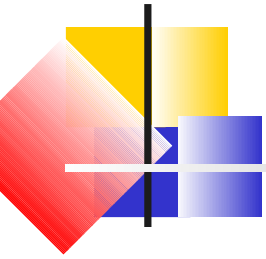


Les tableaux

Principe

Création à l'aide de la fonction array()

Uniquement des tableaux à une dimension

Les éléments d'un tableau peuvent avoir pour élément d'autres tableaux

Les éléments d'un tableau peuvent appartenir à des types distincts

L'index d'un tableau en PHP commence à 0

Pas de limites supérieures pour les tableaux

La fonction count() pour avoir le nombre d'éléments d'un tableau (count($tab))

Remplir un tableau

On peut affecter une valeur à chaque élément.

```
<?php
$tableau[] = "a";
$tableau[] = "b";
?>
```

Les deux valeurs auront pour indice les indices suivant l'indice le plus important. Dans notre cas présent, ce seront les indices 0 et 1

On peut fixer la valeur des indices.

```
<?php
$tableau[0] = "a";
$tableau[1] = "b";
?>
```

Lorsque l'on souhaite insérer plusieurs valeurs, on utilisera la fonction **array()**

```
<?php
$a1 = array("un", "deux", "trois", "nous irons au bois");
$a2 = array(1 => "un", 2 => "deux", 3 => "trois", 4 => "nous irons au bois");
$a3 = array(1 => "un", "deux", "trois", "nous irons au bois");
// pour commencer à 1
?>
```

Les tableaux indicés

Accéder aux éléments par l'intermédiaire de nombres

```
$tableau[indice] = valeur;
$jour[3] = "Mercredi";
$note[0] = 20;

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

$variable = $tableau[indice];
$JS = $jour[6]; // affecte "Samedi" à $JS
echo $note[1] + $note[5]; // affiche 35
```

Les tableaux indicés

Accéder aux éléments par l'intermédiaire de nombres

```
$tableau[indice] = valeur;
$jour[3] = "Mercredi";
$note[0] = 20;

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

$variable = $tableau[indice];
$JS = $jour[6]; // affecte "Samedi" à $JS
echo $note[1] + $note[5]; // affiche 35
```

Les tableaux associatifs

Les éléments sont référencés par des chaînes de caractères : la clé d'index

```
$tableau["indice"] = valeur;
$jour["Dimanche"] = 7
$jour["Mercredi"] = "Le jour des enfants"

$tableau = array(ind0 => val0, ind1 => val1,..., indN => valN);
$jour = array("Dimanche" => 1, "Lundi" => 2, "Mardi" => 3,
  "Mercredi" => 4, "Jeudi" => 5, "Vendredi" => 6, "Samedi" =>
  7);

$variable = $tableau["indice"];
$JS = $jour["Vendredi"]; // affecte 6 à $JS
echo $jour["Lundi"]; // affiche la valeur 2
```

Lecture des éléments d'un tableau I

Avec une boucle for

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

\n ne sert que pour la présentation, afin que le code source semble "humain»

Lecture des éléments d'un tableau II

Avec une boucle foreach

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

```
$jour = array("Lundi", "Mardi", "Mercredi", "Jeudi",
  "Vendredi", "Samedi", " Dimanche");
$i = 0;
foreach($jour as $JS){
   echo "Le jour ".($i+1)." de la semaine est ".$JS."<br />";
   $i++;
}
```

Lecture des éléments d'un tableau associatif

Réalisable en ajoutant la clé associée avant la variable $valeur

```
$tableau = array(cle1 => val1, cle2 => val2, ..., cleN
  => valN);
foreach($tableau as $cle => $valeur) {
      echo "Valeur ($cle): $valeur";
   }

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

Pour déboguer...

En cas de besoin, sachez qu'il est possible d'utiliser **print**_r, afin d'obtenir des informations sur la variable :

```
<pre>
<?php
    $a = array ('a' => 'pomme', 'b' => 'banane',
                        'c' => array ('x', 'y', 'z'));
    print_r ($a);
?>
</pre>
```

Résultat

