- **Connexion au serveur de données:** Pour se c… e 2…
  - …
  - Ouverture d'une connexion persistante avec la fonction mysql_pconnect

```
<php?
if( mysql_connect("ma_base" , $login , $password ) > 0 )
echo "Connexion réussie ! " ;
else
echo "Connexion impossible ! " ;
?>
```

- ***Remarque :*** la deuxième méthode diffère de la première par le fait que la connexion reste active après la fin du script.

Accès aux SGBD

- **Sélection de la base de données:** Pour faire ce
my
pa
- ***Re*** e peuvent être faites en même temps, mais il est plus simple surtout pour une seule base, de sélectionner la table avant de commencer les requêtes. Ainsi, toutes les requêtes à venir utiliseront cette base par défaut.

```
<php?
if( mysql_select_db("ma_base" ) == True )
echo "Sélection de la base réussie" ;
else
echo "Sélection de la base impossible" ;
?>
```

Accès aux SGBD

- **Envoi d'une requête** Pour envoyer ces re
  - r s
  - mysql_db_query dans le cas où l'on voudrait sélectionner la base en même temps.

```
<php?
$requête = "SELECT * FROM membres WHERE pseudo =
'président' ";
$résultat = mysql_query( $requête );
?>
```

Accès aux SGBD

- **Exploitation des requêtes:** Après l'exécution d'une requête de sélection, les données ne sont pas "affichées", elles sont simplement mises en mémoire, il faut les chercher, enregistrement par enregistrement, et les afficher avec un minimum de traitement.
- PHP gère un pointeur de résultat, c'est celui qui est pointé qui sera retourné.

Accès aux SGBD

- Lorsque vous utilisez une fonction de lecture, le pointeur est déplacé sur l'enregistrement suivant et ainsi de suite jusqu'à ce qu'il n'y en ait plus.
- Les fonctions qui retournent un enregistrement sont :
  - mysql_fetch_row, mysql_fetch_array et mysql_fetch_objec

Et prennent comme paramètre l'identifiant de la requête.

Accès aux SGBD

- **Fermeture de la connexion:** Vous pouvez fermer la connexion au moyen de la fonction mysql_close, mais il est bon de savoir que cette opération sera faite lorsque le script se terminera. C'est donc une opération *facultative.*