LES FONCTIONS ET STRUCTURES DE CONTROLE

Les conditions

1. if

Les instructions de type condition permettent d'exécuter du code si une condition est vraie.

Par exemple, si le prénom est égal à Robert, afficher "Bienvenue" :

```
<?php
$prenom = 'Robert'; //déclaration de la variable $prenom


if ($prenom == 'Robert') //test de la variable $prenom
{
    echo 'Bienvenue';
}
?>
```

Ci-dessus, la valeur Robert est affectée à la variable `$prenom` puis cette variable est testée à l'aide de l'instruction **`if`**.

La syntaxe est donc :

```
if (condition) { instruction }
```

Notez que :

- "est égal à" se note **==**.
- "est différent de" se note **!=**.
- "est inférieur à" se note **<**.
- "est supérieur à" se note **>**.
- "est inférieur ou égal à" se note **<=**.
- "est supérieur ou égal à" se note **>=**.

L'instruction **sinon** se note **`else`**.

Par exemple, si le prénom est égal à Robert, afficher "Bienvenue", sinon "A bientôt" :

```
<?php


$prenom = 'Toto'; //déclaration de la variable $prenom


if ($prenom == 'Robert') //test de la variable $prenom
{
```

```
    echo ’Bienvenue’;
}
else
{
    echo ’A bientôt’;
}
?>
```

Affiche :

A bientôt

En effet, le code teste si la variable `$prenom` est égale à Robert, puis comme ce n’est pas le cas, le code passe dans le **`else`** (sinon) et exécute `echo "A bientôt";`.

Enfin la dernière instruction pour les conditions est **`else if`** appelée **sinon si**.

Cela permet de tester d’autres conditions non testées par le **`if`**.

Par exemple, si le prénom est égal à Robert, afficher "Bienvenue", sinon s’il est égal à Toto, afficher "Bonjour", sinon "A bientôt" :

```
<?php

$prenom = ’Toto’; //déclaration de la variable $prenom

if ($prenom == ’Robert’) //test de la variable $prenom
{
    echo ’Bienvenue’;
}
else if ($prenom == ’Toto’) //test de la variable $prenom
{
    echo ’Bonjour’;
}
else
{
    echo ’A bientôt’;
}
?>
```

Affiche :

Bonjour

Le code teste si $prenom est égal à Robert puis si $prenom est égal à Toto. Et comme $prenom est effectivement égal à Toto, le code exécute echo "Bonjour";.

Il est possible d'ajouter autant d'instructions **else if** que vous voulez.

Si la condition ($prenom == "Robert") est vérifiée, les autres conditions dans les **else if** ne sont même pas vérifiées, donc si vous passez dans le **if**, vous êtes sûr de ne jamais passer dans le ou les **else if** ainsi que dans le **else**.

À l'inverse, si vous écrivez :

```
<?php

$prenom = 'Robert'; //déclaration de la variable $prenom

if ($prenom == 'Robert') //test de la variable $prenom
{
    echo 'Bienvenue';
}

if ($prenom == 'Toto') //test de la variable $prenom
{
    echo 'Bonjour';
}
else
{
    echo 'A bientôt';
}
?>
```

Le code teste si $prenom est égal à Robert puis si $prenom est égal à Toto et comme $prenom est initialisé avec la valeur Robert, ce code affiche "Bienvenue" et "A bientôt".

Vous pouvez aussi imbriquer les **if** les uns à l'intérieur des autres autant de fois que vous le souhaitez.

Par exemple :

```
<?php

$age = 30; //déclaration de la variable $age

if ($age > 20) //test de la variable $age
{
```

```
    if ($age == 30) //test de la variable $age
```

```
    {
       echo 'Bienvenue';
    }
    else {
       echo 'A bientôt';
    }
}
?>
```

Affiche :

Bienvenue

Le programme teste si la variable `$age` est supérieure à 20 puis comme c'est le cas, le programme teste ensuite si `$age` est égal à 30 et exécute `echo "Bienvenue";`.

Enfin, vous pouvez intercaler du code HTML entre les conditions en PHP.

Par exemple :

```
<?php

$prenom = 'Robert'; //déclaration de la variable $prenom

if ($prenom == 'Robert') //test de la variable $prenom
{
?>
    Bonjour  <!--Code HTML-->
<?php
}
else
{
?>
   A bientôt  <!--Code HTML-->
<?php
}
?>
```

est équivalent à :

```php
<?php

$prenom = 'Robert'; //déclaration de la variable $prenom

if ($prenom == 'Robert') //test de la variable $prenom
{
    echo 'Bonjour';
}
else
{
    echo 'A bientôt';
}
?>
```

2. switch

L'instruction **`switch`** est équivalente au **`if`** mais elle est utilisée par le développeur pour plus de clarté dans le code. Elle est équivalente au `if` si vous utilisez un `break` pour sortir du `switch`. Sinon toutes les instructions qui suivent le `case` dans lequel vous êtes entré seront exécutées.

La syntaxe est :

```
switch (condition) {
   case expression: instruction
   case expression: instruction
   ...
}
```

```php
<?php

$prenom = 'Robert'; //déclaration de la variable $prenom

switch ($prenom) //test de la variable $prenom
{
   case 'Robert':
     echo 'Bonjour';
     break;
   case 'Jean':
     echo 'A bientôt';
     break;
}
?>
```

L'instruction `break` provoque la sortie du **`switch`** et donc si `$prenom` est égal à "Robert", le code exécutera `echo "Bonjour";` puis **`break`** et sortira du **`switch`** sans même tester "Jean".

Cela équivaut à utiliser l'instruction **`else if`** à la place de **`if`** :

```
<?php

$prenom = 'Robert'; //déclaration de la variable $prenom

if ($prenom == 'Robert') //test de la variable $prenom
{
    echo 'Bonjour';
}
else if ($prenom == 'Jean')
{
    echo 'A bientôt';
}
?>
```

Voici un exemple si vous n'utilisez pas l'instruction `break` :

```
<?php

$prenom = 'Robert'; //déclaration de la variable $prenom

switch ($prenom) //test de la variable $prenom
{
   case 'Robert': echo 'Bonjour<br />';
   case 'Jean':  echo 'A bientôt';
}
?>
```

Ci-dessus, la valeur Robert est affectée à la variable `$prenom` puis cette variable est testée à l'aide de l'instruction **`switch`**. L'exemple ci-dessus affichera :

Bonjour

A bientôt

L'instruction **`case`** permet de comparer chaque valeur par rapport à la variable `$prenom` puis exécute les instructions après les **:**.

Enfin, vous pouvez ajouter l'instruction `default` pour exécuter du code dans le cas où aucune égalité n'a été trouvée avec les instructions **`case`**.

Par exemple :

```
<?php

$prenom = 'Toto'; //déclaration de la variable $prenom

switch ($prenom) //test de la variable $prenom
{
   case 'Robert':
     echo "Bonjour";
     break;
   case 'Jean':
     echo "A bientôt";
     break;
   default:
     echo "Personne n'a ce nom";
}
?>
```

Affiche :

Personne n'a ce nom

En effet, `$prenom` est égal à "Toto" donc le code ne passe ni dans le **`case`** "Robert", ni dans le **`case`**"Jean" mais dans **`default`**.

L'instruction **`default`** est donc équivalente à l'instruction **`else`**.

Voici un autre exemple avec des chiffres :

```
<?php

$age = 25; //déclaration de la variable $age

switch ($age) //test de la variable $age
{
   case 20:
     echo "Vous avez 20 ans.";
     break;
   case 25:
     echo "Vous avez 25 ans.";
     break;
```

```
   default:
     echo "Vous n'avez ni 20 ans, ni 25 ans.";
}
?>
```

Affiche :

Vous avez 25 ans.

Le **`switch`** est plus pratique à utiliser si vous avez beaucoup de conditions à tester.

Les boucles

1. for

Une boucle permet de répéter *x* fois une exécution de code.

Par exemple, si vous voulez afficher dix fois "Bonjour", il suffit d'écrire avec une boucle **for** :

```
<?php

for ($i = 1; $i <= 10; $i++)
{
   echo 'Bonjour <br />';
}
?>
```

`$i` est une variable représentant le compteur de la boucle. Vous n'êtes pas obligé d'appeler votre variable `$i` mais par convention c'est le nom employé.

La syntaxe est donc :

```
for ($i=nombre de départ ; $i <= nombre d'arrivée ; incrément)
{
  instructions
}
```

`$i++` est équivalent à `$i=$i+1` et représente l'incrémentation de `$i`. Vous pouvez écrire `$i=$i+2`pour incrémenter de 2 ou `$i=$i-1` pour décrémenter.

Autre exemple : pour écrire les nombres entre 100 et 150, vous avez le code suivant :

```
<?php

for ($i = 100; $i <= 150; $i++)
{
```

```
    echo $i.'<br />';
}
?>
```

L'instruction `echo $i.'<br />';` est répétée 50 fois avec à chaque fois `$i` augmentant de 1.

Le `<br />` permet de sauter une ligne entre chaque nombre pour ne pas les afficher tous à la suite.

L'instruction **break** permet d'arrêter la boucle.

Par exemple, si vous voulez afficher cinq fois "Bonjour", il suffit d'écrire avec une boucle **for** :

```
<?php


for ($i = 1; $i <= 10; $i++)
{
    echo 'Bonjour <br />';
     if ($i == 5) {
        break;
    }
}


?>
```

Affiche :

Bonjour

Bonjour

Bonjour

Bonjour

Bonjour

En effet, la boucle s'arrête lorsque `$i` est égal à 5 et non pas à 10.

2. While

La boucle **while** signifie « tant que ». C'est-à-dire que la boucle va s'exécuter tant qu'une condition est vraie.

Par exemple, pour afficher dix fois "Bonjour", il suffit d'écrire avec une boucle **while** :

```
<?php
$i = 1;
```

```
while ($i <= 10)
{
  $i=$i+1;
  echo 'Bonjour <br />';
}
?>
```

`$i` est une variable représentant le compteur de la boucle. Mais cette fois, il faut lire : tant que `$i` est inférieur ou égal à 10 alors une boucle se produit.

La syntaxe est donc :

```
$i=nombre de départ
while ($i <= nombre d'arrivée)
{
  incrément
  instructions
}
```

Il ne faut pas aussi oublier de mettre l'incrémentation de `$i` dans les instructions du **while**, c'est-à-dire entre les accolades, sinon `$i` ne vaut jamais 10 et vous avez une boucle infinie !

Vous remarquez aussi que la valeur de départ de `$i` se met avant la boucle et il faut absolument que cette valeur de départ respecte la condition de la boucle (`$i <= nombre d'arrivée`) pour passer dans la boucle.

Si vous écrivez :

```
<?php
$i = 11;
while ($i <= 10)
{
  $i=$i+1;
  echo 'Bonjour <br />';
}
?>
```

Vous ne passez jamais dans la boucle car `$i` vaut 11 au départ donc la condition de boucle n'est pas satisfaite.

La boucle **while** est équivalente à la boucle **for** mais elle est parfois utile si vous ne connaissez pas à l'avance le nombre de fois que vous allez boucler, en particulier si vous lisez en base de données dans la boucle **while** et que votre condition de sortie de boucle dépend de la valeur lue en base de données.

3. do while

La boucle **do while** signifie faire tant que. C'est-à-dire que la boucle va s'exécuter tant qu'une condition est vraie mais à la différence de la boucle while, l'expression est exécutée au moins une fois.

Par exemple, toujours pour afficher dix fois "Bonjour", il suffit d'écrire avec une boucle **do while** :

```
<?php
$i = 1;
do
{
  $i=$i+1;
  echo 'Bonjour <br />';
} while ($i <= 10)
?>
```

`$i` est une variable représentant le compteur de la boucle. Mais cette fois, il faut lire : faire la boucle tant que `$i` est inférieur ou égal à 10.

La syntaxe est donc :

```
$i=nombre de départ
do
{
  incrément
  instructions
} while ($i <= nombre d'arrivée)
```

4. foreach

La description de cette boucle se situe dans la section suivante sur les tableaux.

Les tableaux

1. Tableaux numériques

Un tableau est comme une variable mais pouvant stocker plusieurs valeurs.

Un tableau est défini par une clé (appelée indice) et une valeur :

Pour créer ce tableau, il faut écrire :

```
<?php
$tableau = array('Jean','Robert','Paul','Joe','Alain');
?>
```

Ici, le tableau s'appelle `$tableau`, mais vous pouvez bien sûr l'appeler autrement.

Vous pouvez aussi accéder directement à la valeur d'un tableau grâce à son indice avec cette syntaxe :

```
<?php
echo $tableau[0];
?>
```

Affiche :

Jean

En effet, `$tableau[x]` est une variable avec pour valeur le *xème* élément du tableau.

Attention : les indices des tableaux commencent à 0 !

Pour remplacer la chaîne de caractères 'Robert' par 'Nadia' à l'indice 1 du tableau, il faut écrire :

```
<?php
$tableau[1] = 'Nadia';
?>
```

Il est aussi possible de créer un tableau vide et de le remplir de cette façon :

```
<?php
$tableau = array();
$tableau[0] = 'Jean';
$tableau[1] = 'Robert';
$tableau[2] = 'Paul';
$tableau[3] = 'Joe';
$tableau[4] = 'Alain';
?>
```

ou de cette façon :

```
<?php
$tableau = array();
$tableau[] = 'Jean';
$tableau[] = 'Robert';
$tableau[] = 'Paul';
$tableau[] = 'Joe';
$tableau[] = 'Alain';
?>
```

ou encore :

```
<?php
$tableau = [’Jean’,’Robert’,’Paul’,’Joe’,’Alain’];
?>
```

PHP remplit automatiquement les indices et cela revient au même que d’écrire :

```
<?php
$tableau = array(’Jean’,’Robert’,’Paul’,’Joe’,’Alain’);
?>
```

2. Tableaux associatifs

Dans un tableau associatif, c’est vous qui décidez de la clé à mettre dans celui-ci.

Par exemple :

Dans cet exemple, la clé peut prendre n’importe quelle valeur. Ce n’est pas forcément un nombre.

Ce type de tableau s’écrit :

```
<?php
$tableau = array(’A1’=>’Jean’,’B4’=>’Robert’,3=>’Paul’,
’Toto’=>’Joe’,’H’=>’Alain’);
?>
```

L’association s’écrit avec les symboles =>. Ce qui est placé avant désigne la clé alors que ce qui est après désigne la valeur.

Une autre façon de renseigner ce tableau est :

```
<?php
$tableau = array();
$tableau[’A1’] = ’Jean’;
$tableau[’B4’] = ’Robert’;
$tableau[3] = ’Paul’;
$tableau[’Toto’] = ’Joe’;
$tableau[’H’] = ’Alain’;
?>
```

Il est pratique d’utiliser ce tableau lorsque la clé représente une information pertinente.

Par exemple, si vous voulez stocker les caractéristiques d'une personne dans un tableau `$personne` :

```
<?php
$personne = array();
$personne['Nom'] = 'Martin';
$personne['Prenom'] = 'Monique';
$personne['Age'] = 50;
?>
```

Ensuite, si vous voulez afficher l'âge, il suffit d'écrire :

```
<?php
echo $personne['Age'];
?>
```

3. Constantes de type tableau

Depuis PHP 5.6, il est possible de définir une constante de type tableau :

```
<?php
const COULEURS = array('rouge', 'vert', 'noir');
echo COULEURS[1];
?>
```

Affiche :

vert

Depuis PHP 7, il est possible d'utiliser le mot-clé **define** pour créer des constantes de type tableau :

```
<?php
define('COULEURS', array(
        'rouge',
        'vert',
        'noir'
));
echo COULEURS[1];
?>
```

Affiche :

vert

4. Parcours d'un tableau

Il existe plusieurs solutions pour parcourir un tableau.

a. La boucle for

```
<?php
//création du tableau
$tableau = array('Jean','Robert','Paul','Joe','Alain');
//boucle sur le tableau
for ($i = 0; $i < 5; $i++) {
   //affichage des valeurs du tableau concaténées
   //avec un saut de ligne
    echo $tableau[$i].'<br />';
}
?>
```

Cet exemple affiche :

Jean

Robert

Paul

Joe

Alain

En effet, `$i` varie de 0 à 4 et donc la boucle affiche `$tableau[0]` valant 'Jean' puis `$tableau[1]` valant 'Robert' et ainsi de suite.