```
<pre>
Array
(
    [a] => pomme
    [b] => banane
    [c] => Array
        (
            [0] => x
            [1] => y
            [2] => z
        )
)
</pre>
```

Pour déboguer...

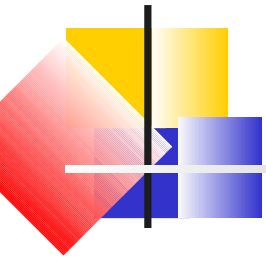
On peut aussi se servir de **var_dump()**, qui affiche les types détectés :

```
<pre>
<?php
    $a = array ('a' => 'pomme', 'b' => 'banane',
                        'c' => array ('x', 'y', 'z'));
    var_dump($a);
?>
</pre>
```

Résultat

```
<pre>
array(3) {
  ["a"]=>
  string(5) "pomme"
  ["b"]=>
  string(6) "banane"
  ["c"]=>
  array(3) {
    [0]=>
    string(1) "x"
    [1]=>
    string(1) "y"
    [2]=>
    string(1) "z"
  }
}

</pre>
```



Données issues d'un formulaire

Les données issues d'un formulaire sont récupérées par une méthode GET ou une méthode POST

L'attribut action du formulaire détermine où les récupérer.

Elles appartiennent à un tableau associatif **`$_GET`** ou **`$_POST`**

Entrée des données coté client

```
<html>
<body>
<form action="acquisition_form.php" method="post">
<pre>
Prénom :      <input type="text" name="prenom" /><br />
Nom :         <input type="text" name="nom" /><br />
Adresse :     <input type="text" name="adresse" /><br />
Ville :       <input type="text" name="ville" /><br />
Code Postal : <input type="text" name="cp" /><br />
<input type="submit" value="Envoyer" /> <input type="reset"
    value="Réinitialiser" />
</pre>
</form>
</body>
</html>
```

Récupération des données coté serveur

acquisition_form.php

```
<html><body>
<h2>Résultats de l'acquisition du formulaire</h2>
<?php
    foreach($_POST as $cle =>$valeur){
    echo "La valeur de  la clé $cle est $valeur <br />\n";}
?>
</body></html>
```

Résultats de l'acquisition du formulaire

La valeur de la clé prenom est Arthur
La valeur de la clé nom est Rimbaud
La valeur de la clé adresse est rue de la gare
La valeur de la clé ville est Béthune
La valeur de la clé cp est 59123

Les tableaux $_GET et $_POST

Les données appartiennent à un tableau

```
$_POST = array("prenom"=> "Arthur", "nom"=>
  "Rimbaud","adresse"=> "rue de la gare",
  "ville"=> "Béthune", "cp"=>"59123")
```

Toutes les données sont récupérées dans le type string

Utilisation $_POST

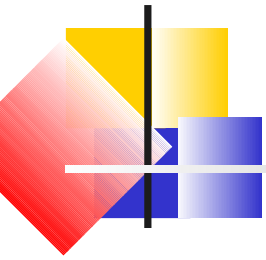
```
<?php
$civil1 = $_POST["prenom"];//Arthur
$civil2 = $_POST["nom"];//Rimbaud
$rue = $_POST["adresse"];// rue de la gare
$cite = $_POST["ville"];// Béthune
$code = $_POST["cp"];//59123
echo "Bonjour $civil1 $civil2, vous habitez $cite <br />";
//affiche Bonjour Arthur Rimbaud, vous habitez Béthune
echo 'Bonjour '. $_POST["prenom"].''.$_POST[nom].',
vous habitez '.$_POST["ville"].'<br />';
//affiche Bonjour Arthur Rimbaud, vous habitez Béthune
echo 'Bonjour $_POST["prenom"] $_POST["nom"],
vous habitez $_POST["ville"] <br />';
//affiche Bonjour $_POST["prenom"] $_POST["nom"], vous habitez
   $_POST["ville"]
?>
```

Variables dans une chaîne

Dès qu'un signe $ est rencontré, l'analyseur PHP va lire autant de caractères qu'il peut pour former **un nom de variable** valide.

Entourez le nom de la variable avec des accolades pour indiquer explicitement son nom (en cas de besoin).

