Premier programme

- Un premier programme “index.php” :

```
<html>
 <head>
  <title>Ma page PHP</title>
 </head>
 <body>
  <?
   echo "Bonjour,";
   echo "On est le ".date('d/M/Y');
  ?>
 </body>
</html>
```

- La balise <? permet d’entrer dans du code PHP
- La balise ?> permet de sortir du code PHP

Inclusion de fichiers

index.php :

```
<html>
 <head>
  <title>Titre</title>
 </head>
 <body>
  <?
  require("tete.inc.php");
  include("corps.html");
  require("pied.inc.php");
  ?>
 </body>
</html>
```

Mais aussi :

include_once et **require_once**

tete.inc.php :

```
<?
 echo "Bienvenue<br>";
?>
```

corps.html :

```
Corps du site<br>
```

pied.inc.php :

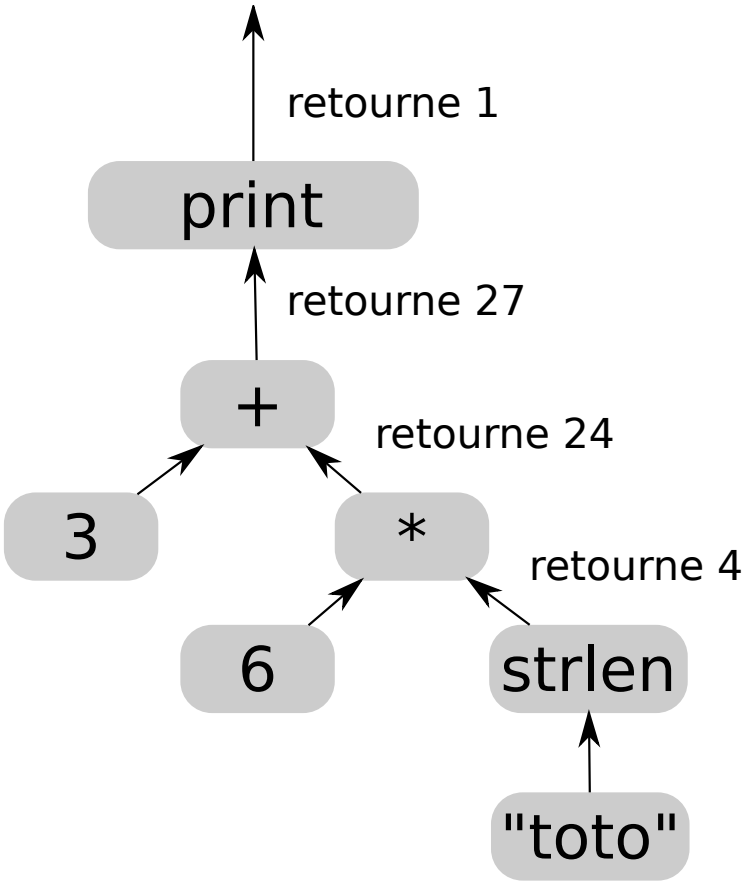
```
<?
 echo date('d/M/Y');
?>
```

Commentaires

```
<html>
 <head>
  <title>Titre</title>
 </head>
 <body>
  <?
   echo "Bonjour"; // commentaire
   echo "Salut"; /* commentaire
    sur plusieurs lignes. */
   echo "Coucou"; # commentaire
  ?>
  <!-- commentaire  -->
 </body>
</html>
```

Instructions, opérations et fonctions

```
<?
 print(3+6*strlen("toto"));
?>
```



Variables

- En C ou en Java, à une variable sont associés :
  - Un nom (ou identifiant);
  - Un type;
  - Une zone mémoire (désignée par une adresse).

```
int a;
a = 2;
```

- En PHP, à une variable sont associés :
  - Un nom (ou identifiant) commençant par $;
  - Un conteneur d'une valeur.

```
<?
$a = 2;
?>
```

Les types des valeurs

- Les variables ne sont pas typées mais les valeurs ont un type :
  - **integer :** `7, 14, 255, 0xFF`
  - **boolean :** `TRUE, FALSE`
  - **double :** `1.95, 1.12e4`
  - **string :** `"bonjour", 'bonjour'`
  - **array :** `array(1,2,3)`
  - **object :** `new maclasse`
  - **ressource :** `mysql_connect("localhost", "moi", "")`
  - **null :** `null, NULL`

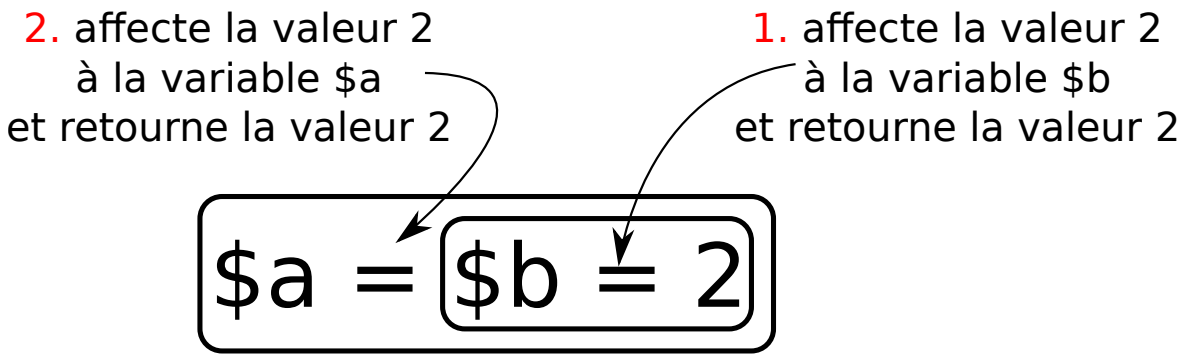
```php
<?
 $a = 2;
 var_dump($a); // affiche int(2)
?>
```

Opérateur d'assignation

- En PHP, on ne déclare pas les variables
- L'opérateur = affecte la valeur d'une expression à une variable :