Parfois, les éléments d'une base de données sont récupérés dans un tableau et vous ne connaissez pas à l'avance le nombre d'éléments de ce tableau. Si vous ne connaissez pas la taille du tableau, vous pouvez utiliser soit la fonction **`count()`** soit la fonction **`sizeof()`**. Ces fonctions sont détaillées un peu plus loin.

Par exemple :

```
<?php
//création du tableau
$tableau = array('Jean','Robert','Paul','Joe','Alain');
//boucle sur le tableau
for ($i = 0; $i < sizeof($tableau); $i++) {
   //affichage des valeurs du tableau concaténées
   //avec un saut de ligne
    echo $tableau[$i].'<br />';
}
```

```
?>
```

Cet exemple affiche :

Jean

Robert

Paul

Joe

Alain

b. La boucle foreach

Cette boucle est plus pratique car il n'y a pas besoin de se soucier de la taille du tableau.

La boucle **`foreach`** n'utilise pas de compteur. Elle stocke les valeurs du tableau une par une dans une variable temporaire appelée dans cet exemple `$val`. Vous pouvez donner le nom que vous voulez à cette variable et il n'y a pas besoin de la définir.

```
<?php
//création du tableau
$tableau = array('Jean','Robert','Paul','Joe','Alain');
//boucle sur le tableau
foreach ($tableau as $val) {
   //affichage des valeurs du tableau concaténées
   //avec un saut de ligne
    echo $val.'<br />';
}
?>
```

Affiche :

Jean

Robert

Paul

Joe

Alain

Cette boucle a un autre avantage qui permet aussi d'afficher la clé du tableau :

```
<?php
//création du tableau
$tableau = array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe',
'H'=>'Alain');
```

```
//boucle sur le tableau
foreach ($tableau as $cle => $val) {
   //affichage des valeurs du tableau concaténées
   //avec un saut de ligne
    echo 'Clé : '.$cle.', valeur : '.$val.'<br />';
}
?>
```

Affiche :

Clé : A1, valeur : Jean

Clé : B4, valeur : Robert

Clé : 3, valeur : Paul

Clé : Toto, valeur : Joe

Clé : H, valeur : Alain

Cette fois, la boucle **`foreach`** s'écrit avec la variable `$cle` (que vous pouvez appeler autrement si vous le souhaitez) contenant la clé du tableau puis le signe => et enfin la variable `$val` contenant la valeur correspondant à la clé.

Vous pouvez aussi très bien utiliser la boucle **`foreach`** sans la clé :

```
<?php
//création du tableau
$tableau = array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe',
'H'=>'Alain');
//boucle sur le tableau
foreach ($tableau as $val) {
   //affichage des valeurs du tableau concaténées
   //avec un saut de ligne
    Echo 'valeur : '.$val.'<br />';
}
?>
```

Affiche :

valeur : Jean

valeur : Robert

valeur : Paul

valeur : Joe

valeur : Alain

c. La fonction print_r

Cette fonction est utilisée particulièrement par les développeurs pour afficher le contenu du tableau simplement mais sans pouvoir modifier la mise en forme.

Exemple :

```
<?php
//création du tableau
$tableau = array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe',
'H'=>'Alain');
print_r($tableau);
?>
```

Affiche :

Array ([A1] => Jean [B4] => Robert [3] => Paul [Toto] => Joe [H] => Alain)

5. Fonctions sur les tableaux

a. Longueur d'un tableau

La fonction qui permet de connaître le nombre d'éléments dans un tableau est **`count()`** ou **`sizeof()`**.

Par exemple :

```
<?php
//création du tableau
$tableau = array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe',
'H'=>'Alain');
$taille = count($tableau);
echo 'La taille du tableau est : '.$taille;
?>
```

Affiche :

La taille du tableau est : 5

Sa syntaxe est donc :

```
$nombre_d_elements = count($tableau);
```

avec `$nombre_d_elements` de type numérique

Cette fonction renvoie 0 si le tableau est vide.

b. Existence d'une valeur dans un tableau

La fonction qui permet de rechercher un élément dans un tableau est **`in_array()`**.

Par exemple :

```
<?php
//création du tableau
$tableau = array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe',
'H'=>'Alain');
if (in_array('Robert',$tableau)) {
   echo 'Robert est dans le tableau';
}
?>
```

Affiche :

Robert est dans le tableau

Sa syntaxe est donc :

```
$presence = in_array($valeur_recherchee,$tableau);
```

avec `$presence` de type booléen.

Cette fonction peut prendre un troisième paramètre en option qui est la vérification du type de la valeur trouvée par rapport à la valeur recherchée. Ce paramètre est une variable booléenne qui vaut false par défaut, c'est-à-dire qui ne tient pas compte du type.

Explication : vous avez la valeur '55' dans un tableau mais vous recherchez le chiffre 55. Ils ne sont pas du même type car '55' est de type string (chaîne de caractères) et 55 de type numérique.

Exemple :

```
<?php
//création du tableau
$tableau = array('10','33','55','78');
if (in_array(33,$tableau, true)) { // ajout de true pour tenir compte
                                   // du type
   echo '33 est dans le tableau';
} else {
   echo "33 n'est pas dans le tableau";
}
?>
```

Affiche :

33 n'est pas dans le tableau

En effet, la fonction recherche le nombre 33 dans le tableau mais il n'existe que la chaîne de caractères '33' dans le tableau.

Sa syntaxe avec le type optionnel est donc :

```
$presence = in_array($valeur_recherchee,$tableau,$type_equivalent);
```

avec `$presence` de type booléen et `$type_equivalent` de type booléen.

c. Existence d'une clé dans un tableau

La fonction qui permet de connaître l'existence d'une clé dans un tableau est **`array_key_exists()`**.

Par exemple :

```
<?php
//création du tableau
$tableau = array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe',
'H'=>'Alain');
if (array_key_exists(3,$tableau)) {
   echo 'La clé 3 est dans le tableau';
}
?>
```

Affiche :

La clé 3 est dans le tableau

Sa syntaxe est donc :

```
$presence = array_key_exists($clé_recherchée,$tableau);
```

avec `$presence` de type booléen.

Autre exemple :

```
<?php
//création du tableau
$tableau = array('Jean','Robert','Paul','Joe','Alain');
if (array_key_exists(4,$tableau)) {
   echo 'La clé 4 est dans le tableau, sa valeur est : '.$tableau[4];
}
?>
```

Affiche :

La clé 4 est dans le tableau, sa valeur est : Alain

d. Tri d'un tableau

Il existe plusieurs fonctions permettant de trier un tableau. Certaines trient dans l'ordre décroissant, d'autres suivant la clé et non la valeur, etc.

- **`sort()`** : trie les valeurs de la plus petite à la plus grande.

Par exemple :

```
<?php
//création du tableau
$tableau =
array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe','H'=>'Alain');
sort($tableau); //tri du tableau
foreach ($tableau as $cle=>$valeur) {
   echo 'Clé : '.$cle.', valeur : '.$valeur.'<br />';
}
?>
```

Affiche :

Clé : 0, valeur : Alain

Clé : 1, valeur : Jean

Clé : 2, valeur : Joe

Clé : 3, valeur : Paul

Clé : 4, valeur : Robert

La fonction sort() fait perdre la clé d'origine. Vous n'avez plus la clé 'H' pour 'Alain' mais 0. Les clés d'origine sont écrasées par un chiffre croissant commençant par 0.

- **`asort()`** : trie les valeurs de la plus petite à la plus grande avec préservation du couple clé/valeur.

Par exemple :

```
<?php
//création du tableau
$tableau =
array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe','H'=>'Alain');
asort($tableau); //tri du tableau avec préservation de la clé
foreach ($tableau as $cle=>$valeur) {
```

```
   echo 'Clé : '.$cle.', valeur : '.$valeur.'<br />';
}
?>
```

Affiche :

Clé : H, valeur : Alain

Clé : A1, valeur : Jean

Clé : Toto, valeur : Joe

Clé : 3, valeur : Paul

Clé : B4, valeur : Robert

Cette fois, les clés ont bien été préservées.

- **`rsort()`** : trie les valeurs de la plus grande à la plus petite.

Par exemple :

```
<?php
//création du tableau
$tableau =
array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe','H'=>'Alain');
rsort($tableau); //tri du tableau de la plus grande à la plus petite valeur
foreach ($tableau as $cle=>$valeur) {
   echo 'Clé : '.$cle.', valeur : '.$valeur.'<br />';
}
?>
```

Affiche :

Clé : 0, valeur : Robert

Clé : 1, valeur : Paul

Clé : 2, valeur : Joe

Clé : 3, valeur : Jean

Clé : 4, valeur : Alain

- **`arsort()`** : trie les valeurs de la plus grande à la plus petite avec préservation du couple clé/valeur.

Par exemple :

```
<?php
//création du tableau
$tableau =
```

```
array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe','H'=>'Alain');
arsort($tableau); //tri du tableau avec préservation de la clé
foreach ($tableau as $cle=>$valeur) {
   echo 'Clé : '.$cle.', valeur : '.$valeur.'<br />';
}
?>
```

Affiche :

Clé : B4, valeur : Robert

Clé : 3, valeur : Paul

Clé : Toto, valeur : Joe

Clé : A1, valeur : Jean

Clé : H, valeur : Alain

- **`ksort()`** : trie les clés du tableau de la plus petite à la plus grande avec préservation du couple clé/valeur.

Par exemple :

```
<?php
//création du tableau
$tableau =
array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe','H'=>'Alain');
ksort($tableau); //tri du tableau suivant la clé
foreach ($tableau as $cle=>$valeur) {
   echo 'Clé : '.$cle.', valeur : '.$valeur.'<br />';
}
?>
```

Affiche :

Clé : A1, valeur : Jean

Clé : B4, valeur : Robert

Clé : H, valeur : Alain

Clé : Toto, valeur : Joe

Clé : 3, valeur : Paul

- **`krsort()`** : trie les clés du tableau de la plus grande à la plus petite avec préservation du couple clé/valeur.

Par exemple :

```
<?php
```

```
//création du tableau
$tableau =
array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe','H'=>'Alain');
krsort($tableau); //tri du tableau descendant suivant la clé
foreach ($tableau as $cle=>$valeur) {
   echo 'Clé : '.$cle.', valeur : '.$valeur.'<br />';
}
?>
```

Affiche :

Clé : 3, valeur : Paul

Clé : Toto, valeur : Joe

Clé : H, valeur : Alain

Clé : B4, valeur : Robert

Clé : A1, valeur : Jean

e. Recherche dans un tableau

`array_search()` est une fonction équivalente à **`in_array()`**. Elle permet de rechercher un élément dans un tableau mais elle renvoie la clé de l'élément recherché.

Par exemple :

```
<?php
//création du tableau
$tableau = array('A1'=>'Jean','B4'=>'Robert',3=>'Paul','Toto'=>'Joe',
'H'=>'Alain');
 $cle_element = array_search('Robert',$tableau);
 echo "La clé de l'élément recherché est : ".$cle_element;

?>
```

Affiche :

La clé de l'élément recherché est : B4

Sa syntaxe est donc :

```
$cle = array_search($valeur_recherchee,$tableau);
```

Autre exemple :

```
<?php
//création du tableau
```

```
$tableau = array('Jean','Robert','Paul','Joe','Alain');
 $cle_element = array_search('Robert',$tableau);
 echo "La clé de l'élément recherché est : ".$cle_element;


?>
```

Affiche :

La clé de l'élément recherché est : 1

En effet, la clé de l'élément 'Robert' est 1 car la clé du tableau commence à 0.

f. Découper une chaîne dans un tableau

`explode()` permet de découper une chaîne dans un tableau en utilisant un séparateur.

Par exemple :

```
<?php
$ensemble = "1;2;3;4;5";
$tableau = explode(";",$ensemble);
echo "La première valeur du tableau est : ".$tableau[0]."<br />";
echo "La deuxième valeur du tableau est : ".$tableau[1];
?>
```

Affiche :

La première valeur du tableau est : 1

La deuxième valeur du tableau est : 2

Sa syntaxe est donc :

```
$tableau_decoupe = explode($separateur,$chaine_a_decouper);
```

Autre exemple :

```
<?php
$ensemble = "1-2-3-4-5";
$tableau = explode("-",$ensemble);
echo "Les valeurs du tableau sont : ";
foreach ($tableau as $valeur) {
   echo $valeur.";";
}
```

```
?>
```

Affiche :

Les valeurs du tableau sont : 1;2;3;4;5;

g. Regrouper les valeurs d'un tableau dans une chaîne

`implode()` permet de regrouper les valeurs d'un tableau dans une chaîne en utilisant un séparateur.

Par exemple :

```
<?php
$tableau = array("Jean","Robert","Paul");
$noms = implode(";",$tableau);
echo "Les noms sont : ".$noms;
?>
```

Affiche :

Les noms sont : Jean;Robert;Paul

Sa syntaxe est donc :

```
$chaine = implode($separateur,$tableau);
```

Autre exemple :

```
<?php
$tableau = array(5=>"Jean",3=>"Robert",2=>"Paul");
$noms = implode("-",$tableau);
echo "Les noms sont : ".$noms;
?>
```

Affiche :

Les noms sont : Jean-Robert-Paul

Vous voyez que les clés du tableau n'ont aucun effet sur la fonction **`implode()`**.

h. Découper une chaîne en morceaux de longueur fixe

`str_split()` permet de découper une chaîne en morceaux de longueur fixe dans un tableau en utilisant un paramètre indiquant la taille des morceaux de chaîne.

Par exemple :

```php
<?php
$ensemble="1;2;3;4;5";
$tableau=str_split($ensemble,2);
echo "Les éléments sont : ";
foreach ($tableau as $valeur) {
  echo $valeur." ";
}
?>
```

Affiche :

Les éléments sont : 1;2;3;4;5;

Sa syntaxe est donc :

```php
$tableau=str_split($chaine,$longueur);
```

Autre exemple :

```php
<?php
$ensemble="11=12=13=14=15=";
$tableau=str_split($ensemble,3);
echo "Les éléments sont : ";
foreach ($tableau as $valeur) {
  echo $valeur."<br/>";
}
?>
```

Affiche :

Les éléments sont : 11=
12=
13=
14=
15=

i. Ajouter des éléments à la fin du tableau

`array_push()` permet d'ajouter un ou plusieurs éléments à la fin du tableau.

Par exemple :

```php
<?php
$tableau = array("Jean","Robert","Paul");
array_push($tableau,"Joe","Alain");
echo $tableau[4];
```

```
?>
```

Affiche :

Alain

Sa syntaxe est donc :

```
array_push($tableau,$valeur1,$valeur2,...);
```

Autre exemple :

```
<?php
$tableau = array();
for ($i=0;$i<=10;$i++) {
   array_push($tableau,$i);
   echo $tableau[$i].";";
}

?>
```

Affiche :

0;1;2;3;4;5;6;7;8;9;10;

j. Suppression d'un élément à la fin du tableau

`array_pop()` permet de supprimer un élément à la fin du tableau et de retourner sa valeur.