```
<?php
$boisson = 'vin';
echo "Du $boisson, du pain et du fromage !”;
// OK, car "," n'est pas autorisé dans les noms de variables
echo "Il a goûté plusieurs $boissons";
// Pas OK, car 's' peut entrer dans un nom de variable, et
   PHP recherche $boissons
echo "Il a goûté plusieurs ${boisson}s"; // OK
?>
```



Accès aux caractères d'une chaîne

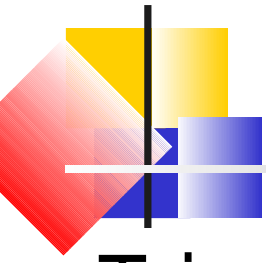
Les caractères d'une chaîne sont accessibles en spécifiant leur offset entre crochets, après le nom de la variable.

Le premier caractère est d'offset 0.

```
/* Premier caractère d'une chaîne */
      $first = $chaine[0];
/* Dernier caractère d'une chaîne. */
      $last = $chaine[strlen($chaine)-1];
```

Exemples de chaînes

```php
<?php
$str = "Ceci est une chaîne"; // Affectation d'une chaîne.
$str = $str . " avec un peu plus de texte"; // Concaténation.
$str .= " et une ligne à la fin.\n"; //autre méthode de concaténation
/* Cette chaîne finira comme : '<p>Nombre: 9</p>' */
$num = 9;
$str = "<p>Nombre: $num</p>";
/* Celle-ci sera '<p>Nombre: $num</p>' */
$num = 9;
$str = '<p>Nombre: $num</p>';
/* Premier caractère d'une chaîne */
$str = 'Ceci est un test.';
$first = $str[0]; // $first contient "C"
/* Dernier caractère d'une chaîne. */
$str = 'This is still a test.';
$last = $str[strlen($str)-1]; // $last contient "."
?>
```



Fonctions de tri de tableau I

Tri selon les valeurs

La fonction sort($tab) effectue un tri sur les **valeurs** des éléments d'un tableau selon les codes ASCII :

"a" est après "Z" et "10" est avant "9")

Le tableau initial est modifié et non récupérable dans son ordre original

Pour les tableaux associatifs les clés seront perdues et remplacées par un indice créé après le tri et commençant à 0

La fonction rsort($tab) effectue la même action mais en **ordre inverse** des codes ASCII.

La fonction asort($tab) trie également les valeurs selon le critère des codes ASCII, mais en **préservant les clé**s pour les tableaux associatifs

La fonction arsort($tab) la même action mais **en ordre inverse** des codes ASCII

la fonction natcasesort($tab) effectue un tri dans l'ordre alphabétique non ASCII ("a" et "A" sont avant "b" et "B" et avant "z", et "10" est après "9") (ordre naturel)

Fonctions de tri de tableau II