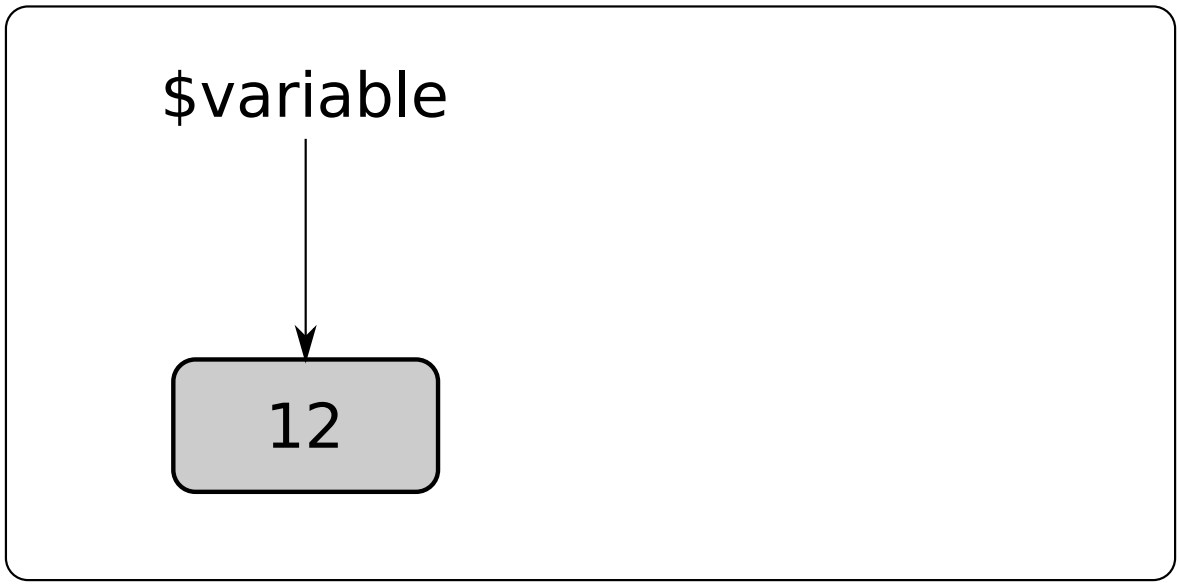
```
<?
 $a = expression;
?>
```

- L'opérateur = retourne la valeur de l'expression assignée à la variable



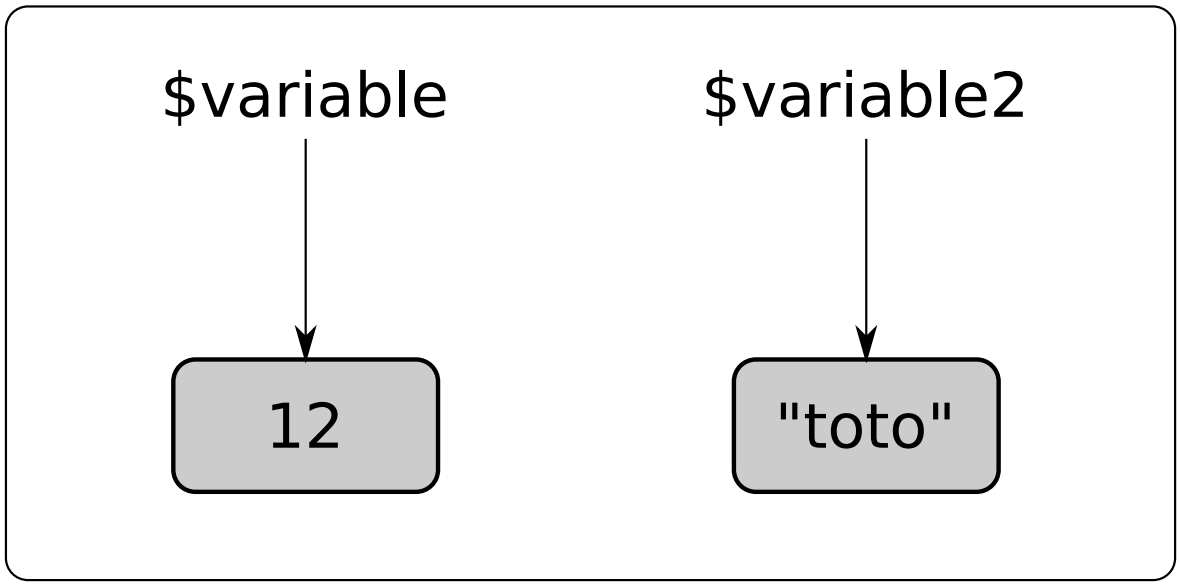
Affectations de valeur

```php
<?
 $variable = 12;
 $variable2 = "toto";
 $variable2 = $variable;
 $variable = 12.12+3;
?>
```



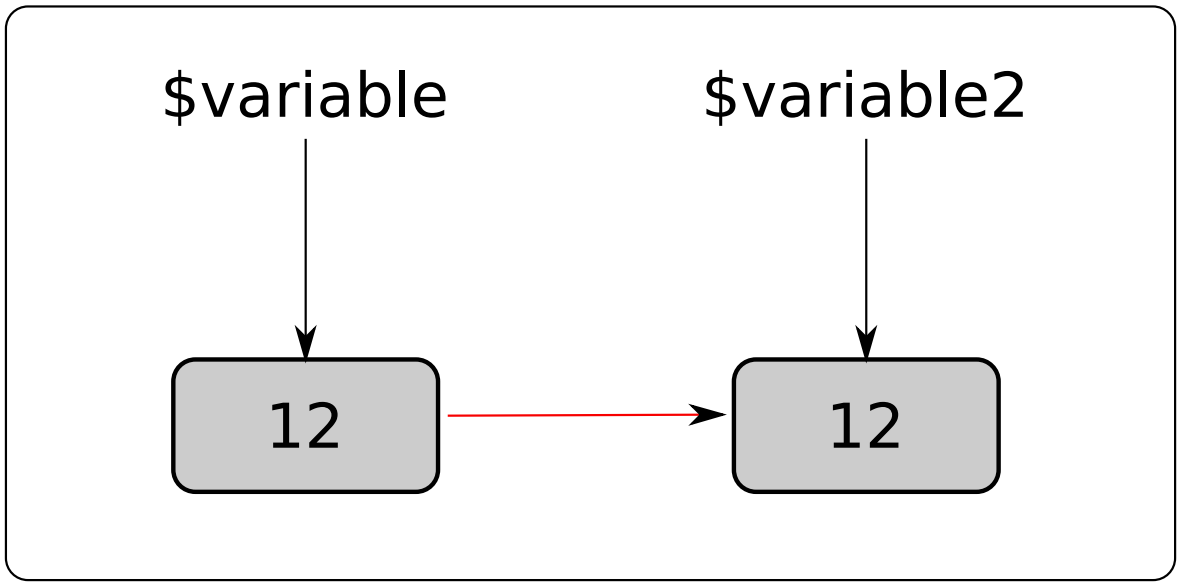
Affectations de valeur

```php
<?
 $variable = 12;
 $variable2 = "toto";
 $variable2 = $variable;
 $variable = 12.12+3;
?>
```