Par exemple :

```
<?php
$tableau = array("Jean","Robert","Paul");
$nom = array_pop($tableau);
echo "Le nom supprimé est : ".$nom;
?>
```

Affiche :

Le nom supprimé est : Paul

Sa syntaxe est donc :

```
$valeur_supprimee=array_pop($tableau);
```

Autre exemple :

```php
<?php

$tableau = array("Jean","Robert","Paul");
echo "Avant la suppression, le nombre d'éléments dans le tableau
est : ".count($tableau)."<br />";
$nombre_elements = count($tableau);
for ($i=0;$i<$nombre_elements;$i++) {
   $nom = array_pop($tableau);
}
echo "Le dernier nom supprimé est : ".$nom."<br />";
echo "Après la suppression, le nombre d'éléments dans le tableau
est : ".count($tableau);

?>
```

Affiche :

Avant la suppression, le nombre d'éléments dans le tableau est : 3

Le dernier nom supprimé est : Jean

Après la suppression, le nombre d'éléments dans le tableau est : 0

k. Sélection aléatoire d'un élément du tableau

`array_rand()` permet de sélectionner de manière aléatoire un ou plusieurs éléments d'un tableau et de retourner les clés correspondantes.

Par exemple :

```php
<?php

$tableau = array("Jean","Robert","Paul");
$cle_aleatoire = array_rand($tableau);
echo "Le nom sélectionné au hasard est : ".$tableau[$cle_aleatoire];

?>
```

Affiche :

Le nom sélectionné au hasard est : Robert (ou Jean ou Paul)

Sa syntaxe est donc :

```
$cle_selectionne=array_rand($tableau, $nombre_selectionne);
```

Le paramètre `$nombre_selectionne` est optionnel, par défaut il vaut 1. Il correspond au nombre de valeurs à prendre au hasard. Voici un exemple l'utilisant :

```
<?php


$tableau = array("Jean","Robert","Paul");

$tableau_cle_aleatoire = array_rand($tableau,2);

echo "Le premier nom sélectionné au hasard est : "

tableau[$tableau_cle_aleatoire[0]]."<br />";

echo "Le deuxième nom sélectionné au hasard est : "

tableau[$tableau_cle_aleatoire[1]];


?>
```

En effet, `$tableau cle aleatoire` est un tableau car il contient les clés des éléments sélectionnés au hasard. Donc `$tableau_cle_aleatoire[0]` contient la clé du premier élément sélectionné au hasard donc sa valeur est obtenue par `$tableau[$tableau_cle_aleatoire[0]]`.

Le nombre sélectionné ne doit pas être supérieur ou égal au nombre d'éléments du tableau.

6. Tableaux à plusieurs dimensions

Un tableau à plusieurs dimensions est un tableau dans un tableau. Cela peut être très utile si vous voulez stocker des informations liées entre elles.

Par exemple, si vous souhaitez créer ce tableau :

Vous avez deux possibilités pour le créer.

- **Par l'intermédiaire d'un tableau temporaire :**

```
<?php
$tab_caracteristique_dupont = array("prénom" => "PAUL","profession" =>

"ministre","age" => 50);

$tab_caracteristique_durand = array("prénom" => "ROBERT","profession" =>

"agriculteur","age" => 45);

$tab_personne['DUPONT'] = $tab_caracteristique_dupont;

$tab_personne['DURAND'] = $tab_caracteristique_durand;


echo $tab_personne['DURAND']['profession'];


?>
```

Affiche :

agriculteur

La syntaxe pour accéder à la valeur d'un élément du tableau est :

```
$element = $tableau['clé1']['clé2']
```

avec clé1 représentant la clé du tableau général et clé2 représentant la clé du sous-tableau.

- **En stockant directement les valeurs dans le tableau général :**

```
<?php


$tab_personne['DUPONT']['prénom'] = "PAUL";
$tab_personne['DUPONT']['profession'] = "ministre";
$tab_personne['DUPONT']['age'] = 50;


$tab_personne['DURAND']['prénom'] = "ROBERT";
$tab_personne['DURAND']['profession'] = "agriculteur";
$tab_personne['DURAND']['age'] = 45;


echo $tab_personne['DUPONT']['prénom'];


?>
```

Affiche :

PAUL

La manière de créer le tableau n'a aucune incidence sur son utilisation. La syntaxe pour accéder à la valeur d'un élément est donc la même.

7. Exercices sur les tableaux

a. Énoncés

Exercice 1 (facile)

Créer un tableau contenant les chiffres de 1 à 10 et un autre tableau contenant les nombres de 11 à 20. Ensuite, créer un autre tableau contenant la somme des deux premiers tableaux et afficher ses valeurs. Il faut utiliser les boucles pour créer ces tableaux.

Exercice 2 (facile)

Créer un tableau de 10 valeurs au hasard entre 1 et 100. La fonction rand($min,$max); permet de tirer un nombre au hasard entre $min et $max. Trier ce tableau du plus petit au plus grand puis mettre toutes les valeurs dans une chaîne de caractères séparées par des ; et afficher la chaîne.

Exercice 3 (difficile)

Dans la section Tableau à plusieurs dimensions, vous avez vu le code pour créer ce tableau multidimensionnel :

Créer le code PHP permettant de générer ce tableau en HTML à l'aide des boucles.

Exercice 4 (facile)

Voici deux tableaux :

Le tableau1 est composé des éléments 6,25,35 et 61.

Le tableau2 est composé des éléments 12,24 et 46.

Écrire le code permettant de calculer une valeur représentative de ces deux tableaux notée S. La valeur S se calcule en multipliant chaque valeur du tableau1 par celle du tableau2 et en additionnant le tout.

Dans cet exemple, la valeur S sera égale à :

12*6+12*25+12*35+12*61+24*6+24*25+24*35+24*61+46*6+46*25+ 46*35+46*61

Il faut bien entendu utiliser les boucles pour réaliser cet exercice.

Exercice 5 (difficulté moyenne)

Afficher des bannières de manière aléatoire parmi trois possibles lorsqu'on arrive sur votre page PHP. Ces bannières ont pour caractéristiques une image, un lien et une description sur cette image. Ces caractéristiques sont stockées dans un tableau.

Par exemple :

```
$tabBannieres = array(
1 => array('http://www.votre_site.com','http://www.votre_site.com/
banniere.gif','Description 1'),
2 => array('http://www.votre_site.com2','http://www.votre_site2.com/
banniere.gif','Description 2'),
3 => array('http://www.votre_site3.com','http://www.votre_site3.com/
banniere.gif','Description 3'));
```

Traitement de chaînes de caractères

1. Les fonctions de manipulation de chaîne

Dans ce chapitre, il est question de toutes les fonctions PHP permettant de manipuler les chaînes de caractères.

a. strlen()

La fonction **strlen()** retourne la longueur d'une chaîne de caractères.

Par exemple :

```
<?php

$nom = "Robert";
$longueur = strlen($nom);
echo "La longueur de la chaîne est : ".$longueur;

?>
```

Affiche :

La longueur de la chaîne est : 6

Sa syntaxe est donc :

```
$longueur= strlen($chaine);
```

Autre exemple :

```
<?php

$nom = " Bonjour, Robert  ";
$longueur = strlen($nom);
echo "La longueur de la chaîne est : ".$longueur;

?>
```

Affiche :

La longueur de la chaîne est : 18

En effet, les espaces sont aussi comptés.

b. substr()

La fonction **substr()** retourne un morceau de la chaîne à partir d'une position et d'une certaine longueur donnée.

Par exemple :

```
<?php
```

```
$nom = "Robert";
$morceau = substr($nom,2,3);
echo "Le morceau de la chaîne est : ".$morceau;

?>
```

Affiche :

Le morceau de la chaîne est : ber

La position du début commence à 0.

Il n'est pas nécessaire d'indiquer la longueur. La fonction retourne alors les caractères jusqu'à la fin de la chaîne de caractères.

Sa syntaxe est donc :

```
$morceau_chaine = substr($chaine,$position_debut,longueur_chaine);
```

Autre exemple :

```
<?php

$nom = "Bonjour, je m'appelle Robert";
$morceau = substr($nom,22);
echo "Le morceau de la chaîne est : ".$morceau;

?>
```

Affiche :

Le morceau de la chaîne est : Robert

En effet, en n'indiquant pas le dernier paramètre, la fonction prend la chaîne de caractères jusqu'à sa fin.

c. strstr()

La fonction **strstr()** retourne un morceau de la chaîne à partir d'un caractère jusqu'à la fin de la chaîne.

Par exemple :

```
<?php
```

```
$email = "robert.dupont@france.fr";
$morceau = strstr($email,'@');
echo "Le morceau de la chaîne est : ".$morceau;

?>
```

Affiche :

Le morceau de la chaîne est : @france.fr

La fonction retourne la chaîne de caractères allant du caractère @ jusqu'à la fin de la chaîne.

Sa syntaxe est donc :

```
$morceau_chaine = strstr($chaine,$caractere_recherche);
```

La fonction retourne `false` si aucune chaîne de caractères n'est trouvée.

Autre exemple :

```
<?php

$email = "robert.dupont@france.fr";
$morceau = strstr($email,'.');
echo "Le morceau de la chaîne est : ".$morceau;

?>
```

Affiche :

Le morceau de la chaîne est : .dupont@france.fr

En effet, la fonction retourne un morceau de chaîne à partir du premier caractère trouvé.

d. str_replace()

La fonction **str_replace()** permet de remplacer un morceau d'une chaîne par une autre chaîne de caractères dans la chaîne de caractères principale.

Par exemple :

```
<?php

$email = "robert.dupont@france.fr";
$nouveau_nom = str_replace('france','belgique',$email);
echo "Le nouveau nom de la chaîne est : ".$nouveau_nom;
```

```
?>
```

Affiche :

Le nouveau nom de la chaîne est : robert.dupont@belgique.fr

Sa syntaxe est donc :

```
$nouvelle_chaine = str_replace($chaine_recherchee,
$chaine_qui_remplace,$chaine_principale);
```

Autre exemple, vous pouvez mettre un tableau à la place de la **$chaine_recherchee** :

```
<?php

$tableau_chaine_recherchee =
array("a","e","i","o","u","y","A","E","I","O","U","Y");
$email = "robert.dupont@france.fr";
$nouveau_nom = str_replace($tableau_chaine_recherchee,'',$email);
echo "Le nouveau nom de la chaîne en enlevant toutes les voyelles
est : " nouveau_nom;

?>
```

Affiche :

Le nouveau nom de la chaîne en enlevant toutes les voyelles est : r brt.dpnt@frnc.fr

En effet, dans cet exemple, la fonction **str_replace** remplace toutes les voyelles du tableau par une chaîne vide.

e. trim()

La fonction **trim()** permet de supprimer les espaces en début et en fin de chaîne.

Par exemple :

```
<?php

$email = "  robert.dupont@france.fr  ";
$longueur_nom = strlen($email);//longueur de la chaîne $nom:27
$nouveau_nom = trim($email);//suppression des espaces

$longueur_nouveau_nom = strlen($nouveau_nom);
```

```
//longueur de la chaîne $nom:23

echo "Le nouveau nom de la chaîne est : ".$nouveau_nom." avec
". $longueur_nouveau_nom." caractères";

?>
```

Affiche :

Le nouveau nom de la chaîne est : robert.dupont@france.fr avec 23 caractères

Sa syntaxe est donc :

```
$nouvelle_chaine = trim($chaine);
```

Cette fonction supprime les espaces, les tabulations et les sauts de ligne.

f. strtolower()

La fonction **strtolower()** permet de transformer une chaîne en minuscules.

Par exemple :

```
<?php

$nom = "ROBERT";
$nom = strtolower($nom);
echo "Le nom de la chaîne en minuscules est : ".$nom;

?>
```

Affiche :

Le nom de la chaîne en minuscules est : robert

Sa syntaxe est donc :

```
$chaine_minuscules = strtolower($chaine);
```

g. strtoupper()

La fonction **strtoupper()** permet de transformer une chaîne en majuscules.

Par exemple :

```
<?php
```

```
$nom = "Jean";
$nom = strtoupper($nom);
echo "Le nom de la chaîne en majuscules est : ".$nom;

?>
```

Affiche :

Le nom de la chaîne en majuscules est : JEAN

Sa syntaxe est donc :

```
$chaine_majuscules = strtoupper($chaine);
```

La fonction **ucfirst()** met le premier caractère en majuscule. La fonction **ucwords()** met la première lettre de chaque mot en majuscule.

h. strpos()

La fonction **strpos()** retourne la position de la première occurrence dans une chaîne de caractères.

Par exemple :

```
<?php

$email = "jean.dupont@france.fr";
$position = strpos($email,'@');
echo "La position de @ est : ".$position;

?>
```

Affiche :

La position de @ est : 11

Sa syntaxe est donc :

```
$position = strpos($chaine,$occurrence_recherchee);
```

La position commence à 0. Ainsi la position de j dans la chaîne *$nom* est 0.

Il existe aussi deux autres fonctions semblables qui sont :

- **strrpos()** : retourne la position de la dernière occurrence dans une chaîne de caractères.

- **stripos()** : retourne la position de la première occurrence dans une chaîne de caractères sans tenir compte de la casse.

Autre exemple :

```
<?php


$email = "jean.dupont@france.fr";
$position = strrpos($email,'f');
echo "La dernière position de la lettre f est : ".$position;


?>
```

Affiche :

La dernière position de la lettre f est : 19

i. str_word_count()

La fonction **str_word_count()** retourne le nombre de mots contenus dans la chaîne de caractères.

Par exemple :

```
<?php


$phrase = "Bonjour, il fait beau";
$nombre = str_word_count($phrase);
echo "Le nombre de mots dans la chaîne est : ".$nombre;


?>
```

Affiche :

Le nombre de mots dans la chaîne est : 4

Sa syntaxe est donc :

```
$position = str_word_count($chaine);
```

Cette fonction peut prendre un paramètre optionnel qui est le format. S'il vaut 0, la fonction retourne le nombre de mots comme précédemment. S'il vaut 1, la fonction retourne un tableau contenant les mots de la chaîne de caractères.

Par exemple :

```
<?php


```

```
$phrase = "Bonjour, il fait beau";
$tableau = str_word_count($phrase,1);
print_r($tableau);


?>
```

Affiche :

Array ([0] => Bonjour [1] => il [2] => fait [3] => beau)

Si ce paramètre vaut 2, la fonction retourne un tableau contenant les mots de la chaîne de caractères et la position de la première lettre du mot en clé.

Par exemple :

```
<?php


$phrase = "Bonjour, il fait beau";
$tableau = str_word_count($phrase,2);
print_r($tableau);


?>
```

Affiche :

Array ([0] => Bonjour [9] => il [12] => fait [17] => beau)

Dans cette fonction, la notion de mot dépend de la configuration de localisation. Ainsi, la virgule n'est pas considérée comme un mot.

j. str_pad()

La fonction **str_pad()** permet de compléter une chaîne jusqu'à une taille donnée.

Par exemple :

```
<?php
  $chaine_debut = "Bonjour";
  echo str_pad($chaine_debut, 10, '!');
?>
```

Affiche :

Bonjour!!!

En effet, la fonction prend la chaîne d'origine "Bonjour" puis complète avec la chaîne "!" jusqu'à 10 caractères.

Sa syntaxe est donc :

```
str_pad($chaîne_origine, $nombre_caractere_total,
$chaine_pour_completer);
```

Pour afficher 10 espaces en HTML, le code est :

```
<?php
  echo str_pad('', 60, ' ');
?>
```

2. Les expressions régulières

Les expressions régulières permettent d'effectuer des recherches ou des remplacements très complexes dans des chaînes de caractères.

Par exemple, si vous voulez savoir si un e-mail contient bien le caractère **@** et le caractère **.**, ou vous voulez changer le format d'une date de l'anglais en français, l'utilisation des expressions régulières le permet en une seule ligne !

Dans la suite de ce support, nous utiliserons PCRE (*Perl-Compatible Regular Expressions*) qui utilise les fonctions d'expressions régulières les plus rapides. Il existe aussi POSIX (*Portable Operating System Interface*) dont les fonctions commencent par `ereg` mais elles sont aujourd'hui obsolètes. L'encodage est positionné sur ANSI dans Notepad++ pour que les exemples suivants fonctionnent correctement.