```
<?php
$tab1 = array("Molière", "Hugo", "Vian", "Sartre");
$tab2 = array("1622"=>"Molière", "1802"=>"Hugo", "1920"=>"Vian",
   "1905"=>"Sartre");
asort($tab1); //***tri de $tab1 ******
echo "<h3> Tri sur les valeurs de \$tab1 </h3> ";
foreach($tab1 as $ind=> $valeur){
       echo "<b> L'élément a pour indice $ind
       et pour valeur $valeur </b> <br /> ";
}
arsort($tab2); // ***tri de $tab2 ****
echo "<h3> Tri sur les valeurs de \$tab2 </h3> ";
foreach($tab2 as $cle => $valeur ){
       echo "<b> L'élément a pour clé $cle
       et pour valeur $valeur </b> <br /> ";
}
?>
```

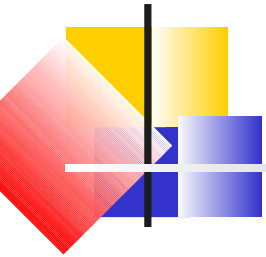
Fonctions de tri de tableau III

Tri sur les valeurs de $tab1

L'élément a pour indice 1 et pour valeur Hugo
L'élément a pour indice 0 et pour valeur Molière
L'élément a pour indice 3 et pour valeur Sartre
L'élément a pour indice 2 et pour valeur Vian

Tri sur les valeurs de $tab2

L'élément a pour clé 1920 et pour valeur Vian
L'élément a pour clé 1905 et pour valeur Sartre
L'élément a pour clé 1622 et pour valeur Molière
L'élément a pour clé 1802 et pour valeur Hugo



Fonctions de tri de tableau IV

Tri sur les clés

La fonction ksort() **trie les clés** du tableau selon le critère des codes ASCII, et préserve les associations clé /valeur

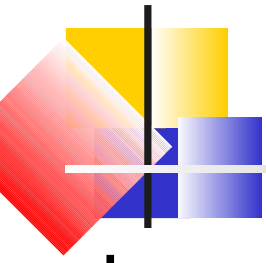
La fonction krsort() effectue la même action mais en **ordre inverse** des codes ASCII

Fonctions de tri de tableau V