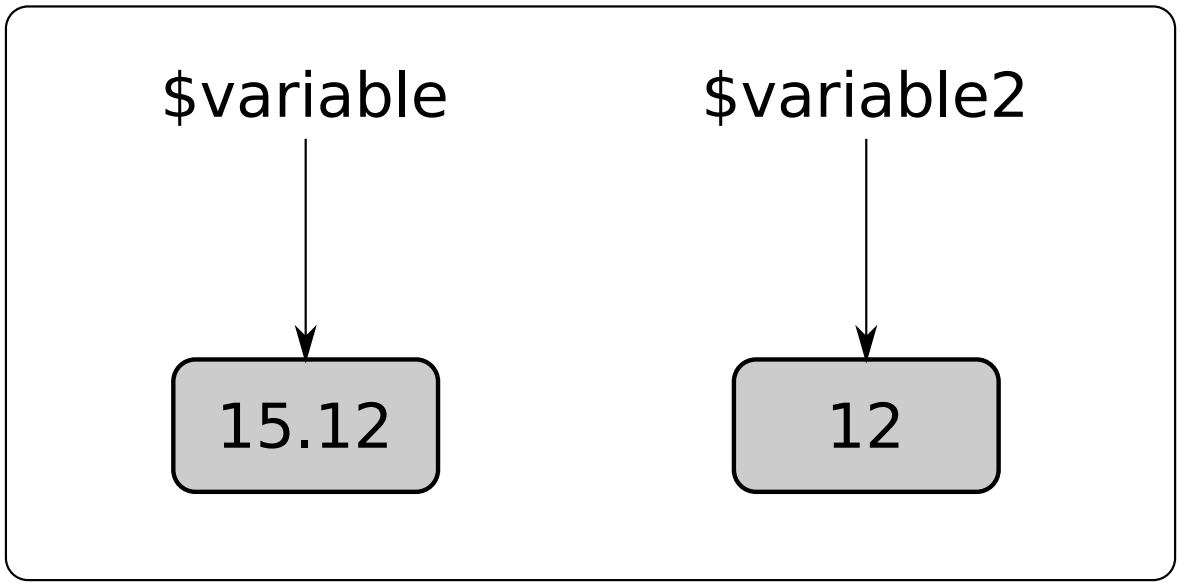
Affectations de valeur

```php
<?
 $variable = 12;
 $variable2 = "toto";
 $variable2 = $variable;
 $variable = 12.12+3;
?>
```



Affectations de valeur

```php
<?
 $variable = 12;
 $variable2 = "toto";
 $variable2 = $variable;
 $variable = 12.12+3;
?>
```



Opérateur d'assignation de référence

- Affectation de référence : l'opérande de droite est une variable précédée du caractère `'&'` :

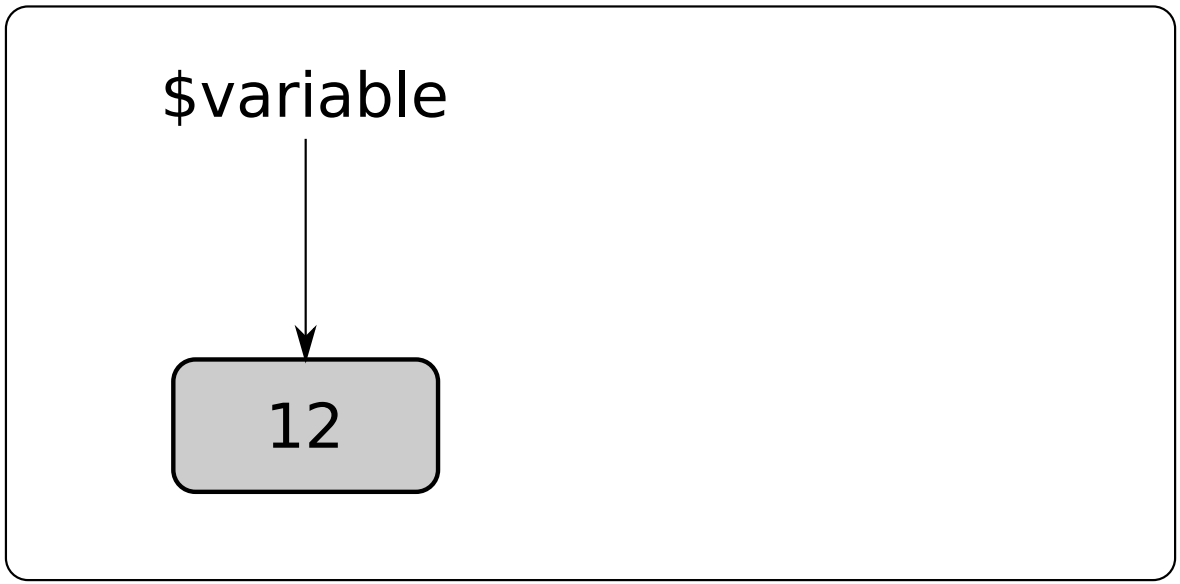
  ```
  $var2 = &$var1;
  ```

- Ici, l'opérateur = retourne la valeur présente dans le conteneur de la variable `$var1`.

- Après l'affectation, la variable `$var2` ne fait que référencer le conteneur associé à la variable `$var1`.

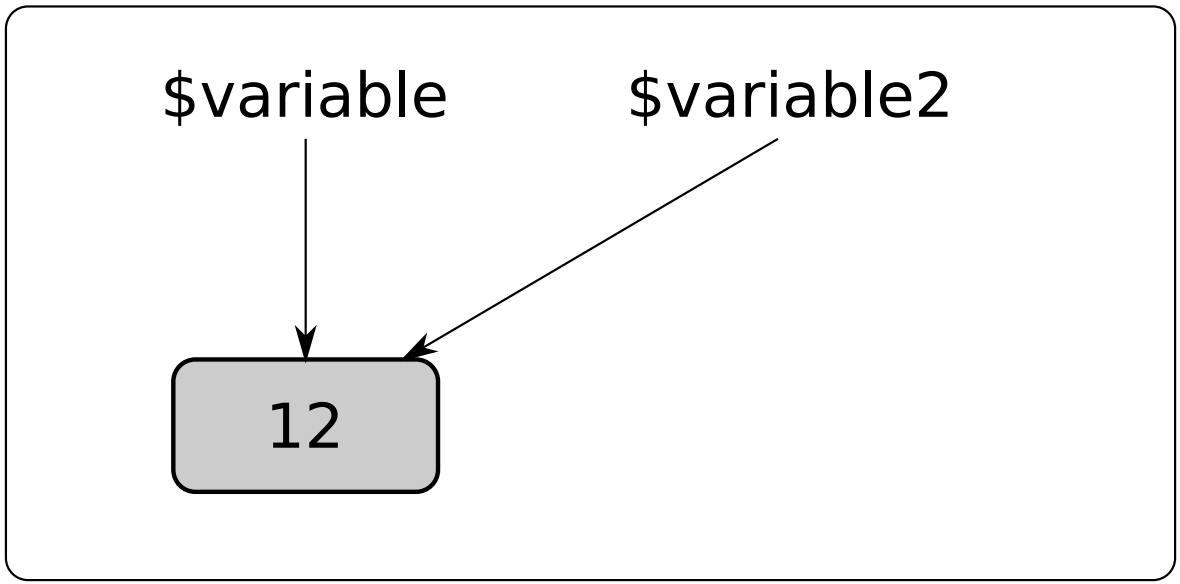
Affectations de référence

```php
<?
 $variable = 12;
 $variable2 = &variable;
 $variable2 = "toto";
 $variable = 12.12;
?>
```