La fonction **preg_match()** retourne vrai si la valeur recherchée est dans la chaîne de caractères.

Par exemple :

```
<?php


if (preg_match("/web/","Le webdesigner conçoit un site web.")) {
    echo "La chaîne web a été trouvée.";
}
else {
    echo "La chaîne web n'a pas été trouvée.";
}


?>
```

Affiche :

La chaîne web a été trouvée.

Sa syntaxe est donc :

```
$existe = preg_match ($pattern,$chaine);
```

avec `$existe` de type booléen.

`$pattern` est une chaîne indiquant à la fonction `preg_match()` comment effectuer la recherche. Cette chaîne commence et finit par un délimiteur qui est souvent par convention le symbole **/**. Il est possible de voir aussi le symbole **#**.

Dans l'exemple ci-dessus, la fonction `preg_match()` recherche si la chaîne "Le webdesigner conçoit un site web." contient la chaîne "web".

Par défaut, cette fonction tient compte de la casse.

Par exemple :

```
<?php


if (preg_match("/WEB/","Le webdesigner conçoit un site web.")) {
    echo "La chaîne web a été trouvée.";
}
else {
    echo "La chaîne web n'a pas été trouvée.";
}


?>
```

Affiche :

La chaîne web n'a pas été trouvée.

Car le mot WEB est différent de web.

Il existe aussi la fonction **preg_replace()** permettant de remplacer le contenu recherché par un autre contenu passé en paramètre. Cette fonction retourne la chaîne transformée.

Sa syntaxe est donc :

```
$chaîne_transformée =
preg_replace($pattern,$chaine_de_remplacement,$chaine originale);
```

a. Insensibilité à la casse

Si vous souhaitez ne pas tenir de compte de la casse, il suffit d'ajouter **i** après le dernier **/**.

Par exemple :

```
<?php
```

```
if (preg_match("/WEB/i","Le webdesigner conçoit un site web."))
{
    echo "La chaîne web a été trouvée.";
}
else {
    echo "La chaîne web n'a pas été trouvée.";
}

?>
```

Affiche :

La chaîne web a été trouvée.

Vous voyez que toute la recherche s'effectue grâce au motif recherché (pattern). Il est possible de lui ajouter encore beaucoup d'options. Nous allons voir les plus utilisées ci-après.

b. Recherche d'un mot et non d'une chaîne

Jusqu'à présent, la recherche de chaîne de caractères s'effectue dans une phrase. Par exemple, si vous cherchez la chaîne "designer" :

```
<?php

if (preg_match("/designer/","Le webdesigner conçoit un site web."))
{
    echo "La chaîne designer a été trouvée.";
}
else {
    echo "La chaîne designer n'a pas été trouvée.";
}

?>
```

Affiche :

La chaîne designer a été trouvée.

Mais si vous voulez rechercher uniquement les mots commençant par "designer", il faut ajouter **\b**devant le mot dans le motif recherché.

Par exemple :

```php
<?php

if (preg_match("/\bdesigner/","Le webdesigner conçoit un site web."))
{
    echo "Le mot designer a été trouvé.";
}
else {
    echo "Le mot designer n’a pas été trouvé.";
}

?>
```

Affiche :

Le mot designer n’a pas été trouvé.

En effet, designer fait partie du mot webdesigner mais n’est pas un mot en lui-même.

En revanche si vous écrivez :

```php
<?php

if (preg_match("/\bweb/","Le webdesigner conçoit un site web."))
{
    echo "Le mot web a été trouvé.";
}
else {
    echo "Le mot web n’a pas été trouvé.";
}

?>
```

Affiche :

Le mot web a été trouvé.

Si vous voulez trouver un mot sans rien autour, il suffit de l’entourer par \b.

Par exemple :

```php
<?php

if (preg_match("/\bsite\b/","Le webdesigner conçoit un site web."))
```

```
{
   echo "Le mot site a été trouvé.";
}
else {
   echo "Le mot site n'a pas été trouvé.";
}

?>
```

Affiche :

Le mot site a été trouvé.

c. Recherches de chaînes avec OU

Ce symbole se note **|**, il permet de rechercher une chaîne ou une autre.

Par exemple :

```
<?php

if (preg_match("/webdesigner|graphiste/","Le webdesigner conçoit
un site web."))
{
    echo "La chaîne webdesigner ou la chaîne graphiste a été trouvée.";
}
else {
   echo "La chaîne webdesigner ou la chaîne graphiste n'a pas été
trouvée."; }

?>
```

Affiche :

La chaîne webdesigner ou la chaîne graphiste a été trouvée.

En effet, la chaîne "graphiste" n'existe pas mais la chaîne "webdesigner" si.

d. Début de chaîne

Ce symbole se note ^, il permet de rechercher une chaîne commençant par un mot.

Par exemple :

```
<?php
```

```
if (preg_match("/^Le/","Le webdesigner conçoit un site web.")) {
    echo "La chaîne commence par le mot 'Le'.";
}
else {
   echo "La chaîne ne commence pas par le mot 'Le'.";
}

?>
```

Affiche :

La chaîne commence par le mot 'Le'.

e. Fin de chaîne

Ce symbole se note **$**, il permet de rechercher une chaîne finissant par un mot.

Par exemple :

```
<?php

if (preg_match("/web.$/","Le webdesigner conçoit un site web.")) {
    echo "La chaîne finit par le mot 'web.'.";
}
else {
   echo "La chaîne ne finit pas par le mot 'web.'.";
}

?>
```

Affiche :

La chaîne finit par le mot 'web.'.

f. Un caractère dans une classe

Une classe permet de définir un ensemble de caractères pouvant être contenu dans une chaîne. Sa syntaxe se note **[...]** avec les caractères à l'intérieur des crochets.

Par exemple :

```
<?php

if (preg_match("/p[au]r/","par ailleurs ce site web est beau.")) {
    echo "La chaîne contient la chaîne par ou pur.";
```

```
}
else {
   echo "La chaîne ne contient pas la chaîne par ou pur.";
}


?>
```

Affiche :

La chaîne contient la chaîne par ou pur.

En effet, l'expression régulière est vraie si elle contient la chaîne "par" ou la chaîne "pur".

Si vous écrivez `preg_match("/p[aieu]r/",`*`chaîne de caractères`*`)`, l'expression régulière sera vraie si la chaîne contient le mot "par" ou le mot "pir" ou le mot "per" ou le mot "pur".

Par exemple :

```
<?php


if (preg_match("/p[aou]r$/","par ailleurs ce site web est beau."))
{
    echo "La chaîne finit par le mot par ou por ou pur.";
}
else {
    echo "La chaîne ne finit pas par le mot par ou por ou pur.";
}


?>
```

Affiche :

La chaîne ne finit pas par le mot par ou por ou pur.

En effet, un caractère $ a été ajouté à la fin de l'expression régulière, ce qui signifie que la chaîne devait finir par les mots par, por ou pur.

g. Une plage de caractères dans une classe

Il est très fastidieux de noter toutes les lettres de l'alphabet ou tous les chiffres de 0 à 9 dans une classe. Heureusement il existe le symbole - (tiret) qui permet de définir une plage de caractères ou de chiffres.

Par exemple :

```php
<?php


if (preg_match("/p[a-z]r/","Dans le port d'Amsterdam.")) {
    echo "La chaîne contient la chaîne por.";
}
else {
    echo "La chaîne ne contient pas la chaîne por.";
}


?>
```

Affiche :

La chaîne contient la chaîne por.

En effet, l'expression régulière est vraie si la chaîne contient une autre chaîne commençant par p puis n'importe quelle lettre de l'alphabet puis un r. Donc le mot "port" rentre bien dans ce cadre-là.

Autre exemple :

```php
<?php


if (preg_match("/ [0-9]/","Ce fruit coûte 10 euros.")) {
    echo "La chaîne contient un espace puis un chiffre entre 0 et 9.";
}
else {
   echo "La chaîne ne contient pas d'espace puis un chiffre
entre 0 et 9.";
}


?>
```

Affiche :

La chaîne contient un espace puis un chiffre entre 0 et 9.

En effet, l'expression régulière est vraie si la chaîne contient un espace suivi d'un chiffre entre 0 et 9. Comme la chaîne contient le nombre 10 précédé d'un espace, l'expression régulière est vérifiée.

h. La non-présence d'une plage de caractères dans une classe

Si vous ne voulez pas les caractères présents dans une classe, il faut ajouter le symbole ^ au début dans la classe. Ce symbole est le même que lorsque vous voulez indiquer un mot en début de chaîne.

Par exemple :

```
<?php

if (preg_match("/P[^a-zA-Z]/","L'EPE(E) ne brise pas la pensée.")) {
    echo "La chaîne contient la lettre 'P' suivie d'un caractère
non alphabétique.";
}
else {
    echo "La chaîne ne contient pas la lettre 'P' suivie d'un caractère
non alphabétique.";
}

?>
```

Affiche :

La chaîne ne contient pas la lettre 'P' suivie d'un caractère non alphabétique.

En effet, le mot EPE(E) contient la lettre P mais il est suivi d'une lettre entre a et z ou entre A et Z, donc l'expression régulière retourne faux.

i. Les quantificateurs

Les quantificateurs servent à définir combien de fois peut être répété un caractère ou une classe. Les trois principaux sont :

Le symbole **?** indique qu'il y a une ou aucune occurrence du caractère ou de la classe précédente.

Par exemple :

```
<?php

if (preg_match("/da?m/","Dans le port d'Amsterdam.")) {
    echo "La chaîne contient dam ou dm.";
}
else {
   echo "La chaîne ne contient pas dam ou dm.";
}

?>
```

Affiche :

La chaîne contient dam ou dm.

En effet, l'expression régulière recherche dans la chaîne "Dans le port d'Amsterdam" la lettre d éventuellement suivie de la lettre a et suivie de la lettre m.

Le symbole **+** indique une ou plusieurs occurrences du caractère ou de la classe précédente.

Par exemple :

```
<?php


if (preg_match("/da+m/","Dans le port d'Amsterdam.")) {
    echo "La chaîne contient dam ou daam ou daaam...";
}
else {
   echo "La chaîne ne contient pas dam ou daam ou daaam...";
}


?>
```

Affiche :

La chaîne contient dam ou daam ou daaam…

En effet, l'expression régulière recherche dans la chaîne "Dans le port d'Amsterdam" la lettre d suivie d'une ou plusieurs fois de la lettre a et suivie de la lettre m.

Le symbole ***** indique zéro, une ou plusieurs occurrences du caractère ou de la classe précédente.

Par exemple :

```
<?php


if (preg_match("/da*m/","Dans le port d'Amsterdam.")) {
    echo "La chaîne contient dm ou dam ou daam ou daaam...";
}
else {
    echo "La chaîne ne contient pas dm ou dam ou daam ou daaam...";
}


?>
```

Affiche :

La chaîne contient dm ou dam ou daam ou daaam…

En effet, l'expression régulière recherche dans la chaîne "Dans le port d'Amsterdam" la lettre d éventuellement suivie d'une ou plusieurs fois de la lettre a et suivie de la lettre m.

j. Les intervalles de reconnaissance

Ils servent à définir avec précision combien de fois un caractère ou un groupe de caractères peut être répété. Cet intervalle se note grâce aux accolades **{}**.

- Si vous voulez que la lettre "a" soit répétée exactement deux fois, l'expression régulière est : a{2}
- Si vous voulez que la lettre "a" soit répétée au minimum deux fois, l'expression régulière est : a{2,}
- Si vous voulez que la lettre "a" soit répétée entre deux et cinq fois, l'expression régulière est : a{2,5}

Par exemple :

```
<?php


if (preg_match("/1{1,}/","N° de téléphone:0600112233.")) {
    echo "Il y a au moins une fois le chiffre 1 dans votre N° de
téléphone.";
}
else {
   echo "Il n'y pas de chiffre 1 dans votre N° de téléphone.";
}


?>
```

Affiche :

Il y a au moins une fois le chiffre 1 dans votre N° de téléphone.

Si dans votre expression régulière vous mettez une chaîne avec un ?, par exemple si vous voulez rechercher la chaîne "Qui ?", il ne faut pas que le ? soit interprété comme un quantificateur indiquant 0 ou 1 occurrence du caractère précédent. Pour éviter cela, il faut utiliser le symbole ****(antislash) qui permet d'échapper le symbole suivant le ****, c'est-à-dire de ne pas l'interpréter comme un symbole mais comme un caractère.

Par exemple :

```
<?php


if (preg_match("/\?/","Qui est là ?")) {
    echo "Il y a le caractère ? dans votre phrase.";
```

```
}
else {
    echo "Il n'y a pas le caractère ? dans votre phrase.";
}

?>
```

Affiche :

Il y a le caractère ? dans votre phrase.

Attention : Les symboles ?, +, ^, *, $ sont vraiment interprétés comme des caractères et non comme des symboles d'expressions régulières lorsqu'ils sont à l'intérieur d'une classe […].

Vous constatez qu'il devient très complexe d'écrire une expression régulière. Heureusement, vous pouvez trouver facilement sur Internet la plupart des expressions régulières les plus utilisées.

Ci-dessous vous trouverez l'expression régulière pour vérifier si une adresse e-mail est valide :

```
<?php

$email = "jean.dupont@france.fr";
if (preg_match("/^[a-z0-9._-]+@[a-z0-9._-]{2,}\.[a-z]{2,4}$/",
$email)) {
    echo "L'adresse e-mail est valide.";
}
else {
    echo "L'adresse e-mail n'est pas valide.";
}

?>
```

Affiche :

L'adresse e-mail est valide.

Les opérateurs

1. Les opérateurs de chaîne

a. La concaténation

Vous avez eu l'occasion de le voir dans le chapitre Les bases du langage PHP : la concaténation se note avec **.** (point) ou **,** (virgule). Elle permet d'attacher deux chaînes de caractères.

Par exemple :

```
<?php


$prenom = "Jean";

$nom = "Dupont";

echo $prenom." ".$nom; //concaténation du prénom + un espace +

le nom


?>
```

Affiche :

Jean Dupont

b. L'assignation

Vous l'utilisez déjà depuis le début dans tous les exemples.

L'assignation se note avec le signe **=**. Elle permet d'affecter une valeur à une variable.

Vous pouvez aussi utiliser la combinaison **.=** qui permet de concaténer une chaîne à une variable et de l'affecter à cette variable.

Par exemple :

```
<?php


$prenom = "Jean";

$nom = "Dupont";

$prenom .= " "; //concaténation du prénom + un espace

$prenom .= $nom; //concaténation du prénom + le nom

echo $prenom;


?>
```

est équivalent à :

```
<?php


$prenom = "Jean";

$nom = "Dupont";

$prenom = $prenom." "; //concaténation du prénom + un espace

$prenom = $prenom.$nom; //concaténation du prénom + le nom

echo $prenom;
```

```
?>
```

Affiche :

Jean Dupont

2. Les opérateurs arithmétiques

a. L'addition

L'opérateur se note **+**.

Par exemple :

```
<?php

$nombre = 11;
$resultat = $nombre + 5;
echo $resultat;

?>
```

Affiche :

b. La soustraction

L'opérateur se note **-**.

Par exemple :

```
<?php

$nombre = 11;
$resultat = $nombre - 5;
echo $resultat;

?>
```

Affiche :

c. La multiplication

L'opérateur se note *****.

Par exemple :

```
<?php

$nombre = 11;
$resultat = $nombre * 5;
echo $resultat;

?>
```

Affiche :

55

d. La division

L'opérateur se note **/**.

Par exemple :

```
<?php

$nombre = 10;
$resultat = $nombre / 5;
echo $resultat;

?>
```

Affiche :

2

e. Le modulo

C'est le reste de la division du dividende par le diviseur. L'opérateur se note **%**.

Par exemple :

```
<?php

$nombre = 11;
$resultat = $nombre % 5;
echo $resultat;
```

```
?>
```

Affiche :

1

En effet, 11 est égal à 10+1. 10 est divisible par 5 et il reste 1.