```
<?php
$tab2 = array ("1622"=>"Molière", "1802"=>"Hugo",
  "1920"=>"Vian", "1905"=>"Sartre") ;
ksort ($tab2);
echo "<h3 >  Tri sur les clés de \$tab2 </h3>" ;
foreach ($tab2 as $cle=>$valeur) {
   echo "<b> l'élément a pour clé $cle et pour valeur $valeur
    </b> <br />\n";}
?>
```

Tri sur les clés de $tab2

l'élément a pour clé 1622 et pour valeur Molière
l'élément a pour clé 1802 et pour valeur Hugo
l'élément a pour clé 1905 et pour valeur Sartre
l'élément a pour clé 1920 et pour valeur Vian



Les opérateurs

Les opérateurs

les opérateurs de calcul

les opérateurs d'assignation

les opérateurs d'incrémentation

les opérateurs de comparaison

les opérateurs logiques

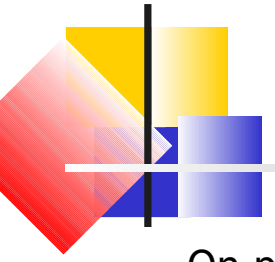
les opérateurs bit-à-bit

autres opérateurs

Les opérateurs de calcul

Les opérateurs de calcul permettent de modifier mathématiquement la valeur d'une variable

+	*addition*	Ajoute deux valeurs		$x+3
-	*soustraction*	Soustrait deux valeurs	$x-3	
*	*multiplication*	Multiplie deux valeurs	$x*3	
/	*division*	Divise deux valeurs	$x/3	
%	*modulo*	Reste de la division entière		$x%3
=	*affectation*	Affecte une valeur à une variable		$x=3



Les opérateurs d'assignation

On peut écrire l'opération : ajouter une valeur dans une variable et stocker le résultat dans la variable (x=x+2 par exemple) sous la forme x+=2

+= addition deux valeurs et stocke le résultat dans la variable (à gauche)
$x+=2 équivaut à $x=$x+2

-= soustrait deux valeurs et stocke le résultat dans la variable
$x-=2 équivaut à $x=$x-2

***=** multiplie deux valeurs et stocke le résultat dans la variable
$x*=2 équivaut à $x=$x*2

/= divise deux valeurs et stocke le résultat dans la variable
$x/=2 équivaut à $x=$x/2

Les opérateurs d'incrémentation

Un opérateur de type **$x++** permet de remplacer des notations telles que **$x=$x+1** ou bien **$x+=1**

++ Incrémentation
Augmente d'une unité la variable
$x++

-- Décrémentation
Diminue d'une unité la variable
$x--

On peut aussi utiliser la notation préfixée
$e=--$x;

Les opérateurs de comparaison

== A ne pas confondre avec le signe d'affectation (=)

opérateur d'égalité Compare deux valeurs et vérifie leur égalité

$x==3 1 si $x est égal à 3, sinon 0

<

opérateur d'infériorité stricte Vérifie qu'une variable est strictement inférieure à une valeur

$x<3 1 si $x est inférieur à 3, sinon 0

<=

opérateur d'infériorité Vérifie qu'une variable est inférieure ou égale à une valeur $x<=3

1 si $x est inférieur à 3, sinon 0

>

opérateur de supériorité stricte Vérifie qu'une variable est strictement supérieure à une valeur

$x>3 1 si $x est supérieur à 3, sinon 0

>=

opérateur de supériorité Vérifie qu'une variable est supérieure ou égale à une valeur

$x>=3 1 si $x est supérieur ou égal à 3, sinon 0

!=

opérateur de différence Vérifie qu'une variable est différente d'une valeur

$x !=3 1 si $x est différent de 3, sinon 0

Enfin, *l'étonnant opérateur* === testant à la fois la valeur et le type

$x===3 1 so $3 vaut 3 et est bien un entier (et non « 3 »)

Les opérateurs logiques (booléens)

Ce type d'opérateur permet de vérifier si plusieurs conditions sont vraies :

|| ou OR

OU logique Vérifie qu'une des conditions est réalisée ((condition1)||(condition2))

&& ou AND

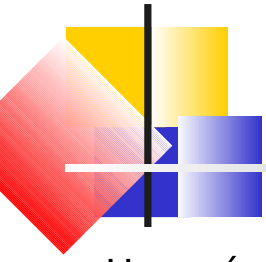
ET logique Vérifie que toutes les conditions sont réalisées ((condition1)&&(condition2))

XOR

OU exclusif Vérifie qu'une et une seule des conditions est réalisée ((condition1)XOR(condition2))

!

NON logique Inverse l'état d'une variable booléenne (!condition)
(retourne 1 si la variable vaut 0, 0 si elle vaut 1)



Les opérateurs bit-à-bit

Un opérateur bit-à-bit traite ses opérandes comme des données binaires.
Les opérateurs suivants effectuent des opérations bit-à-bit,
c'est-à-dire avec des bits de même poids.

&

ET bit-à-bit Retourne 1 si les deux bits de même poids sont à 1

9 & 12 (1001 & 1100) 8 (1000)

|

OU bit-à-bit Retourne 1 si l'un ou l'autre des deux bits de même poids est à 1 (ou les deux)

9 | 12 (1001 | 1100) 13 (1101)

^

XOR bit-à-bit Retourne 1 si l'un des deux bits de même poids est à 1 (mais pas les deux)

9 ^ 12 (1001 ^ 1100) 5 (0101)

Autres opérateurs

.		
Concaténation	Joint deux chaînes bout à bout	"Bonjour"."Au revoir"
&		
Référencement de variable	Définir un pointeur sur une variable	$MaVariable = &$var2;

Les priorités

ordre décroissant des priorités de tous les opérateurs

parenthèses	() []
opérateurs unaires	++ -- !
mult,div,modulo	* / %
addition,soustraction	+ -
opérateurs relationnels	< > <= >=
Égalité	== !=
ET binaire	&
OU exclusif binaire(XOR)	^
OU binaire	|
ET logique	&&
OU logique	||
Affectations diverses	= += -= *= /= %= ^=
	AND
	XOR

Si plusieurs priorités se trouvent dans la même expression et sont de même niveau, ce sera l'opérateur le plus à gauche qui sera effectué en premier. Grâce aux parenthèses qui sont toujours prioritaires, on peut définir l'ordre de calcul des opérateurs. Les parenthèses rendent un programme plus clair, plus lisible.

Les fonctions

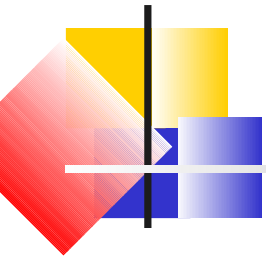
Déclaration et appel d'une fonction

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

le nom doit **commencer par une lettre**

un nom de fonction peut comporter des lettres, des chiffres et les caractères _ et & (les espaces ne sont pas autorisés!)

le nom de la fonction, comme celui des variables est **sensible à la casse** (différenciation entre les minuscules et majuscules)



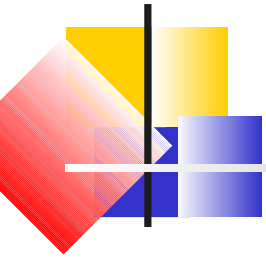
Renvoi d'une valeur par une fonction

La fonction peut **renvoyer une valeur** (et donc se terminer) grâce au mot-clé **`return`**. Lorsque l'instruction **`return`** est rencontrée, la fonction **évalue la valeur qui la suit, puis la renvoie** au programme appelant (programme à partir duquel la fonction a été appelée).

Une fonction peut contenir plusieurs instructions **`return`**, ce sera toutefois **la première instruction `return`** rencontrée qui provoquera la fin de la fonction et le renvoi de la valeur qui la suit.

La syntaxe de l'instruction return est :