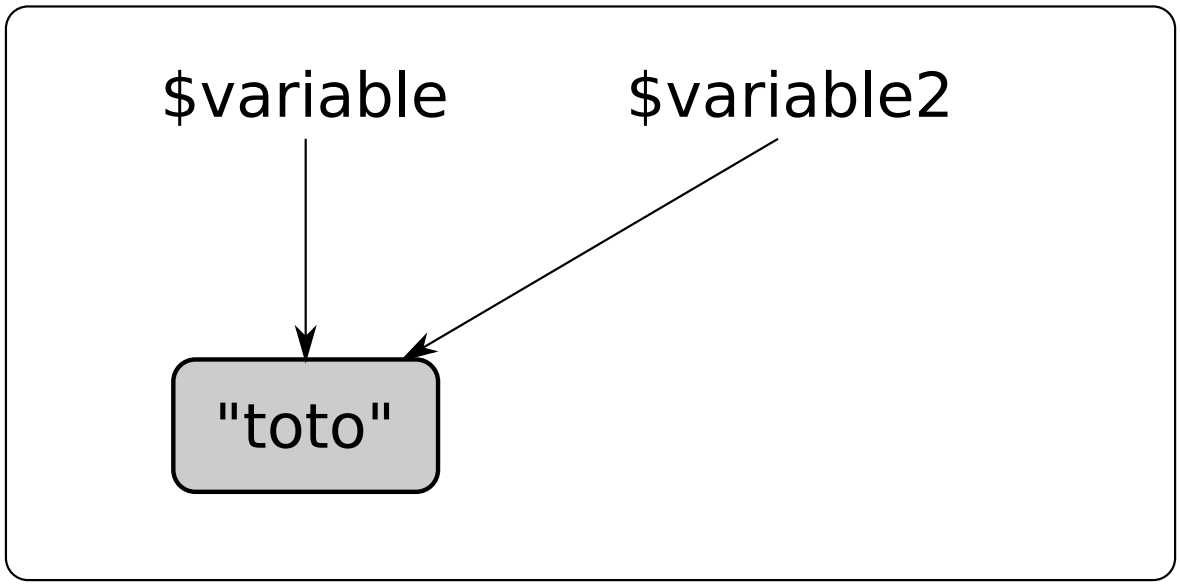
Affectations de référence

```php
<?
 $variable = 12;
 $variable2 = &variable;
 $variable2 = "toto";
 $variable = 12.12;
?>
```



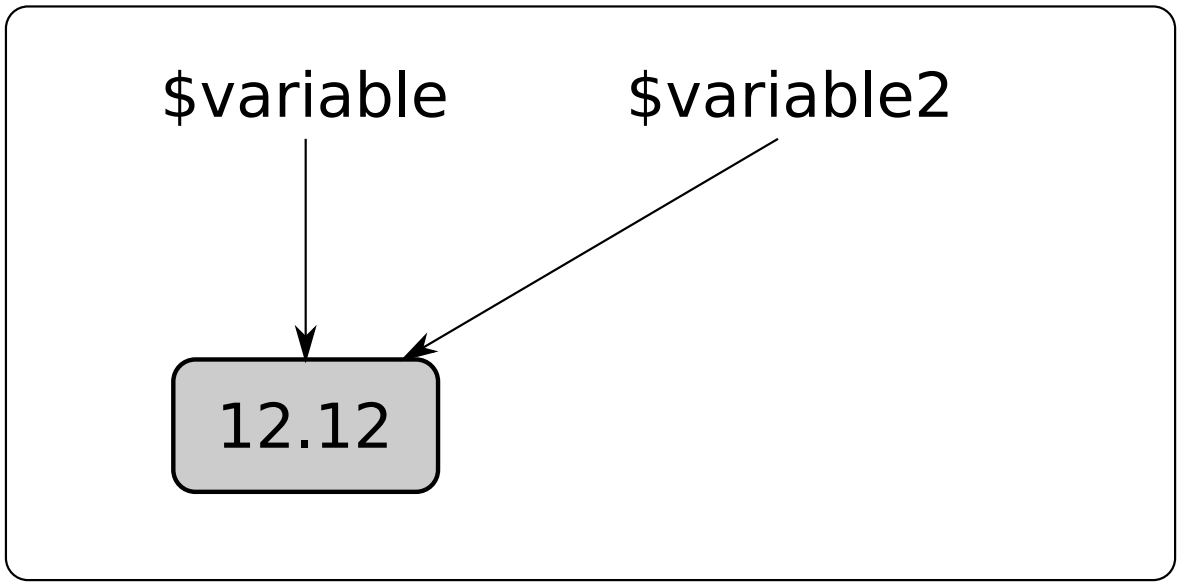
Affectations de référence

```php
<?
 $variable = 12;
 $variable2 = &variable;
 $variable2 = "toto";
 $variable = 12.12;
?>
```



Affectations de référence

```php
<?
 $variable = 12;
 $variable2 = &variable;
 $variable2 = "toto";
 $variable = 12.12;
?>
```



État d'une variable

- La fonction `isset($var)` retourne :
  - `FALSE` si la variable n'est pas initialisée ou a la valeur `NULL`;
  - `TRUE` sinon.
- La fonction `empty($var)` retourne :
  - `TRUE` si une des conditions suivantes est vérifiée :
    - la variable n'est pas initialisée
    - la variable a la valeur `""` (chaîne vide)
    - la variable a la valeur `0` (entier)
    - la variable a la valeur `0.0` (flottant)
    - la variable a la valeur `"0"`
    - la variable a la valeur `NULL`
    - la variable a la valeur `FALSE`
    - la variable a la valeur `array()` (tableau vide)
  - `FALSE` sinon.
- La fonction `unset($var)` détruit une variable.