Cet opérateur est très pratique pour savoir si un nombre est divisible par un autre.

Par exemple :

```
<?php

$nombre = 11;
$reste = $nombre % 5;
if ($reste == 0) {
   echo "Le nombre ".$nombre." est divisible par 5";
}
else {
   echo "Le nombre ".$nombre." n'est pas divisible par 5";
}

?>
```

Affiche :

Le nombre 11 n'est pas divisible par 5

f. L'incrémentation

L'opérateur se note **++**. L'ordre dans lequel on place l'opérateur a une importance. ++`$nombre`incrémente `$nombre` puis retourne `$nombre` alors que `$nombre`++ retourne `$nombre` puis l'incrémente.

Par exemple :

```
<?php

$nombre = 10;
echo ++$nombre; //incrémente puis affiche
echo ";";
echo $nombre;
```

```
?>
```

Affiche :

11;11

Autre exemple :

```
<?php

$nombre = 10;
echo $nombre++; //affiche puis incrémente
echo ";";
echo $nombre;

?>
```

Affiche :

10;11

g. La décrémentation

L'opérateur se note **--**.

Par exemple :

```
<?php

$nombre = 10;
echo --$nombre; //décrémente puis affiche
echo ";";
echo $nombre;

?>
```

Affiche :

9;9

Autre exemple :

```
<?php

$nombre = 10;
```

```
echo $nombre--; //affiche puis décrémente
echo ";";
echo $nombre;


?>
```

Affiche :

```
10;9
```

h. L'exponentiation

L'opérateur permettant de mettre un nombre en exponentiel est **.

Ainsi, 3 exposant 4 s'écrit : 3 ** 4

Par exemple :

```
<?php
echo 3 ** 4;
?>
```

Affiche :

```
81
```

Il existe également l'opérateur d'assignation équivalent **=.

Par exemple :

```
<?php
$nombre = 3;
$nombre **= 4;
echo "3^4 = ".$nombre;
?>
```

Affiche :

```
3^4 = 81
```

3. Les opérateurs de comparaison

a. L'égalité

L'opérateur se note **==**.

Par exemple :

```php
<?php


$nombre_1 = 11;

$nombre_2 = 11.0;

if ($nombre_1 == $nombre_2) {

   echo "Les deux nombres sont identiques";

}

else {

   echo "Les deux nombres ne sont pas identiques";

}


?>
```

Affiche :

Les deux nombres sont identiques

En effet, 11 et 11.0 ne sont pas du même type mais leurs valeurs sont égales.

En revanche, l'opérateur **===** teste la valeur et le type. Donc 11 de type int est différent de 11.0 qui est de type float.

Exemple :

```php
<?php


$nombre_1 = 11;

$nombre_2 = 11.0;

if ($nombre_1 === $nombre_2) {

   echo "Les deux nombres sont identiques";

}

else {

   echo "Les deux nombres ne sont pas identiques";

}


?>
```

Affiche :

Les deux nombres ne sont pas identiques

b. La différence

L'opérateur se note **!=**.

Par exemple :

```
<?php

$nombre = 11;
$chaine = "11";
if ($nombre != $chaine) {
   echo "Le nombre et la chaîne sont différents";
}
else {
   echo "Le nombre et la chaîne ne sont pas différents";
}

?>
```

Affiche :

Le nombre et la chaîne ne sont pas différents

En revanche, **!==** teste la valeur et le type des deux variables, donc :

```
<?php

$nombre = 11;
$chaine = "11";
if ($nombre !== $chaine) {
   echo "Le nombre et la chaîne sont différents";
}
else {
   echo "Le nombre et la chaîne ne sont pas différents";
}

?>
```

Affiche :

Le nombre et la chaîne sont différents

c. La comparaison

L'opérateur "inférieur à" se note **<**.

Par exemple :

```php
<?php

$nombre_1 = 11;
$nombre_2 = 12;
if ($nombre_1 < $nombre_2) {
   echo "Le nombre ".$nombre_1." est strictement inférieur au nombre ".$nombre_2;
}
else {
   echo "Le nombre ".$nombre_1." n'est pas strictement inférieur au nombre "
.$nombre_2;
}

?>
```

Affiche :

Le nombre 11 est strictement inférieur au nombre 12

L'opérateur "inférieur ou égal à" se note **<=**.

De la même façon, l'opérateur "supérieur à" se note **>**.

Par exemple :

```php
<?php

$nombre_1 = 11;
$nombre_2 = 12;
if ($nombre_1 > $nombre_2) {
   echo "Le nombre ".$nombre_1" est strictement supérieur au nombre ".$nombre_2;
}
else {
   echo "Le nombre ".$nombre_1" n'est pas strictement supérieur au nombre "
.$nombre_2";
}

?>
```

Affiche :

Le nombre 11 n'est pas strictement supérieur au nombre 12

L'opérateur "supérieur ou égal à" se note **>=**.

d. L'opérateur à trois issues

Depuis PHP 7, il existe un nouvel opérateur **<=>**, appelé aussi vaisseau spatial. Avec les opérateurs de comparaison traditionnels, le résultat est binaire, mais avec l'opérateur à trois issues, le résultat peut être -1, 0 ou 1.

Par exemple :

```
<?php


echo 2 <=> 2; // 0
echo " ";
echo 2 <=> 3; // -1
echo " ";
echo 3 <=> 2; // 1


?>
```

Affiche :

0 -1 1

4. L'opérateur ternaire

L'opérateur se note **?**. Il est équivalent à un if else mais sur une seule ligne. Sa syntaxe est :

```
condition?expression1:expression2
```

Si la condition est vraie, l'expression1 est utilisée, sinon c'est l'expression2.

Par exemple :

```
<?php


$nombre_1 = 11;
$nombre_2 = 12;
echo ($nombre_1 == $nombre_2)?"Les deux nombres sont identiques":
"Les deux nombres ne sont pas identiques";


?>
```

Affiche :

Les deux nombres ne sont pas identiques

En effet, les valeurs de `$nombre_1` et `$nombre_2` sont différentes donc la condition (`$nombre_1 == $nombre_2`) est fausse donc l'expression2 est affichée, c'est-à-dire "Les deux nombres ne sont pas identiques".

Depuis PHP 7, il existe un nouvel opérateur ternaire qui est **??**. Il permet de vérifier l'existence d'une variable avant son contenu et ainsi d'éviter l'affichage d'une erreur.

Par exemple :

```
<?php
//$valeur n'est pas initialisé
$valeur = $valeur ?? 'bonjour';
echo $valeur;



?>
```

Affiche :

bonjour

5. Les opérateurs logiques

a. ET

L'opérateur se note **&&** ou **and**.

Par exemple :

```
<?php


$age = 10;
$nom = "Jean";
if ($age == 10 && $nom == "Jean") {
   echo "ok";
}
else {
   echo "pas ok";
}


?>
```

Affiche :

ok

En effet, pour que la condition du if soit vérifiée, il faut que `$age` soit égal à 10 ET que `$nom` soit égal à "Jean". Comme c'est le cas, ok est affiché.

b. OU

L'opérateur se note **||** ou **or**.

Par exemple :

```
<?php

$age = 10;
$nom = "Robert";
if ($age == 10 || $nom == "Jean") {
   echo "ok";
}
else {
   echo "pas ok";
}

?>
```

Affiche :

ok

En effet, pour que la condition du `if` soit vérifiée, il faut que `$age` soit égal à 10 OU que `$nom` soit égal à "Jean". Comme c'est le cas, ok est affiché.

Autre exemple :

```
<?php

$age = 11;
$nom = "Robert";
if ($age == 10 && $nom == "Jean") {
   echo "ok";
}
else {
   echo "pas ok";
}

?>
```

Affiche :

pas ok

En effet, il n'y a ni `$nom` égal à "Jean", ni `$age` égal à 11, donc la condition du `if` est fausse donc c'est le code du bloc `else` (sinon) qui s'exécute.

Dans le cas où les opérateurs || et && se suivent, l'opérateur && est prioritaire devant l'opérateur ||.

Autre exemple :

```
<?php


$age = 11;
$nom = "Robert";
if ($nom == "Robert" || $age == 11 && $age == 10) {
   echo "ok";
}
else {
   echo "pas ok";
}


?>
```

est équivalent à :

```
<?php


$age = 11;
$nom = "Robert";
if ($nom == "Robert" || ($age == 11 && $age == 10)) {
   echo "ok";
}
else {
   echo "pas ok";
}


?>
```

Affiche :

ok

Mais si vous placez les parenthèses comme ceci :

```
<?php


$age = 11;

$nom = "Robert";

if (($nom == "Robert" || $age == 11) && $age == 10) {

   echo "ok";

}

else {

   echo "pas ok";

}


?>
```

Cela affiche :

pas ok

Donc suivant où vous placez les parenthèses, le sens de la condition peut être complètement différent.

Les fonctions

1. Création

Les fonctions permettent de réutiliser du code PHP plusieurs fois. Par exemple, si vous avez une page web avec le prix HT de différents produits, vous allez créer une fonction qui vous calcule le prix TTC de chaque produit. Cela évite d'écrire le calcul en PHP sur chaque ligne. En développement, il faut essayer de factoriser (c'est-à-dire de regrouper) le code. Ainsi, sa maintenance est plus facile. En effet, plus vous écrivez du code, plus vous risquez de faire des erreurs.

La syntaxe pour créer une fonction est :

```
function nom_de_la_fonction ($paramètre) { }
```

Il ne faut jamais mettre d'espaces ni de caractères spéciaux dans le nom d'une fonction.

Le paramètre est aussi appelé argument.

Essayez de nommer la fonction avec des mots expliquant ce que fait la fonction, séparés par le symbole _ ou avec des majuscules : nom_de_la_fonction ou NomDeLaFonction.

Par exemple :

```
<?php
```

```
function calcul_ttc($prix_ht) {
   return $prix_ht * 1.196;
}

?>
```

Cette fonction calcule le prix TTC à partir du prix HT passé en paramètre et retourne le résultat par l'intermédiaire du mot-clé **return**. Une fonction n'est pas obligée de retourner un résultat, elle peut juste servir à afficher un message par exemple.

Pour appeler cette fonction, il suffit d'écrire par exemple :

```
<?php
calcul_ttc(12);
?>
```

Soit au final :

```
<?php

function calcul_ttc($prix_ht) {
   return $prix_ht * 1.2;
}
$prix_ttc = calcul_ttc(12);
echo "12 euros HT correspondent à ".$prix_ttc." euros TTC";

?>
```

Affiche :

12 euros HT correspondent à 14.4 euros TTC

En effet, le nombre 12 est passé en paramètre de la fonction donc `$prix_ht` devient égal à 12, puis la fonction retourne 12*1.2 (14.4) dans la variable `$prix_ttc`.

Il est possible de créer des fonctions qui prennent plusieurs paramètres ou bien aucun paramètre.

Par exemple, voici une fonction qui prend plusieurs paramètres et retourne une valeur :

```
<?php

function assemble_mot($mot1,$mot2,$mot3) {
    $retour = $mot1." ".$mot2." ".$mot3;
```

```
    Return $retour;
}
echo assemble_mot("Bonjour","Jean","Dupont");

?>
```

Affiche :

Bonjour Jean Dupont

Autre exemple, une fonction ne prenant et ne retournant aucune valeur :

```
<?php

function affiche_bonjour() {
    echo "Bonjour";
}
affiche_bonjour();

?>
```

Affiche :

Bonjour

Depuis PHP 5.6, il est possible de donner des valeurs par défaut aux paramètres des fonctions. Ainsi, si aucune valeur n'est donnée pour un paramètre lors de l'appel de la fonction, il prendra sa valeur par défaut.

Par exemple :

```
<?php

function assemble_mot($mot1,$mot2,$mot3="Dupont") {
    $retour = $mot1." ".$mot2." ".$mot3;
    Return $retour;
}
echo assemble_mot("Bonjour","Jean");

?>
```

Affiche :

Bonjour Jean Dupont

Il est possible d'utiliser des expressions littérales, des constantes et des opérateurs dans les valeurs par défaut.

2. Retour du type des déclarations

Depuis PHP 7, il est possible de définir le type des paramètres et de la valeur de retour d'une fonction. Cela permet d'avoir un code plus clair et plus précis.

Par exemple :

```
<?php
//déclaration d'une fonction retournant la somme des paramètres
function ajout(float $a, float $b):float {
    return $a + $b;
}

//affichage de la somme des valeurs
echo ajout(5.22, 6.65);

?>
```

Affiche :

01/11/87

Il est possible d'ajouter la directive **`declare(strict_types = 1)`** en début de fichier pour retourner une erreur en cas d'incompatibilité de type.

Par exemple :

```
<?php
declare(strict_types=1);

//déclaration d'une fonction retournant un type entier
function affiche(): int{
   return "1"; //la fonction retourne en fait une chaîne de caractères
}

try {
   affiche(); //appel de la fonction
} catch (TypeError $e) {
   echo($e->getMessage());
```

```
}
?>
```

Affiche :

Return value of affiche() must be of the type integer, string returned in test.php on line 5

En effet, la fonction est censée retourner un entier (int) mais retourne à la place une chaîne de caractères ("1").

3. Portée des variables

Cette notion est très importante car elle est source de beaucoup d'erreurs, surtout chez les personnes n'ayant pas appris le développement en général.

Les variables déclarées ou passées en paramètres d'une fonction ne sont valables que dans la fonction.

Par exemple :

```
<?php


function affiche_mot($nom) {
    echo $nom;
}
affiche_mot("Bonjour "); //affiche Bonjour
echo $nom;  //erreur car $nom n'est pas défini


?>
```

Affiche :

Bonjour

Notice : Undefined variable: nom

En effet, la variable `$nom` n'est valable que dans la fonction `affiche_mot`.

Pour ne pas avoir cette erreur, il aurait fallu définir la variable `$nom` en dehors de la fonction :

```
<?php


$nom = "Jean";
function affiche_mot($nom) {
    echo $nom;
}
```

```
affiche_mot("Bonjour "); //affiche Bonjour
echo $nom;  //affiche Jean


?>
```

Affiche :

Bonjour Jean

Attention : la variable `$nom` définie avant la fonction n'est pas la même que `$nom` passée en paramètre de la fonction !

De la même façon, une variable déclarée en dehors de la fonction n'est pas valable dans la fonction.

Par exemple :

```
<?php


$prenom = "Jean";
function affiche_mot() {
    echo $prenom; //affiche une erreur
}
affiche_mot();


?>
```

Affiche :

Notice: Undefined variable: prenom

4. Les variables globales

Une variable peut être déclarée avec le mot-clé **global**. Cela a pour effet de définir la variable dans tout le code PHP de votre page aussi bien dans les fonctions qu'ailleurs.

Si vous reprenez l'exemple précédent en ajoutant dans la fonction `$prenom` comme variable globale :

```
<?php


$prenom = "Jean";
function affiche_mot() {
    global $prenom;
    echo $prenom;
}
```

```
affiche_mot();


?>
```

Affiche :

Jean

L'autre solution consiste à utiliser le tableau associatif **$GLOBALS** qui contient toutes les variables avec leur valeur. Ce tableau a une portée dans toute la page PHP (cf. chapitre Transmettre des données d'une page à l'autre - Les variables superglobales).

5. Les variables statiques

Une variable déclarée avec le mot-clé `static` dans une fonction permet de conserver sa valeur lors de plusieurs appels de cette fonction. En effet, la valeur d'une variable normale est écrasée lors de chaque appel de la fonction :

```
<?php


function affiche_nombre() {
    $nombre = 0;
    $nombre=$nombre + 1;
    echo $nombre."; ";
}
affiche_nombre(); //affiche 1
affiche_nombre(); //affiche 1


?>
```

Affiche :

1; 1;

Si maintenant `$nombre` est déclaré comme `static` :

```
<?php


function affiche_nombre() {
    static $nombre = 0;
    $nombre=$nombre + 1;
    echo $nombre."; ";
}
affiche_nombre(); //affiche 1
```

```
affiche_nombre(); //affiche 2


?>
```

Affiche :

1; 2;

En effet, `$nombre` conserve sa valeur d'un appel à l'autre de la fonction.