```
return valeur_ou_variable;
```



Les arguments d'une fonction

Passer des arguments à une fonction, c'est lui **fournir** une **valeur** ou une variable afin que la fonction puisse effectuer des opérations sur ces arguments.

Le passage d'arguments à une fonction se fait au **moyen d'une liste d'arguments** (séparés par des virgules) entre parenthèses.

Il est possible de **donner une valeur par défaut** à ces arguments en faisant suivre le nom de la variable par le signe "=" puis la valeur que l'on affecte par défaut à la variable. Ainsi, on **soulage l'appelant**.

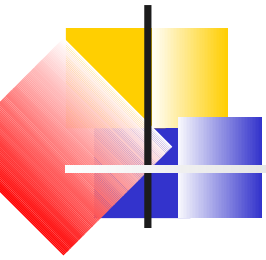
Pour utiliser un argument dans le corps de la fonction en tant que variable, celui-ci doit être précédé par le signe `$`.

Appel de fonction

```
<?php
function direTexte($qui, $texte = 'Bonjour') {
 if(empty($qui))
    { // $qui est vide, on retourne faux
    return false; }
else
    { echo "$texte $qui"; // on affiche le texte
    return true; // fonction exécutée avec succès }}
?>
```

Ainsi cette fonction peut être appelée de deux façons différentes:

```
<?php
// Passage des deux paramètres
     direTexte("cher phpeur", "Bienvenue"); // affiche "Bienvenue cher phpeur"
// Utilisation de la valeur par défaut du deuxième paramètre
     direTexte("cher phpeur"); // affiche "Bonjour cher phpeur"
?>
```



Portée d'une variable

Il existe plusieurs niveaux de définition de variables :

Une variable précédée du mot clé **global** sera visible dans l'ensemble du code, c'est-à-dire que sa portée ne sera pas limitée à la fonction seulement. Ainsi, toutes les fonctions pourront utiliser et modifier cette même variable

Le niveau **static** permet de définir une variable locale à la fonction, qui persiste durant tout le temps d'exécution du script

Par défaut, la variable possède le niveau **local**, c'est-à-dire que la variable ne sera modifiée qu'à l'intérieur de la fonction et retrouvera la valeur qu'elle avait juste avant l'appel de fonction à la sortie de celle-ci

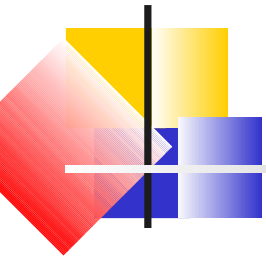
Exemple portée