Type d'une variable

- Pour connaître le type de la valeur contenue dans le conteneur d'une variable `$var` :
  - `gettype($var)` retourne une chaîne de caractères contenant le type de la valeur (ex : `"integer"`)
  - `is_integer($var)` ou `is_int($var)`, `is_double($var)`, `is_scalar($var)`, `is_string($var)`, `is_bool($var)` , `is_array($var)` , `is_object($var)`, `is_ressource($var)`, `is_numeric($var)`

```php
<?
  $var = 12;
  if (is_integer($var)) {
    echo "je suis un entier";
  }
?>
```

Conversion de type

- Opérateur de Cast :
  - `$var2 = (nouveau_type)$var`
  - ou même `$var = (nouveau_type)$var`

```php
<?
  $var = "4.34 litre";
  $var = (double)$var;
  echo $var; // affiche 4.34
  $var = (integer)$var;
  echo $var; // affiche 4
  $var = (boolean)$var;
  echo $var; // affiche 1
?>
```

- On peut aussi utiliser la fonction `settype($var, "nouveau_type")`

Les constantes

- Pour définir une constante :

  `define("MA_CONSTANTE", 12.76, TRUE);`
  ↪ si le dernier paramètre vaut `TRUE`, le nom est insensible à la casse

- Pour savoir si une constante existe :

  `defined("MA_CONSTANTE")`
  ↪ retourne `TRUE` si la constante existe, `FALSE` sinon

- Utilisation d'une constante :

```php
<?
  define("TOTO", 12.45, TRUE);
  echo TOTO, "<br/>";
  echo ToTo, "<br/>";
  if (defined("TOTO"))  echo "ok";
?>
```

Opérateurs numériques

Négation	-$a	Opposé de $a
Addition	$a + $b	Somme de $a et $b
Soustraction	$a - $b	Différence de $a et $b
Multiplication	$a * $b	Produit de $a et $b
Division	$a / $b	Quotient de $a et $b
Modulo	$a % $b	Reste de $a divisé par $b

Pre-incrémente	++$a	Incrémente $a de 1, puis retourne $a
Post-incrémente	$a++	Retourne $a, puis incrémente $a de 1
Pré-décrémente	--$a	Décrémente $a de 1, puis retourne $a
Post-décrémente	$a--	Retourne $a, puis décrémente $a de 1

Opérateurs logiques

et	`$a and $b`	TRUE si $a et $b valent TRUE
ou	`$a or $b`	TRUE si $a ou $b valent TRUE
ou exclusif	`$a xor $b`	TRUE si $a ou $b est égal TRUE mais pas les deux en même temps
non	`!$a`	TRUE si $a n'est pas égal à TRUE
et	`$a && $b`	TRUE si $a et $b sont égaux TRUE
ou	`$a || $b`	TRUE si $a ou $b est égal TRUE

- Attention à la précédence des opérateurs :

```php
<?
  $e = false || true; // (e = (false || true))
  $e = false or true; // ((e = false) or true)
  $e = false && true; // (e = (false && true))
  $e = false and true; // ((e = false) and true)
?>
```

Opérateurs de comparaison

égal	`$a == $b`	TRUE si $a est égal à $b
identique	`$a === $b`	TRUE si $a et $b sont égaux et ont le même type
différent	`$a != $b`	TRUE si $a est différent de $b
différent	`a <> b+`	TRUE si $a est différent de $b
non identique	`$a !== $b`	TRUE si $a et $b sont différents ou n'ont pas le même type
plus petit	`$a < $b`	TRUE si $a est strictement plus petit que $b
plus grand	`$a > $b`	TRUE si $a est strictement plus grand que $b
inférieur ou égal	`$a <= $b`	TRUE si $a est plus petit ou égal à $b
supérieur ou égal	`$a >= $b`	TRUE si $a est plus grand ou égal à $b

Opérateurs de chaînes

- L'opérateur . permet de concaténer deux chaînes de caractères (comme le + en Java) :

```php
<?
var_dump("Bonj"."our");// string(7) "Bonjour"
var_dump(1 . 2); // string(2) "12"
var_dump(1.2); // float(1.2)
$a = "Bonj";
$a = $a . "our";
var_dump($a); // string(7) "Bonjour"
$a = "Bonj";
$a .= "our";
var_dump($a); // string(7) "Bonjour"
?>
```

Opérateurs de commande

- L'opérateur ` (guillements obliques) permet d'exécuter des commandes shell :

```php
<?
  $output = `ls -al`;
  echo "<pre>$output</pre>";
?>
```

- Remarque : cet opérateur n'est pas actif lorsque le "safemode" est activé ou lorsque la fonction `shell_exec()` est désactivée.

Opérateurs d'affectation combinée

addition	`$a += $b`	additionne $a et $b puis affecte le résultat à $a
soustraction	`$a -= $b`	soustrait $a et $b puis affecte le résultat à $a
multiplication	`$a *= $b`	multiplie $a et $b puis affecte le résultat à $a
division	`$a /= $b`	divise $a et $b puis affecte le résultat à $a
modulo	`$a %= $b`	divise $a et $b puis affecte le reste à $a
concaténation	`$a .= $b`	concatène $a et $b puis affecte le résultat à $a

```php
<?
  $a = 13;
  $a += 12;
  echo $a; // affiche 25
?>
```

Block d'instructions

- Comme en **C** ou en **Java**, on définit un block d'instructions à l'aide des accolades ouvrantes et fermantes { } :

```
<?
  if ($a == 2) {
    echo "instruction 1";
    echo "instruction 2";
  }
?>
```

if, boucles while et do...while

- On utilise le **if**, du **while** et le **do...while** de la même façon qu'en **C** ou qu'en **Java** :

```php
<?
  $a = 1;
  if ($a == 2) echo "oui"; else echo "non";
  while ($a < 4) {
   echo $a;
   $a++;
  }
  do {
   echo $a;
   $a--;
  } while ($a>0);
?>
```

boucle for

- La syntaxe du **for** est la même qu'en **C** ou qu'en **Java** :

```
for (expression; expression; expression) instruction;
```

```
<?
  for ($a=0; $a<10; $a++) {
    echo $a.":";
    for ($b=0; $b<10; $b+=2)
    echo ($a+$b)." ";
    echo "<br/>";
  }
?>
```

break et continue

- la commande **break** arrête l'exécution de la boucle :

```
<?
for ($i = 0; $i < 5; $i++) {
 if ($tab[$i]=="bonjour") break;
 echo $tab[$i];
}
?>
```

Truc
Toto
Bonjour
Bip
Salut

- la commande continue arrête l'itération en cours de la boucle :

```
<?
for ($i = 0; $i < 5; $i++) {
 if ($tab[$i]=="bonjour") continue;
 echo $tab[$i];
}
?>
```

Truc
Toto
Bonjour
Bip
Salut

switch

```php
<?
switch ($a) {
case 0 :
 echo '0';
 break;
case 1 :
 echo '1';
 break;
default :
 echo 'default';
}
?>
```

```php
<?
switch ($a) {
case "a" :
 echo 'a';
 break;
case "b" :
 echo 'b';
 break;
default :
 echo 'default';
}
?>
```