6. Fonctions utiles

- La fonction **`isset()`** permet de tester l'existence d'une variable. Elle retourne true si la variable existe, false sinon.

Par exemple :

```
<?php


$phrase = "Bonjour, il fait beau";
if (isset($phrase)){
    echo "La variable existe.";
}
else {
   echo "La variable n'existe pas.";
}


?>
```

Affiche :

La variable existe.

Sa syntaxe est donc :

```
$existe = isset ($variable);
```

- La fonction **`var_dump()`** permet d'afficher le type du contenu et le contenu d'une variable.

Par exemple :

```
<?php


$phrase = "Bonjour, il fait beau";
var_dump($phrase);
```

```
?>
```

Affiche :

string(21) "Bonjour, il fait beau"

Sa syntaxe est donc :

```
var_dump($variable);
```

Cette fonction accepte aussi les tableaux en paramètres.

Par exemple :

```
<?php

$tableau = array("Fraise", "Banane", array(1, 2, 3));
var_dump($tableau);

?>
```

Affiche :

array(3) { [0]=> string(6) "Fraise" [1]=> string(6) "Banane" [2]=> array(3) { [0]=> int(1) [1]=> int(2) [2]=> int(3) } }

- La fonction **`empty()`** permet de tester si une variable est nulle ou pas. Elle retourne `true` si la valeur est nulle, 0 ou une chaîne vide, `false` sinon.

Par exemple :

```
<?php

$phrase = "Bonjour, il fait beau";
if (empty($phrase)){
    echo "La variable est nulle.";
}
else {
    echo "La variable n'est pas nulle.";
}

?>
```

Affiche :

La variable n'est pas nulle.

Depuis PHP 5.5, il est possible de passer une fonction en paramètre.

Par exemple :

```
<?php


function always_false(){
    return false;
}
if (empty(always_false())){
    echo 'Ceci sera affiché'
}


?>
```

Affiche :

Ceci sera affiché

Sa syntaxe est donc :

```
$nulle = empty($variable ou fonction());
```

7. Passage par référence

Par défaut lorsqu'une variable est passée en paramètre d'une fonction, elle est passée par valeur, c'est-à-dire que c'est une copie de la variable qui est passée en paramètre.

Exemple de passage par valeur :

```
<?php


function ajoute_monsieur($parametre) {
    $parametre = $parametre . " Monsieur";
}
$texte = "Bonjour";
ajoute_monsieur($texte);
echo $texte;


?>
```

Affiche :

Bonjour

En effet, la variable `$texte` passée en paramètre n'a pas été changée par la fonction car c'est seulement sa valeur qui est envoyée à la fonction.

Le passage du paramètre par référence se fait en ajoutant le symbole & devant la variable. Cela a pour effet de passer l'adresse mémoire de la variable et ainsi de pouvoir modifier sa valeur.

Exemple de passage par référence :

```
<?php


function ajoute_monsieur(&$parametre) { //passage par référence
    $parametre = $parametre . " Monsieur";
}
$texte = "Bonjour";
ajoute_monsieur($texte);
echo $texte;


?>
```

Affiche :

Bonjour Monsieur

En effet, la variable `$texte` est passée en paramètre de la fonction par l'intermédiaire de `&$parametre` puis est modifiée en concaténant `$paramètre` avec la chaîne de caractères " Monsieur". À la sortie de la fonction, `$texte` a donc pour valeur celle de `$parametre`, c'est-à-dire "Bonjour Monsieur".

8. Fonctions sur la gestion de fonction

- La fonction **`func_get_arg(int $numero)`** retourne un élément de la liste d'arguments passés en paramètre avec `$numero` comme le numéro de l'argument.

Par exemple :

```
<?php


function affiche_parametres($parametre1,$paramètre2) {
    echo "La valeur du premier argument est : ".func_get_arg(0);
    echo "La valeur du deuxième argument est : ".func_get_arg(1);
}
affiche_parametres("Bonjour","Robert");


?>
```

Affiche :

La valeur du premier argument est : Bonjour

La valeur du deuxième argument est : Robert

- La fonction **`func_num_args()`** retourne le nombre d'arguments passés en paramètre.

Par exemple :

```
<?php

function affiche_nombre_parametres($parametre1,$paramètre2) {
    echo "Le nombre d'arguments est : ".func_num_args();
}
affiche_nombre_parametres("Bonjour","Robert");

?>
```

Affiche :

Le nombre d'arguments est : 2

- La fonction **`func_get_args()`** retourne les arguments passés en paramètre sous forme de tableau.

Par exemple :

```
<?php

function affiche_parametres($parametre1,$paramètre2) {
    $num_args = func_num_args();
    $arg_liste = func_get_args();
    for ($i = 0; $i < $num_args; $i++) {
        echo "L'argument $i est : " . $arg_liste[$i] . "<br />\n";
    }
}
affiche_parametres("Bonjour","Robert");

?>
```

Affiche :

L'argument 0 est : Bonjour

L'argument 1 est : Robert

- La fonction **`function_exists()`** retourne un booléen indiquant si une fonction existe ou pas.

Par exemple :

```
<?php


function affiche_nombre_parametres($parametre1,$paramètre2) {
    echo "Le nombre d'arguments est : ".func_num_args();
}
if (function_exists('affiche_nombre_parametres')) {
    echo "La fonction existe.";
} else {
    echo "La fonction n'existe pas.";
}


?>
```

Affiche :

La fonction existe.

Il est possible de créer sa fonction dynamiquement grâce à la fonction **`create_function()`**. Pour plus d'informations, consultez la page suivante : http://www.php.net/manual/fr/function.create-function.php

9. Fonctions variables via l'opérateur ...

Depuis PHP 5.6, il est possible de passer un nombre quelconque de paramètres via l'opérateur **...** (trois points de suspension). Cela peut se révéler utile lorsque vous voulez utiliser la même fonction avec un nombre de paramètres différent.

Par exemple :

```
<?php
//fonction avec un nombre quelconque de paramètres
function fonction_variable(...$parametres)
{
      echo "Le nombre de paramètres est : ".count($parametres)."<br />";
      //transformation du tableau en chaîne de caractères
      echo "Les valeurs sont : ".implode(" ",$parametres)."<br />";
}


//Appel de la fonction
fonction_variable('Tigre');
fonction_variable('Tigre', 'Lion');
```

```
fonction_variable('Tigre', 'Lion', 'Zèbre');
?>
```

Affiche :

Le nombre de paramètres est : 1
Les valeurs sont : Tigre
Le nombre de paramètres est : 2
Les valeurs sont : Tigre Lion
Le nombre de paramètres est : 3
Les valeurs sont : Tigre Lion Zèbre

10. Décompression des arguments via l'opérateur ...

Depuis PHP 5.6, il est également possible de décompresser un tableau dans la liste d'arguments lors de l'appel d'une fonction. La syntaxe est **...** suivi du nom du tableau.

Par exemple :

```
<?php
//fonction de concaténation avec trois paramètres
function concatenation($a, $b, $c) {
{
     return $a." ". $b." ".$c."<br />";
}

//affichage du retour de la fonction de concaténation
echo concatenation('Mouton', 'Lion', 'Tigre');

$tableau1 = ['Lion', 'Tigre'];
//affichage du retour de la fonction de concaténation
echo concatenation('Mouton', ...$tableau1);

//affichage du retour de la fonction de concaténation
echo concatenation('Mouton', ...['Lion', 'Tigre']);
?>
```

Affiche :

Mouton Lion Tigre
Mouton Lion Tigre
Mouton Lion Tigre

11. Fonction anonyme

Depuis PHP 5.3, les fonctions anonymes (*closure* en anglais) font partie des plus grandes avancées en PHP. Ces fonctions ne portent pas de nom et peuvent être utilisées comme fonctions de rappel ou comme valeurs de variables.

Par exemple :

```
<?php
//déclaration de la fonction anonyme
$func_anonymous = function ()
{
      echo "<p>Bonjour !</p>";
}
//exécution de la fonction anonyme
$func_anonymous();
?>
```

Affiche :

Bonjour !

Bien sûr, il est possible d'ajouter des paramètres comme dans n'importe quelle fonction :

```
<?php
//déclaration de la fonction anonyme
$func_anonymous = function ($parametre)
{
      echo "<p>Bonjour ".$parametre." !</p>";
}
//exécution de la fonction anonyme
$func_anonymous("Robert");
?>
```

Affiche :

Bonjour Robert !

Grâce à la fonction **`array_walk`** prenant en paramètres un tableau et une fonction de retour (appelée *callback*), il est possible d'exécuter cette fonction de retour pour chaque élément du tableau.

Par exemple :

```
<?php
//déclaration du tableau
$data = array('tigre', 'lion', 'vache');
```

```
//appel de la fonction array_walk
array_walk($data, function (&$item) {
//passage par référence pour modifier les valeurs du tableau
    $item = ucwords($item); //mise en majuscule de la première lettre
});
//affichage du tableau modifié
var_dump($data);
?>
```

Affiche :

array(3) { [0]=> string(5) "Tigre" [1]=> string(4) "Lion" [2]=> string(5) "Vache" }

Les fonctions anonymes peuvent hériter des variables du contexte de leur parent. Ces variables doivent alors être passées derrière la signature de la fonction à l'aide du mot-clé `use` :

```
<?php
$nom = 'Robert';
//déclaration des fonctions
$fonction1 = function () {
      echo "<p>Bonjour, ".$nom." !</p>";
};
$fonction2 = function () use ($nom) {
      echo "<p>Bonjour, ".$nom." !</p>";
};

$fonction1();   //sans le mot-clé use
$fonction2();   //avec le mot-clé use
?>
```

Affiche :

Notice: Undefined variable: nom in test.php on line 4
Bonjour, !
Bonjour, Robert

Il se produit une erreur en ligne 4 car dans la fonction1 $nom n'est pas connu.

12. Fonction générateur

Depuis PHP 5.5, les fonctions générateurs (*generator* en anglais) permettent de retourner autant de valeurs que nécessaire grâce à l'opérateur **yield**. Il est ainsi possible de récupérer toutes les valeurs générées sans passer par un tableau. Une fonction générateur n'a donc pas besoin du mot-clé `return`.

Par exemple :

```
<?php
//déclaration d'une fonction générant les chiffres de 1 à 5
function boucle_yield() {
      for ($i = 1; $i <= 5; $i++) {
         // $i est préservé entre chaque production de valeur.
         yield $i
      }
}

//appel de la fonction
$generateur = boucle_yield();

//affichage des valeurs
foreach ($generateur as $val) {
      echo $val." ";
}
?>
```

Affiche :

1 2 3 4 5

Deuxième exemple :

```
<?php
//déclaration d'une fonction générant les carrés des nombres reçus
function carre($a) {
      foreach ($a as $val) yield $val*$val;
}

//déclaration d'un tableau de nombres
$arr = array(5,8,15,35);

//affichage des carrés des valeurs du tableau
foreach (carre($arr) as $val) {
      echo $val," ";
}
?>
```

Affiche :

25 64 225 1225

13. Récursivité

La récursivité signifie qu'une fonction s'appelle elle-même. Cela peut être très utile pour parcourir une arborescence, surtout si le nombre de branches n'est pas connu à l'avance.

Par exemple :

```
<?php


function fonction_recursive($n)
{
 $n++; //incrémente n de 1
 echo "$n,";
 if($n < 10){ // si n est inférieur à 10, rappel de la fonction
  fonction_recursive($n);
 }
}
fonction_recursive(0); // affiche les nb de 1 à 10


?>
```

Affiche :

1,2,3,4,5,6,7,8,9,10,

Dans la condition `if($n < 10)`, la fonction s'appelle elle-même, ce qui permet de l'appeler dix fois en tout. En effet, s'il n'y avait pas la condition (`$n < 10`), il y aurait une boucle infinie.

Voici ci-dessous le code pour afficher les valeurs de tableaux multidimensionnels :

```
<?php


function afficher_tableau($tableau,$titre="",$niveau=0) {
      // Paramètres
      //    - $tableau = tableau dont il faut afficher le contenu
      //    - $titre = titre à afficher au-dessus du contenu
      //    - $niveau = niveau d'affichage
      if ($titre != "") { // S'il y a un titre, l'afficher.
        echo "<br /><b>".$titre."</b><br />";
      }
```

```
      // Tester s'il y a des données.
      if (isset($tableau)) { // il y a des données
        // Parcourir le tableau passé en paramètre.
        foreach ($tableau as $cle => $valeur) {
          // Afficher la clé (avec indentation en fonction du niveau)
         // htmlentities() est une fonction transformant les
         //caractères spéciaux HTML
           echo str_pad('',12*$niveau, ' ').
            htmlentities($cle).' = ';
          // Afficher la valeur
          if (is_array($valeur)) { // vérifie si c'est un tableau
            echo '<br />';
            // Appelle récursivement afficher_tableau pour
            // afficher le tableau en question
            afficher_tableau($valeur,'',$niveau+1);
          } else { // c'est une valeur scalaire
            // Afficher la valeur
            echo htmlentities($valeur).'<br />';
          }
        }
      } else { // pas de données
        echo '<br />';
      }
    }
    // Déclaration des tableaux
    $tab_caracteristique_dupont = array("prénom" => "PAUL",
"profession" => "ministre","age" => 50);
    $tab_caracteristique_durand = array("prénom" => "ROBERT",
"profession" => "agriculteur","age" => 45);
    $tab_personne['DUPONT'] = $tab_caracteristique_dupont;
    $tab_personne['DURAND'] = $tab_caracteristique_durand;

    // Afficher un tableau à deux dimensions (Nom/Caractéristiques)
    afficher_tableau($tab_personne,'Nom/Caractéristiques');

?>
```

Affiche :

Nom/Caractéristiques

DUPONT =

prénom = PAUL

profession = ministre

age = 50

DURAND =

prénom = ROBERT

profession = agriculteur

age = 45

L'avantage de cette fonction récursive est que si vous aviez un tableau à trois, quatre dimensions ou plus, le code de la fonction ne changerait pas car il ne dépend pas du nombre de dimensions.

14. Fonctions prédéfinies dans PHP

Il existe à peu près 4500 fonctions prédéfinies en PHP. Vous les trouvez regroupées par thèmes à cette adresse : http://www.php.net/manual/fr/funcref.php

Vous avez vu certaines d'entre elles dans les sections précédentes comme **`substr()`** ou **`implode()`**. Nous allons voir maintenant quelques fonctions supplémentaires qui pourraient vous être utiles.

a. Générer un nombre aléatoire

La fonction **`rand()`** permet de générer une valeur aléatoire entre 0 et 32768. Vous pouvez passer en paramètres un minimum et un maximum.

Par exemple :

```
<?php


echo rand()."<br />";
echo rand(10,20)."<br />";


?>
```

Affiche :

b. Arrondir un nombre décimal

La fonction **`round($nombre_décimal)`** permet d'arrondir un nombre décimal.

Par exemple :

```
<?php

echo round(3.141592);

?>
```

Affiche :

3

Cette fonction accepte en paramètre supplémentaire la précision, c'est-à-dire le nombre de chiffres souhaités après la virgule.

Par exemple :

```
<?php

echo round(3.141592,2); //2 chiffres après la virgule

?>
```

Affiche :

3.14

Autre exemple :

```
<?php

echo round(3.779592,2); //2 chiffres après la virgule

?>
```

Affiche :

3.78

En effet, c'est bien un arrondi qui est fait.

c. Récupérer la valeur absolue d'un nombre

La fonction **`abs($nombre)`** permet de récupérer la valeur absolue d'un nombre.