```
<?
 $chaine = "Nombre de camions : ";
function ajouteCamion($mode='') {
    global $chaine;
    static $nb=0;
    $nb++; // on incrémente le nombre de camions
    if($mode == "affiche")    {
        echo $chaine.$nb;
        } // on affiche le nombre de camions
 }
ajouteCamion(); // nb == 1
ajouteCamion(); // nb == 2
ajouteCamion(); // nb == 3
ajouteCamion("affiche"); // affiche Nombre de camions : 4
?>
```

Retourner plusieurs variables

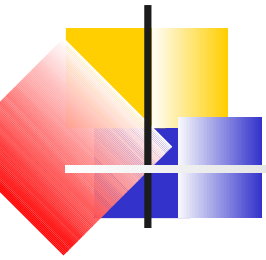
Lorsque vous souhaitez qu'une fonction retourne plusieurs valeurs, le plus simple est d'utiliser un tableau.

```
<?
function nomFonction()
{ .....
   return array( $variable1, $variable2, $variable3 );
   // on retourne les valeurs voulues dans un tableau
}
$retour = nomFonction();
echo "$retour[0] - $retour[1] - $retour[2]";
?>
```



Convention de codage

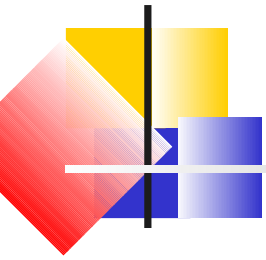
http://pear.php.net/manual/fr/standards.control.php



Définitions des fonctions

La déclaration des fonctions respecte l'indentation classique des accolades :

```
<?php
function nomDeLaFonction($arg1, $arg2 = ' ') {
      if (condition) {
            statement;
      }
      return $val;
 }
?>
```



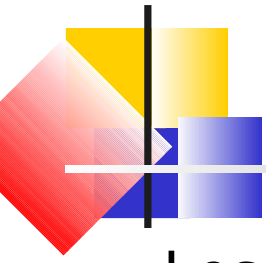
Fonctions et Méthodes

Les fonctions et les méthodes doivent être nommées en utilisant **le style dromadaire** (camel caps) : utilisation de majuscules en milieu de mot pour marquer visuellement la séparation entre les mots sans utiliser d'espaces ou séparateurs.

La première lettre du nom (après le préfixe pour une fonction) est une minuscule, et chaque premier caractère d'un nouveau mot doit être une majuscule.

Quelques exemples :

```
connect()
getData()
buildSomeWidget()
```



Librairies

Les librairies sont des fichiers PHP, leur extension est **`.inc.php`** par convention. On inclut un fichier en utilisant **`include()`** ou **`require()`**.

Il existe une différence importante entre les deux :

Un fichier inclus par **`include`** est inclus dynamiquement, lors de l'exécution du code.

Un fichier inclus par **`require`** est inclus avant l'interprétation du code. Il est équivalent à la directive #include du C

Utilisation include / require

include() et **require()** incluent et exécutent le fichier spécifié en argument.

```
include("menu.inc.php");
```

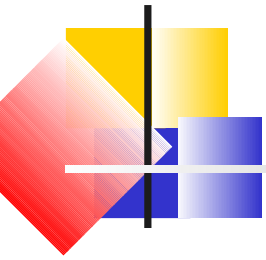
Les deux structures de langage sont identiques, sauf dans la gestion des erreurs :

include() produit une **alerte (warning)**

require() génère une **erreur fatale**

Utiliser **require()** si vous voulez qu'un fichier d'inclusion manquant interrompe votre script.

Avec **include()**, le script continuera son exécution.



Constantes

Les constantes doivent toujours être en majuscules, les mots séparés par des '_'.

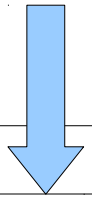
Note : Les constantes **true, false et null** font exception à la règle des majuscules, et doivent toujours être en minuscules.

Quelques exemples :

```
VAR_NAME
C_PATH_ROOT
```

Conseils écriture de code (source CodeIncome)

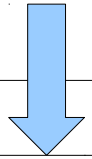
```
if($condition_1 == true) {
    if($condition_2 == true) {
        print "Toutes les conditions sont remplies!";
    }   else if($condition_3 == true) {
            print "Toutes les conditions sont remplies!";
    }
}
```



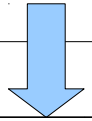
```
if($condition_1 && ($condition_2 || $condition_3)) {
    print "Toutes les conditions sont remplies!";
}
```

Conseils en écriture de code