Fonctions

```php
<?
 function ajouter(&$a /* 1 */ , $b=5 /* 2 */) {
   $a+=$b;
 }

 $n = 12;
 ajouter($n, 2);
 var_dump($n); // affiche int(14)
 ajouter($n);
 var_dump($n); // affiche int(19)
?>
```

1. Passage d'un paramètre par référence.
2. La valeur par défaut du paramètre est 5.

Portée des variables

```php
<?
 function modif() {
  $var = "salut";
 }

 $var = "toto";
 var_dump($var); // 1
 modif();
 var_dump($var); // 2
?>
```

1. `string(4) "toto"`
2. `string(4) "toto"`

```php
<?
 function modif() {
  global $var;
  $var = "salut";
 }

 $var = "toto";
 var_dump($var); // 3
 modif();
 var_dump($var); // 4
?>
```

3. `string(4) "toto"`
4. `string(5) "salut"`

Affichage des chaînes

```php
<?
 $a = "bonjour";
 $b = "salut";
 $c = 2;

 echo "$a $b\n"; // 1
 echo '$a $b\n'; // 2
 echo "\n";
 echo "$a $b{$c}\n"; // 3
 echo date('d')."\n"; // 4
 echo "date('d')\n"; // 5
?>
```

1. bonjour salut
2. $a $b\n
3. bonjour l
4. 18
5. date('d')

Les caractères

```php
<?
 $chaine="ABCDEF";
 for ($i = 0; $i<strlen($chaine); $i++) {
   echo ord($chaine{$i})."\n"; // 1
 }

 $chaine="";
 for ($i = 0; $i<6; $i++) {
  $c = rand(65,90);
  $chaine.=chr($c);
 }
 echo "$chaine\n"; // 2
?>
```

1
65
66
67
68
69
70

2 GZXNIY

Affichage formaté

```php
<?
$chaine = "Bonjour";
$nombre = "65";
$valeur = "65535";
$flotant = "12.2345";
printf("%s\n", $chaine); // 1
printf("%c %d\n", $nombre, $nombre); // 2
printf("%x %o\n", $valeur, $valeur); // 3
printf("%'#8.3f\n", $flotant); // 4
$a = sprintf("%'#8.3f", $flotant);
var_dump($a); // 5
$a = array("65", "66","67");
vprintf("%c %c %c\n", $a); // 6
$b = vsprintf("%c %c %c", $a);
var_dump($b); // 7
?>
```

1. Bonjour
2. A 65
3. ffff 177777
4. ##12.235
5. string(8) "##12.235"
6. A B C
7. string(5) "A B C"

Modification de la casse

```php
<?
 $chaine = "PHP est super bien !\n";
 echo strtolower($chaine); /* 1 */
 echo strtoupper($chaine); /* 2 */
 echo ucwords($chaine); /* 3 */
 echo ucfirst($chaine); /* 4 */
?>
```

1. php est super bien !
2. PHP EST SUPER BIEN !
3. PHP Est Super Bien !
4. PHP est super bien !

Gestion des espaces

```php
<?
 $a="    ...Salut././.";
 echo "[".ltrim($a)."]\n"; /* 1 */
 echo "[".ltrim($a," .")."]\n"; /* 2 */
 echo "[".rtrim($a,"./")."]\n"; /* 3 */
 echo "[".trim($a," ./")."]\n"; /* 4 */
?>
```

1. `[...Salut././.]`
2. `[Salut././.]`
3. `[    ...Salut]`
4. `[Salut]`

Caractères spéciaux dans les URL et en XHTML

```php
<?
 $a="<b>d&é&truire</b>";
 $b=htmlentities($a);
 echo $b."\n"; /* 1 */
 $c=html_entity_decode($b);
 echo $c."\n"; /* 2 */
 $b=strip_tags($a);
 echo $b."\n"; /* 3 */
 $b=urlencode($a);
 echo $b."\n"; /* 4 */
 $c=urldecode($b);
 echo $c."\n"; /* 5 */
?>
```

1. `&lt;b&gt;d&eacute;truire&lt;b&gt;`
2. `<b>détruire</b>`
3. `détruire`
4. `%3Cb%3Ed%E9truire%3C%2Fb%3E`
5. `<b>détruire</b>`

Recherche de sous-chaînes

```php
<?
$ch = "bonjour salut bonjour";
$nb = substr_count($ch, "bonjour");
var_dump($nb); /* 1 */
$ch2 = str_replace("bonjour", "salut", $ch);
var_dump($ch2); /* 2 */
$pos = strpos($ch,"salut");
var_dump($pos); /* 3 */
$pos = strpos($ch,"Salut");
var_dump($pos); /* 4 */
$pos = stripos($ch,"Salut");
var_dump($pos); /* 5 */
$ch3 = substr($ch,8,5);
var_dump($ch3); /* 6 */
?>
```

1. `int(2)`
2. `string(17) "salut salut salut"`
3. `int(8)`
4. `bool(false)`
5. `int(8)`
6. `string(5) "salut"`

Comparaison de chaînes de caractères

```php
<?
$ch1=11;
$ch2="11toto";
var_dump($ch1); /* 1 */
var_dump($ch2); /* 2 */
var_dump($ch1==$ch2); /* 3 */
var_dump($ch1===$ch2); /* 4 */
var_dump("$ch1"==$ch2); /* 5 */
var_dump($ch1*$ch2); /* 6 */
var_dump("$ch1"*$ch2); /* 7 */
?>
```

1. `int(11)`
2. `string(6) "11toto"`
3. `bool(true)`
4. `bool(false)`
5. `bool(false)`
6. `int(121)`
7. `int(121)`

Comparaison de chaînes de caractères

```php
<?
var_dump(strcmp("toto2","toto2"));
var_dump(strcmp("toto12","toto2"));
var_dump(strcmp("toto2","toto12"));
var_dump(strcasecmp("toto","ToTo"));
var_dump(strnatcmp("toto12","toto2"));
?>
<?
$ch1 = "abc";
$ch2 = "bcd";
if ($ch1 < $ch2) echo "<"; else echo ">";
?>
```

→ int(0) 1
→ int(-1) 2
→ int(1) 3
→ int(0) 4
→ int(1) 5

Tableaux

- On peut indicer les tableaux avec des entiers ou des chaînes de caractères;
- La fonction `count($tab)` retourne le nombre d'éléments présents dans le tableau.

```php
<?
 $a[2] = 12;
 $a[4] = 23;
 $a["toto"] = 12.13;
 var_dump($a); /* 1 */
 $b = count($a);
 var_dump($b); /* 2 */
?>
```

1
```
array(3) {
  [2]      => int(12)
  [4]      => int(23)
  ["toto"] => float(12.13)
  [clé]    => valeur
}
```