Par exemple :

```
<?php

echo abs(-5.2);

?>
```

Affiche :

5.2

Cette fonction accepte aussi en paramètre une chaîne de caractères.

Par exemple :

```
<?php

echo abs("68");

?>
```

Affiche :

68

d. Créer un identifiant unique

La fonction **`uniqid()`** permet de générer une valeur de treize caractères de façon aléatoire de telle façon que cette valeur soit unique. C'est parfois pratique lorsque vous voulez générer des identifiants uniques à insérer en base de données.

Par exemple :

```
<?php

echo "Id unique : ".uniqid()."<br />";
echo "Id unique : ".uniqid()."<br />";
echo "Id unique : ".uniqid();

?>
```

Affiche :

Id unique : 4df0d26502f82, Id unique : 4df0d26502f86, Id unique : 4df0d26502f87

e. Afficher les informations sur PHP

La fonction **phpversion()** permet d'afficher la version courante de PHP.

Par exemple :

```
<?php

echo phpversion();

?>
```

Affiche :

5.3.0

La fonction **phpinfo()** permet d'afficher les informations sur la configuration de PHP installée sur votre serveur comme les variables d'environnement ou la configuration d'Apache. Ces informations sont stockées dans le fichier php.ini dont nous expliquerons l'utilisation dans un prochain chapitre.

Par exemple :

```
<?php

phpinfo();

?>
```

La fonction **ini_get_all()** retourne aussi toutes les informations du fichier php.ini mais sous forme de tableau.

Enfin, la fonction **get_loaded_extensions()** retourne un tableau contenant toutes les extensions chargées. Nous en reparlerons dans le chapitre Configuration.

f. Envoyer un e-mail

La fonction **mail()** permet d'envoyer un e-mail. C'est une fonction basique qui n'est pas à utiliser si vous voulez envoyer un grand volume d'e-mails car elle ferme et rouvre une connexion au serveur à chaque envoi. Il existe d'autres fonctions comme PEAR ou PHPMailer plus pratiques et plus efficaces. Néanmoins, l'étude de cette fonction vous permet de voir les bases de l'envoi d'e-mail qui sont communes à toutes les fonctions.

Les paramètres de la fonction sont :

- $to : le ou les destinataires de l'e-mail.
- $subject : sujet de l'e-mail.
- $message : contenu de l'e-mail.
- $headers : paramètre optionnel contenant l'en-tête de l'e-mail.

L'en-tête permet de définir l'expéditeur de l'e-mail (From), le type MIME, l'encodage ainsi que d'autres paramètres dont vous trouverez plus d'informations sur ce lien : http://php.net/manual/fr/function.mail.php

L'expéditeur (From), l'adresse SMTP ainsi que le numéro du port SMTP sont définis dans le fichier php.ini accessible par le menu **Configuration - PHP**.

Par exemple :

```
<?php


$to = 'personne@example.com';

$subject = 'Sujet';

$message = 'Bonjour à tous !';

$headers = 'From: webmaster@monsite.com'."\r\n"

     'Reply-To: webmaster@monsite.com'."\r\n"

     'MIME-Version: 1.0'."\r\n";


mail($to, $subject, $message, $headers);

?>
```

La fonction `mail()` ne permet pas de s'authentifier et donc ne fonctionne pas en local en utilisant le serveur SMTP de Gmail par exemple.

Pour cela, vous devez utiliser la bibliothèque Mail-1.2.0 de PEAR disponible à cette adresse : http://pear.php.net/package/Mail/download/1.2.0/

Par défaut, l'e-mail envoyé est en format texte. Pour l'envoyer sous format HTML, il faut déclarer ce format dans l'en-tête (header) :

```
$headers = 'From: webmaster@monsite.com'."\r\n".

     'Reply-To: webmaster@monsite.com'."\r\n".

     'MIME-Version: 1.0'."\r\n".

     'Content-type: text/html; charset=iso-8859-1'."\r\n";
```

15. Stocker une fonction dans une variable

PHP permet de stocker une fonction dans une variable. Il suffit de passer en paramètre le nom d'une fonction déjà existante.

Par exemple :

```php
<?php


function ajouter($parametre1,$parametre2) {
  return $parametre1 + $parametre2;
}
function soustraire($parametre1,$parametre2) {
  return $parametre1 - $parametre2;
}
function operation($type_operation,$valeur1,$valeur2) {
  return $type_operation($valeur1,$valeur2);
}
echo "6 + 4 =".operation('ajouter',6,4).", ";
echo "6 - 4 =".operation('soustraire',6,4);


?>
```

Affiche :

6 + 4 =10, 6 - 4 =2

Vous pouvez constater que le type d'opération est passé en paramètre de la fonction **operation**. Cette fonction récupère le type d'opération (ajouter par exemple) et va appeler la fonction **ajouter** avec ses paramètres. Bien entendu, pour que cela puisse fonctionner, il faut que la fonction **ajouter** existe déjà.

Cette notion peut vous être utile si vous faites de la programmation orientée objet en PHP, mais nous n'irons pas plus loin car ceci dépasse le cadre de ce support.

16. Exercices sur les fonctions

a. Énoncés

Exercice 1 (difficulté facile)

Créer un tableau contenant dix chiffres de 1 à 10 puis afficher ceux-ci séparés par un point-virgule sans utiliser de boucle.

Exercice 2 (difficile)

Créer un tableau contenant dix chiffres aléatoires entre 1 à 100 puis trier celui-ci sans utiliser les méthodes de tri de tableau comme sort(). Il faudra créer une fonction pour échanger deux valeurs dans un tableau. Afficher ces valeurs séparées par une virgule.

Exercice 3 (moyen)

Écrire une fonction qui permet de calculer la factorielle d'un nombre de manière récursive.

Par exemple la factorielle de 7 est : 1*2*3*4*5*6*7

Afficher alors la factorielle de 20 (2.4329020081766E+18).

Exercice 4 (difficile)

Créer une fonction prenant en paramètre un tableau contenant un nombre quelconque de mots pour afficher une phrase contenant de manière aléatoire tous ces mots.

Exercice 5 (difficile) : création d'un tableau HTML avec des valeurs au cube

Soit le tableau A avec les éléments 3,8,15,16. Créer un tableau B à l'aide d'une boucle contenant tous les éléments de 1 à 20 sauf les éléments du tableau A. Créer une fonction qui calcule le cube de ce chiffre et afficher dans un tableau HTML les éléments du tableau B dans une première colonne et le cube des éléments de B dans une seconde colonne.

Les dates

Dans cette section, il est question de toutes les fonctions PHP permettant de manipuler les dates. En effet, souvent les dates sont récupérées dans un format suivant la langue et vous êtes obligé de la retransformer dans votre langue.

- La fonction **`time()`** retourne l'heure courante, mesurée en secondes depuis le début de l'époque UNIX (1er janvier 1970 00:00:00 GMT). Cette heure est aussi appelée timestamp UNIX.

Par exemple :

```
<?php


echo time();


?>
```

Affiche :

1381329777

Sa syntaxe est donc :

```
time()
```

Cette fonction est surtout utilisée pour effectuer des calculs sur des dates comme par exemple pour trouver la durée d'un traitement en base de données.

- La fonction **`date()`** retourne la date dans le format passé en paramètre.

Par exemple :

```
<?php
```

```
echo date('d.m.y');


?>
```

Affiche :

09.10.13

Sa syntaxe est donc :

```
$date_du_jour = date($format)
```

avec `$format` une chaîne contenant les lettres permettant de définir le format.

Voici la liste des principaux formats à utiliser dans la fonction date. Cette liste est non exhaustive et vous trouverez plus d'informations à cette adresse : http://php.net/manual/fr/function.date.php

Jour

J	Jour du mois sur deux chiffres sans les zéros initiaux 1 à 31.
d	Jour du mois sur deux chiffres avec un zéro initial en fonction du jour 01 à 31.
l	(L minuscule) Jour de la semaine en anglais Sunday à Saturday.
w	Jour de la semaine au format numérique 0 (dimanche) à 6 (samedi).
z	Jour de l'année 0 à 366.

Semaine

W	Numéro de semaine dans l'année (les semaines commencent le lundi). Exemple : 42 (la 42ème semaine de l'année).

Mois

F	Mois, textuel, version longue en anglais, comme January ou December.
m	Mois au format numérique, avec zéros initiaux 01 à 12.
n	Mois sans les zéros initiaux 1 à 12.
t	Nombre de jours dans le mois 28 à 31.

Année

L	Est-ce que l'année est bissextile : 1 si bissextile, 0 sinon.
Y	Année sur 4 chiffres (par exemple 1999 et 2003).
y	Année sur 2 chiffres (par exemple 99 et 03).

Heure

a	Ante méridien et Post méridien (minuscules) am ou pm.
A	Ante méridien et Post méridien (majuscules) AM ou PM.
g	Heure (format 12h) sans les zéros initiaux 1 à 12.
G	Heure (format 24h) sans les zéros initiaux 0 à 23.
h	Heure (format 12h) avec les zéros initiaux 01 à 12.
H	Heure (format 24h) avec les zéros initiaux 00 à 23.
s	Secondes avec zéros initiaux 00 à 59.
i	Minutes avec zéros initiaux 00 à 59.

Cette fonction **`date()`** peut prendre aussi en paramètre optionnel le timestamp UNIX pour définir quelle autre date que la date du jour vous souhaitez lui faire afficher.

Par exemple :

```
<?php

$semaineProchaine = time() + (7 * 24 * 60 * 60);// Ajout d'une semaine
//à l'heure actuelle. Soit 7 jours = 7x24 heures = 7x24x60 minutes =
//7x24x60x60 secondes
echo "Aujourd'hui :       ".date('d-m-Y')."; ";
echo "Semaine prochaine : ".date('d-m-Y', $semaineProchaine)."\n";

?>
```

Affiche :

Aujourd'hui : 01-02-2013, Semaine prochaine : 08-02-2013

- La fonction **`mktime()`** retourne le timestamp UNIX à partir d'une date passée en paramètre.

Par exemple :

```
<?php

echo mktime(0,0,0,2,1,2012); //affiche le timestamp du 01/02/2012

?>
```

Affiche :

1328050800

Sa syntaxe est :

```
$timestamp = mktime($heure,$minute,$seconde,$mois,$jour,$annee)
```

Cette fonction corrige automatiquement les dates invalides. Par exemple, le 32 janvier sera corrigé en 1 février.

- La fonction **`microtime()`** retourne le timestamp UNIX en microsecondes. Elle prend en paramètre optionnel un booléen qui permet de retourner un nombre réel s'il est à vrai ou une chaîne de caractères sinon.

Par exemple :

```
<?php


// Affichage sous la forme d'une chaîne.
echo microtime(). '<br />';
// Affichage sous la forme d'un réel.
echo microtime(TRUE). '<br />';
// Pour n'afficher que les microsecondes, transformation (cast)
// de la chaîne en réel.
echo (float) microtime() .'<br />';


?>
```

Affiche :

0.31211200 1328104356

1328104356.3121

0.3121

Sa syntaxe est donc :

```
$microseconde = microtime($bool)
```

- La fonction **`getdate()`** retourne un tableau associatif de la date et de l'heure actuelle.

Par exemple :

```
<?php


print_r(getdate());


?>
```

Affiche :

Array ([seconds] => 37 [minutes] => 48 [hours] => 10 [mday] => 15 [wday] => 3 [mon] => 6 [year] => 2011 [yday] => 165 [weekday] => Wednesday [month] => June [0] => 1308127717)

Sa syntaxe est donc :

```
$tableau = getdate()
```

- La fonction **`checkdate()`** indique si une date est valide ou non. Cette fonction tient compte des années bissextiles. Elle prend en paramètres le mois, le jour et l'année et renvoie vrai ou faux.

Par exemple :

```
<?php

$valide = checkdate(13, 10, 2011);

if($valide == true )
 {
    echo 'La date est valide';
 }
else
 {
      echo 'La date n\'est pas valide';
 }

?>
```

Affiche :

La date n'est pas valide

En effet, il n'existe pas de 13ème mois.

Sa syntaxe est donc :

```
$valide = checkdate($mois,$jour,$annee)
```

- La fonction **`strtotime()`** permet de convertir un texte en anglais en timestamp donc en date.

Par exemple :

```
<?php

//strtotime renvoie un timestamp et la fonction date() permet d'afficher
//la date dans un bon format à partir du timestamp
```

```
echo strtotime("now")."<br />";

echo strtotime("10 September 2011")."<br />";

echo strtotime("next Thursday")."<br />";

echo strtotime("last Tuesday")."<br />";

echo strtotime("+1 day")."<br />";

echo strtotime("+1 week")."<br />";

echo strtotime("+2 week 2 days 2 hours 2 seconds")."<br />";

?>
```

Affiche :

1381822566, 15-10-2013

1315605600, 10-09-2011

1381960800, 17-10-2013

1381183200, 08-10-2013

1381908966, 16-10-2013

1382427366, 22-10-2013

1383215768, 31-10-2013

Sa syntaxe est donc :

```
$timestamp = strtotime($chaine)
```

- La fonction **`strftime()`** permet de convertir une date sous forme de timestamp en chaîne correctement formatée. Elle prend en paramètre le format de type chaîne de caractères et le timestamp en option si vous ne voulez pas la date actuelle. Cette fonction est utilisée avec **`setlocale()`** qui permet de définir le pays où l'on est.

Par exemple :

```
<?php
```

```
setlocale(LC_ALL, "fr_FR","fra");
echo strftime("En France le jour est : %A <br />");

echo strftime("Nous sommes le %d %m %Y <br />");

setlocale(LC_TIME, "de_DE","deu");
echo strftime("En Allemagne le mois est : %B.\n");

?>
```

Affiche :

En France le jour est : mercredi

Nous sommes le 09 10 2013

En Allemagne le mois est : Oktober.

Sa syntaxe est donc :

```
$chaine = strftime($format)
```

Voici quelques options de format :

`%d` : jour du mois, sur deux chiffres (01 à 31).

`%m` : numéro du mois, sur deux chiffres (01 à 12).

`%y` : année sur deux chiffres (01 par exemple).

`%Y` : année sur quatre chiffres (2001 par exemple).

`%H` : heure, au format 24h.

`%M` : minutes sur deux chiffres (00 à 59).

`%S` : secondes sur deux chiffres (00 à 59).

`%a` : nom abrégé du jour de la semaine.

`%A` : nom complet du jour de la semaine.

`%j` : numéro du jour de l'année sur trois chiffres (001 à 365).

`%w` : numéro du jour de la semaine (0 = dimanche à 6 = samedi).

`%u` : numéro du jour de la semaine (1 = lundi à 7 = dimanche).

`%b` ou `%h` : nom abrégé du mois.

%B : nom complet du mois.

%U : numéro de semaine dans l'année (le premier dimanche de l'année étant le premier jour de la première semaine).

%W : numéro de semaine dans l'année (le premier lundi de l'année étant le premier jour de la première semaine).

%V : numéro de la semaine dans l'année selon la norme ISO 9601.

%c : format par défaut pour la date et l'heure.

%x : format par défaut pour la date seule.

%X : format par défaut pour l'heure seule.

%R : identique à %H:%M.

%T : identique à %H:%M:%S.

%Z : fuseau horaire, ou nom ou abréviation.

%t : tabulation.

%n : retour à la ligne.

%% : un caractère '%' littéral.

- La fonction **date_default_timezone_set()** définit le décalage horaire de toutes les fonctions date et heure. Cette fonction prend en paramètre l'identifiant de décalage qui est une chaîne de caractères avec la zone et le pays. Lorsque vous voulez formater une date, il faut définir la zone de décalage horaire sinon vous aurez un message d'avertissement de type E_NOTICE.