```
if($age >= 18) {
      $peut_boire = true;
} else {
     $peut_boire = false;
}
```



```
$peut_boire = ($age >= 18) ? true : false;
```

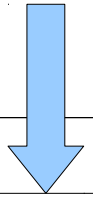


```
$peut_boire = $age >= 18;
```

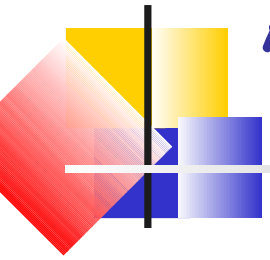
```
$reponse = (defined($_POST['choix']) &&!empty($_POST['choix'])) ?
                $_POST['choix'] : 'valeur par défaut';
```

Conseils écriture

```
$i = 0;
while($i < 100) {
        echo "Entre ici, Jean Moulin !";
        $i++;
}
```



```
for($i = 0; $i < 100; $i++){
        print "Entre ici, Jean Moulin !";
}
```



Autres conseils...

- Aération du code
- Nom de variables (clair mais concis)
- INDENTATION
- Alignement
- Découpage en fonctions, lisibles sur une page
- Commentaires (utiles!!)

Exemples de la documentation

```
<?php
   $allinone =callSomeFunction('param1',       'second',        true);
   $bar       =callSomeFunction('parameter2', 'third',          false);
   $c          =callSomeFunction('3',              'verrrrrrylong', true);
  ....

function superFunct($firstParam = 'foo', $secondParam = 'bar',
       $third = null, $fourthParam = false, $fifthParam = 123.12,
       $sixthParam = true
) { ...
?>
```

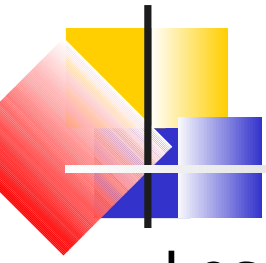
Exemples documentation

```
<?php
   $some_array = array(
        'foo'  => 'bar',
        'spam' => 'ham'
    );
?>
```

PhpDocumentor

```
<?php
/* vim: set expandtab tabstop=4 shiftwidth=4 softtabstop=4: */
/**
 * Short description for file
 *
 *
 * LICENSE: This source file is subject to version 3.01 of the PHP license
 * that is available through the world-wide-web at the following URI:
 * send a note to license@php.
 * If you did not receive a copy
 * the PHP License and are una
 * send a note to license@php.
 * @category   CategoryName
 * @package    PackageName
 * @author     Original Author <author@example.com>
 * @author     Another Author <another@example.com>
 * @copyright  1997-2005 The PHP Group
 * @license    http://www.php.net/license/3_01.txt  PHP License 3.01
 * @version    SVN: $Id$
 * @link       http://pear.php.net/package/PackageName
 * @see        NetOther, Net_Sample::Net_Sample()
 * @since      File available since Release 1.2.0
 * @deprecated File deprecated in Release 2.0.0
 */

?>
```



Librairies

Les librairies sont des fichiers PHP, leur extension est **`.inc.php`** par convention. On inclut un fichier en utilisant **`include()`** ou **`require()`**.

Il existe une différence importante entre les deux :

Un fichier inclus par **`include`** est inclus dynamiquement, lors de l'exécution du code.

Un fichier inclus par **`require`** est inclus avant l'interprétation du code. Il est équivalent à la directive #include du C

Utilisation include / require

include() et **require()** incluent et exécutent le fichier spécifié en argument.

```
include("menu.inc.php");
```

Les deux structures de langage sont identiques, sauf dans la gestion des erreurs :

include() produit une **alerte (warning)**

require() génère une **erreur fatale**

Utiliser **require()** si vous voulez qu'un fichier d'inclusion manquant interrompe votre script.

Avec **include()**, le script continuera son exécution.