2
```
int(3)
```

Tableaux

- Le mot clé **array** : Il prend un nombre variable de paramètres sous la forme "clé => valeur" (ou simplement "valeur") :

```php
<?
 $a = array(12=>3, "a"=>12.12, 15, "c", "1"=>2);
 var_dump($a); /* 1 */
 $a = array(1,2,3,4);
 var_dump($a); /* 2 */
?>
```

1
```
array(5) {
    [12]    => int(3)
    ["a"]   => float(12.12)
    [13]    => int(15)
    [14]    => string(1) "c"
    [1]     => int(2)
}
```

2
```
array(4) {
    [0]   => int(1)
    [1]   => int(2)
    [2]   => int(3)
    [3]   => int(4)
}
```

Intervalles

```php
<?
$a=range(1,4);
var_dump($a); /* 1 */
$a=range(0,30,10);
var_dump($a); /* 2 */
$a=range('d','g');
var_dump($a); /* 3 */
?>
```

1
```
array(4) {
  [0] => int(1)
  [1] => int(2)
  [2] => int(3)
  [3] => int(4)
}
```

2
```
array(4) {
  [0] => int(0)
  [1] => int(10)
  [2] => int(20)
  [3] => int(30)
}
```

3
```
array(4) {
  [0] => string(1) "d"
  [1] => string(1) "e"
  [2] => string(1) "f"
  [3] => string(1) "g"
}
```

Ré-indexation

```php
<?
 $a = array(1,"a"=>2);
 $a[] = 3;
 var_dump($a); /* 1 */
 unset($a[1]);
 $a[] = 4;
 var_dump($a); /* 2 */
 $a = array_values($a);
 var_dump($a); /* 3 */
?>
```

1
```
array(3) {
  [0]    => int(1)
  ["a"]  => int(2)
  [1]    => int(3)
}
```

2
```
array(3) {
  [0]    => int(1)
  ["a"]  => int(2)
  [2]    => int(4)
}
```

3
```
array(3) {
  [0]    => int(1)
  [1]    => int(2)
  [2]    => int(4)
}
```

Tableaux multidimensionnels

```php
<?
  $a = array(12=>array(1,15=>2), 15=>2);
  var_dump($a); /* 1 */
  var_dump($a[12][15]); /* 2 */
  $a[12][20] = 2;
  var_dump($a[12]); /* 3 */
?>
```

2

```
int(2)
```

1

```
array(2) {
  [12]=> array(3) {
    [0]=> int(1)
    [15]=> int(2)
  }
  [15]=> int(2)
}
```

3

```
array(4) {
  [0]   => int(1)
  [15]  => int(2)
  [20]  => int(2)
}
```

foreach

- La boucle **foreach** parcourt tous les couples "clé ⇒ valeur" contenus dans un tableau :

```php
<?
  $a = array(1=>12, "a"=>12.12, "c"=>3, 4);
  foreach ($a as $k=>$v)
    echo $k."=>".$v."\n"; /* 1 */
  foreach ($a as $v)
    echo $v."\n"; /* 2 */
?>
```

1. 1=>12
   a=>12.12
   c=>3
   2=>4

2. 12
   12.12
   3
   4

reset et each

```php
<?
  $a = array(1=>12, "a"=>12.12, "c"=>3, 4);
  reset($a);
  while ($tab=each($a))
    echo $tab[0]."=>".$tab[1]."\n"; /* 1 */
  reset($a);
  while ($tab=each($a))
    echo $tab["key"]."=>".$tab["value"]."\n"; /* 1 */
?>
```

1 {
1=>12
a=>12.12
c=>3
2=>4

list

- la fonction **list** affecte plusieurs variables simultanément :

```php
<?
  $a = array("a", 12, "c");
  list($x,$y,$z)=$a;
  var_dump($x); /* 1 */
  var_dump($y); /* 2 */
  var_dump($z); /* 3 */
  list($i,,$j)=$a;
  var_dump($i); /* 4 */
  var_dump($j); /* 5 */
  list($i,$j)=$a;
  var_dump($i); /* 6 */
  var_dump($j); /* 7 */
?>
```

1. string(1) "a"
2. int(12)
3. string(1) "c"
4. string(1) "a"
5. string(1) "c"
6. string(1) "a"
7. int(12)

list et each

```php
<?
  $a = array(1=>12, "a"=>12.12, "c"=>3, 4);
  reset($a);
  while (list($k,$v)=each($a))
    echo $k."=>".$v."\n"; /* 1 */
?>
```

1 {
1=>12
a=>12.12
c=>3
2=>4

Manipulation d'un tableau

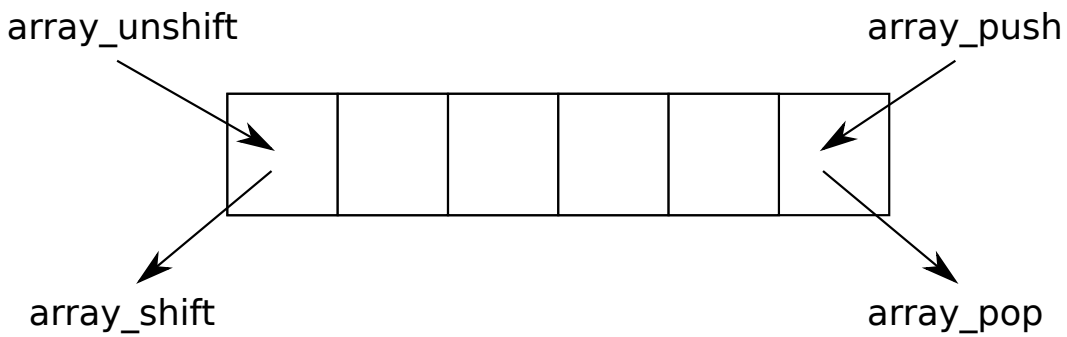
- `array_push($tab, $var, $var2, ...)` :

  empile des valeurs à la fin du tableau
- `$var = array_pop($tab)` :

  dépile une valeur située à la fin du tableau
- `array_unshift($tab, $var, $var2, ...)` :

  ajoute des valeurs au début du tableau
- `$var = array_shift($tab)` :

  supprime et retourne la première valeur du tableau



Manipulation d'un tableau

```php
<?
 $a=array(1, "a"=>2, 3);
 array_push($a, 12.12);
 array_unshift($a, "toto");
 var_dump($a); /* 1 */
 $b = array_pop($a);
 var_dump($b); /* 2 */
 $c = array_shift($a);
 var_dump($c); /* 3 */
?>
```

1.
```
array(5) {
    [0]     => string(4) "toto"
    [1]     => int(1)
    ["a"]   => int(2)
    [2]     => int(3)
    [3]     => float(12.12)
}
```

2. float(12.12)

3. string(4) "toto"

Fusion de tableaux

```php
<?
 $a=array(1,"a"=>2, 3, "b"=>4, 4=>5);
 $b=array("c"=>6, 1=>7, 8, "b"=>9);
 $c=array_merge($a,$b);
 print_r($c); /* 1 */
 $c=array_merge_recursive($a,$b);
 print_r($c); /* 2 */
?>
```

1
```
Array (
  [0] => 1
  [a] => 2
  [1] => 3
  [b] => 9
  [2] => 5
  [c] => 6
  [3] => 7
  [4] => 8
)
```

2
```
Array (
  [0] => 1
  [a] => 2
  [1] => 3
  [b] => Array ( [0] => 4 [1] => 9 )
  [2] => 5
  [c] => 6
  [3] => 7
  [4] => 8
)
```

Intersection et différence de tableaux

```php
<?
 $a=array(1,"a"=>2, 3=>3, "b"=>4, 4=>5);
 $b=array("c"=>1, 1=>3, 4);
 $c=array_intersect($a,$b);
 print_r($c); /* 1 */
 $c=array_diff($a,$b);
 print_r($c); /* 2 */
?>
```

1
```
Array (
    [0] => 1
    [3] => 3
    [b] => 4
)
```

2
```
Array (
    [a] => 2
    [4] => 5
)
```

Tri de tableaux indicés

```php
<?
 $a=array("a10", "b11", "b"=>"a9", "C12", "c12");
 sort($a);
 print_r($c); /* 1 */
 rsort($a);
 print_r($c); /* 2 */
 natsort($a);
 print_r($c); /* 3 */
 natcasesort($a);
 print_r($c); /* 4 */
?>
```

1
```
Array (
  [0] => C12
  [1] => a10
  [2] => a9
  [3] => b11
  [4] => c12
)
```

2
```
Array (
  [0] => c12
  [1] => b11
  [2] => a9
  [3] => a10
  [4] => C12
)
```

1
```
Array (
  [4] => C12
  [2] => a9
  [3] => a10
  [1] => b11
  [0] => c12
)
```

2
```
Array (
  [2] => a9
  [3] => a10
  [1] => b11
  [0] => c12
  [4] => C12
)
```

Tri personalisé

```php
<?
 function comparaison($a, $b) {
   return ($a[0]+$a[1]) - ($b[0]+$b[1]);
 }

 $c = array(array(1,5),array(2,2), array(1,4));
 usort($c, "comparaison");
 print_r($c); /* 1 */
?>
```

1
```
Array (
  [0] => Array ([0] => 2 [1] => 2)
  [1] => Array ([0] => 1 [1] => 4)
  [2] => Array ([0] => 1 [1] => 5)
)
```

Tri de tableaux associatifs

```php
<?
 $a = array("a"=>"c", "b"=>"a", "c"=>"d");
 asort($a);
 print_r($a); /* 1 */
 arsort($a);
 print_r($a); /* 2 */
 sort($a);
 print_r($a); /* 3 */
?>
```

1
```
Array (
  [b] => a
  [a] => c
  [c] => d
)
```

2
```
Array (
  [c] => d
  [a] => c
  [b] => a
)
```

3
```
Array (
  [0] => a
  [1] => c
  [2] => d
)
```

Tri de tableaux associatifs

```php
<?
 $a = array("a"=>"c", "b"=>"a", "c"=>"d");
 ksort($a);
 print_r($a); /* 1 */
 krsort($a);
 print_r($a); /* 2 */
?>
```

1.
```
Array (
  [a] => c
  [b] => a
  [c] => d
)
```

2.
```
Array (
  [c] => d
  [b] => a
  [a] => c
)
```

Tri de tableaux associatifs

```php
<?
 function compar1($a,$b) {
  return ($a[0]+$a[1])-($b[0]+$b[1]);
 }

 function compar2($a,$b) {
  return strlen($a) - strlen($b);
 }

 $a = array("aa"=>array(0,1),
            "aaa"=>array(2,2),
            "a"=>array(1,2));
 uasort($a,"compar1");
 print_r($a); /* 1 */
 uksort($a,"compar2");
 print_r($a); /* 2 */
?>
```

1
```
Array (
  [aa] => Array([0] => 0 [1] => 1)
  [a] => Array([0] => 1 [1] => 2)
  [aaa] => Array([0] => 2 [1] => 2)
)
```

2
```
Array (
  [a] => Array([0] => 1 [1] => 2)
  [aa] => Array([0] => 0 [1] => 1)
  [aaa] => Array([0] => 2 [1] => 2)
)
```

Tri de tableaux associatifs

```php
<?
 function filtre($a) {
  return ($a[0] <= $a[1]);
 }

 $a = array("a"=>array(0,1),
            "b"=>array(3,2),
            "c"=>array(1,2),
            "d"=>array(1,0));

 $selection = array_filter($a, "filtre");
 print_r($selection); /* 1 */
?>
```

1
```
Array (
  [a] => Array([0] => 0 [1] => 1)
  [c] => Array([0] => 1 [1] => 2)
)
```

Appliquer une fonction à un tableau

```php
<?
 function affichage($a) {
  echo "<b>".$a[0]."</b> : ".$a[1]."<br/>\n"; /* 1 */
 }

 $a = array(array(0,1),
            array(3,2),
            array(1,2),
            array(1,0));

 array_walk($a, "affichage");
?>
```

1
```
<b>0</b> : 1<br/>
<b>3</b> : 2<br/>
<b>1</b> : 2<br/>
<b>1</b> : 0<br/>
```

Chaînes et tableaux

```php
<?
 $ch = "J'aime le PHP. Vive le web!";
 $tab = explode(' ',$ch);
 print_r($tab); /* 1 */
 $tab = explode('.',$ch);
 print_r($tab); /* 2 */
 $tab = array("J'aime", "le", "PHP.");
 $ch = implode(" ",$tab);
 echo $ch."\n"; /* 3 */
 $ch = implode("--",$tab);
 echo $ch."\n"; /* 4 */
?>
```

1
```
Array (
  [0] => J'aime
  [1] => le
  [2] => PHP.
  [3] => Vive
  [4] => le
  [5] => web!
)
```

2
```
Array (
  [0] => J'aime le PHP
  [1] => Vive le web!
)
```

3 J'aime le PHP.

4 J'aime--le--PHP.

Autres fonctions utiles sur les tableaux

- La fonction `array_unique($tab)` supprime les valeurs en double dans le tableau (une seule clé est conservée).
- La fonction `$slice = array_slice($t, $p, $n)` extrait les `$n` éléments du tableau `$t` à partir de la position `$p`.
  (voir la documentation pour les autres utilisations)
- La fonction `shuffle($a)` mélange les éléments d'un tableau et renumérote les éléments.