Par exemple, pour la France :

```
<?php
date_default_timezone_set('Europe/Paris');
?>
```

- La fonction **date_create_from_format()** permet de retourner un objet date formaté à partir d'une chaîne de caractères contenant une date. Cette fonction peut aussi s'appeler sous la forme **DateTime::createFromFormat()**. Elle prend en paramètres le format de type chaîne de caractères, la date et l'heure sous forme d'une chaîne de caractères et en option l'objet **DateTimeZone** définissant la zone du fuseau horaire. Le format est le même que celui utilisé avec la fonction **date()**.

Par exemple :

```
<?php
date_default_timezone_set('Europe/Paris');
$format = 'Y-m-d H:i:s';
```

```
$date = date_create_from_format($format, '2011-11-15 12:14:19');
echo "Affichage sous le format jour-mois-année heure:minute:seconde -> "
.date-format($date, 'd-m-Y H:i:s');
?>
```

Affiche :

Affichage sous le format jour-mois-année heure:minute:secondes -> 15-11-2011 12:14:19

Les fichiers

1. Introduction

Il est parfois très utile de stocker des informations dans un fichier sur le serveur plutôt que dans une base de données. Cela peut être plus rapide et plus facilement accessible mais ce sera moins sécurisé et il peut y avoir des conflits d'écriture en cas de connexions simultanées.

Nous allons voir dans la suite de cette section les fonctions les plus couramment utilisées.

Pour commencer, il faut créer un fichier appelé « fichier.txt » dans le répertoire www, c'est-à-dire là où se trouve vos pages PHP. Il faut absolument que ce fichier ait les droits en écriture pour pouvoir écrire dedans. Cela sera automatiquement le cas lorsque vous travaillez en local mais il faudra changer éventuellement les permissions du fichier si vous transférez votre fichier par FTP chez un hébergeur.

2. Lecture rapide

- La fonction **`file_get_contents()`** permet de lire le contenu d'un fichier et de le retourner dans une chaîne de caractères.

Dans cet exemple, le fichier texte contient la phrase "Bonjour !".

```
<?php

$contenu = file_get_contents('fichier.txt');
echo $contenu;

?>
```

Affiche :

"Bonjour !"

- La fonction **`readfile()`** permet aussi de lire le contenu d'un fichier mais elle retourne le nombre de caractères du fichier et affiche automatiquement le contenu.

Dans cet exemple, le fichier texte contient la phrase : "Bonjour !"

```
<?php
```

```
$fichier = 'fichier.txt';
$nombre=readfile($fichier);
echo "<br /> Le nombre de caractères du fichier est : ".$nombre;

?>
```

Affiche :

"Bonjour !"

Le nombre de caractères du fichier est : 11

- La fonction **`file()`** permet de lire le contenu d'un fichier mais elle retourne le contenu dans un tableau ligne par ligne.

Dans cet exemple, le fichier texte contient la phrase "Bonjour !", un saut de ligne et la phrase "Monsieur Dupont.".

```
<?php

$tableau = file('fichier.txt');
foreach ($tableau as $ligne) {
   echo $ligne."<br />";
}

?>
```

Affiche :

"Bonjour !"

"Monsieur Dupont."

En effet, chaque ligne du fichier se retrouve dans chaque élément du tableau.

3. Écriture rapide

La fonction **`file_put_contents()`** permet d'écrire le contenu d'une chaîne de caractères dans un fichier. Elle prend en paramètres le nom du fichier et la variable contenant le texte à insérer dans le fichier. Si le fichier existe déjà, son contenu est écrasé.

Par exemple :

```
<?php

$contenu = "Bonjour Mr Dupont.";
```

```
file_put_contents("fichier.txt",$contenu);


?>
```

Écrit dans le fichier nommé fichier.txt :

Bonjour Mr Dupont.

4. Ouverture et fermeture d'un fichier

La fonction **`fopen()`** permet de déclencher l'ouverture du fichier. Cette fonction a deux paramètres : le nom du fichier et le mode d'ouverture du fichier.

Par exemple :

```
<?php


$ressource = fopen('fichier.txt', 'r');


?>
```

La variable `$ressource` contient un objet nous permettant de manipuler le fichier. Nous verrons plus loin son utilisation.

Vous remarquez le caractère 'r' en deuxième paramètre.

Le `'r'` signifie que vous ouvrez le fichier en lecture seule.

Le `'r+'` signifie que vous ouvrez le fichier en lecture/écriture.

Le `'w'` signifie que vous ouvrez le fichier en écriture seule, cela vide le fichier et le crée s'il n'existe pas.

Le `'w+'` signifie que vous ouvrez le fichier en lecture/écriture, cela vide le fichier et le crée s'il n'existe pas.

Le `'a'` signifie que vous ouvrez le fichier en écriture seule en ajout et crée le fichier s'il n'existe pas. Ce qui sera écrit le sera à la suite sans modifier le texte existant.

Le `'a+'` signifie que vous ouvrez le fichier en lecture/écriture en ajout et crée le fichier s'il n'existe pas.

- La fonction **`fclose()`** permet de fermer le fichier. Cette fonction retourne `true` ou `false` en cas d'erreur.

Par exemple :

```
<?php
```

```
$ressource = fopen('fichier.txt', 'r');
fclose($ressource);


?>
```

5. Lecture et écriture

La fonction **`fgetc()`** permet de lire le fichier caractère par caractère. Il faut donc l'inclure dans une boucle pour parcourir tous les caractères du fichier.

Par exemple :

```
<?php


$ressource = fopen('fichier.txt', 'r');
if (!$ressource) { //vérifie si le fichier est bien ouvert
    echo "Impossible d'ouvrir le fichier fichier.txt";
}
//boucle tant qu'il y a un caractère
while (false !== ($char = fgetc($ressource))) {
    echo $char;
}


fclose($ressource);


?>
```

Affiche :

"Bonjour !" "Monsieur Dupont."

Vous remarquez que le saut de ligne n'est pas lu, donc cette fonction n'est pas très pratique.

- La fonction **`fgets()`** permet de lire le fichier ligne par ligne. Il faut donc l'inclure dans une boucle pour parcourir toutes les lignes du fichier. Elle prend en paramètre la ressource du fichier et la taille de chaque ligne en option. La fonction récupère ce nombre de caractères ou bien récupère les caractères jusqu'à ce qu'elle trouve une fin de ligne.

Par exemple :

```
<?php


$ressource = fopen('fichier.txt', 'r');
if ($ressource) { //si le fichier a bien été ouvert
```

```
    while (!feof($ressource)) { //tant que la fin de fichier
                                //n'est pas trouvée
        $buffer = fgets($ressource, 4096); //4096-1 est le nombre
//maximum de caractères ramenés par ligne, c'est-à-dire la taille
//du buffer-1
        echo $buffer."<br />";
    }
}
fclose($ressource);

?>
```

Affiche :

"Bonjour !"

"Monsieur Dupont."

La fonction **`feof()`** retourne true si la fin de fichier est trouvée.

Si vous voulez afficher les lignes cinq par cinq caractères :

```
<?php

$ressource = fopen('fichier.txt', 'r');
if ($ressource) {
   while (!feof($ressource)) { //tant que la fin de fichier n'est
                               //pas trouvée
       $buffer = fgets($ressource, 6); //6-1 est le nombre maximum
                               //de caractères ramenés sur une ligne
       echo $buffer."<br />";
   }
}
fclose($ressource);

?>
```

Affiche :

"Bonj

our !

"

"Mons

ieur

Dupon

t."

- La fonction **`fread()`** permet de lire un fichier en entier et de retourner son contenu dans une chaîne de caractères. Elle prend en paramètre la ressource et en option la longueur maximale à lire en octets.

Par exemple :

```
<?php


$ressource = fopen('fichier.txt', 'r');
if ($ressource) {
    $contenu = fread($ressource, 10000); //contenu limité à
                                         //10000-1 caractères
    echo $contenu;
}
fclose($ressource);


?>
```

Affiche :

"Bonjour !" "Monsieur Dupont."

Vous remarquez que les chaînes "Bonjour !" et "Monsieur Dupont." sont sur la même ligne car le saut de ligne du fichier texte ne correspond pas au saut de ligne HTML (
).

- La fonction **`fwrite()`** permet d'écrire dans un fichier. Elle prend en paramètres la ressource et une chaîne de caractères à insérer dans le fichier. Cette fonction est un alias de **`fputs`** (voir plus loin).

Par exemple :

```
<?php


$ressource = fopen('fichier.txt', 'w');
if ($ressource) {
    fwrite($ressource, 'Bonjour Madame Durand.');
}
fclose($ressource);
```

```
?>
```

Affiche dans le fichier fichier.txt :

Bonjour Madame Durand.

Si vous écrivez :

```
<?php


$ressource = fopen('fichier.txt', 'w');
if ($ressource) {
   fwrite($ressource, 'Bonjour ');
   fwrite($ressource, 'Madame Durand.');
}
fclose($ressource);


?>
```

Cela ne change rien. Le fichier contiendra toujours Bonjour Madame Durand.

Si vous voulez écrire Bonjour sur une ligne et Madame Durand sur une autre ligne, il faut insérer un saut de ligne après Bonjour. Le saut de ligne en format texte est `\r\n`. Il s'écrit en tant que constante PHP_EOL en PHP.

Donc :

```
<?php


$ressource = fopen('fichier.txt', 'w');
if ($ressource) {
   fwrite($ressource, 'Bonjour'.PHP_EOL);
   fwrite($ressource, 'Madame Durand.');
}
fclose($ressource);


?>
```

Affiche dans le fichier fichier.txt :

Bonjour

Madame Durand.

Si maintenant vous voulez ajouter du texte, il faut ouvrir le fichier en mode ajout.

Par exemple :

```
<?php


$ressource = fopen('fichier.txt', 'a');
if ($ressource) {
   fwrite($ressource, ' Au revoir.');
}
fclose($ressource);


?>
```

Affiche dans le fichier fichier.txt :

Bonjour

Madame Durand. Au revoir.

- La fonction **`rewind()`** permet de replacer le pointeur en début de fichier. Cela a pour effet de récrire par-dessus le texte existant.

Par exemple :

```
<?php


$ressource = fopen('fichier.txt', 'r+');
if ($ressource) {
    fwrite($ressource, 'Au revoir Madame Durand.');
    rewind($ressource); // place le pointeur au début
    fwrite($ressource, 'A bientôt ');
}
fclose($ressource);


?>
```

Affiche dans le fichier fichier.txt :

A bientôt Madame Durand.

En effet, la chaîne de caractères "Au revoir" va être remplacée par la chaîne de caractères "A bientôt" car la fonction **`rewind()`** va replacer le pointeur d'écriture en début de fichier avant d'écrire A bientôt.

Attention, les caractères écrits remplacent ceux déjà présents !

- La fonction **`fputs()`** permet d'écrire une ligne dans un fichier. Elle prend en paramètres la ressource et la chaîne de caractères à écrire.

Par exemple :

```
<?php

$ressource = fopen('fichier.txt', 'r+');
if ($ressource) {
   fputs($ressource, 'Au revoir');
   fputs($ressource, ' Madame Durand.');
}
fclose($ressource);

?>
```

Affiche dans le fichier fichier.txt :

Au revoir Madame Durand.

- Si après une instruction `fgets` ou `fputs`, vous voulez retourner au début du fichier, il faut utiliser la fonction **`fseek()`**. Cette fonction prend en paramètres la ressource et la position dans le fichier.

Par exemple :

```
<?php

$ressource = fopen('fichier.txt', 'r+');
if ($ressource) {
  fputs($ressource, 'Au revoir');
  fseek($ressource, 0); //repositionne le curseur en début de fichier
  fputs($ressource, 'Madame Durand.');
}
fclose($ressource);

?>
```

Affiche dans le fichier fichier.txt :

Madame Durand.

- Enfin, la fonction **`ftell()`** permet de connaître la position actuelle du pointeur. Cette fonction prend en paramètre la ressource.

Par exemple :

```php
<?php


$ressource = fopen('fichier.txt', 'r+');
if ($ressource) {
   fputs($ressource, 'Au revoir');
   fputs($ressource, 'Madame Durand.');
   echo ftell($ressource);
}
fclose($ressource);


?>
```

Affiche :

23

En effet, les chaînes de caractères 'Au revoir' et 'Madame Durand' contiennent 23 caractères.

6. Accès concurrent

Le problème est que lorsque vous avez un fichier texte sur un serveur, vous pouvez être plusieurs personnes à vouloir écrire dedans simultanément, ce qui provoquerait une erreur.

Pour éviter ce problème, il existe une fonction **`flock()`** qui prend en paramètre la ressource et une constante signifiant le type de verrouillage souhaité.

Un verrou permet de signifier que vous êtes en train d'utiliser le fichier. Il existe quatre types de verrou :

1. LOCK_SH pour acquérir un verrou partagé (lecture).
2. LOCK_EX pour acquérir un verrou exclusif (écriture).
3. LOCK_UN pour libérer un verrou (partagé ou exclusif).
4. LOCK_NB si vous voulez que flock() ne se bloque pas durant le verrouillage (non supporté sous Windows). Cette option est utilisée conjointement (OU binaire |) avec LOCK_EX ou LOCK_SH et la fonction flock() retourne alors une erreur si le fichier est déjà verrouillé.

Cette fonction renvoie `true` en cas de succès et `false` sinon.

Par exemple :

```php
<?php


$ressource = fopen('fichier.txt', 'r+');
if (flock($ressource, LOCK_EX)) { // pose un verrou exclusif
    fwrite($ressource, "Le verrou est maintenant posé.");
```

```
    flock($ressource, LOCK_UN); // libère le verrou
} else {
    echo "Impossible de verrouiller le fichier !";
}
fclose($ressource);


?>
```

Affiche dans le fichier texte :

Le verrou est maintenant posé.

7. Manipulation de fichiers

- La fonction **`copy()`** permet de recopier un fichier. Cette fonction prend en paramètres une chaîne de caractères contenant le fichier source et une autre chaîne de caractères contenant le fichier destination.

Par exemple :

```
<?php


copy("fichier.txt","test.txt");


?>
```

a pour effet de recopier le fichier "fichier.txt" dans le fichier "test.txt" du même répertoire.

- La fonction **`file_exists()`** permet de tester l'existence d'un fichier ou d'un dossier. Cette fonction prend en paramètre une chaîne de caractères contenant le nom du fichier ou du dossier.

Par exemple :

```
<?php


$fichier = 'fichier.txt';
if(file_exists($fichier)){
   echo "Ce fichier existe.";
}
else {
   echo "Ce fichier n'existe pas.";
}
```

```
?>
```

Affiche dans le fichier texte :

Ce fichier existe.

- La fonction **`is_file()`** permet aussi de tester l'existence d'un fichier mais ne fonctionne pas avec les répertoires. Cette fonction prend en paramètre une chaîne de caractères contenant le nom du fichier.

Par exemple :

```
<?php


$fichier = 'fichier.txt';
if(is file($fichier)){
   echo "Ce fichier existe.";
}
else {
   echo "Ce fichier n'existe pas.";
}


?>
```

Affiche dans le fichier texte :

Ce fichier existe.

- La fonction **`is_executable()`** permet de tester si le fichier est exécutable. Cette fonction prend en paramètre une chaîne de caractères contenant le nom du fichier.

Par exemple :

```
<?php


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


?>
```

Affiche dans le fichier texte :

Ce fichier n'est pas exécutable.

- La fonction **`touch()`** permet de créer un fichier. Cette fonction prend en paramètre une chaîne de caractères contenant le nom du fichier. Si le fichier existe déjà, cela change uniquement sa date de dernière modification.

Par exemple :

```
<?php
   touch("fichier.txt");


?>
```

a pour effet de créer le fichier "fichier.txt" s'il n'existe pas.

- La fonction **`unlink()`** permet de supprimer un fichier. Cette fonction prend en paramètre une chaîne de caractères contenant le nom du fichier. Il faut penser à vérifier l'existence du fichier avant de le supprimer sinon PHP génère une erreur.

Par exemple :

```
<?php


if(file_exists("fichier.txt")){
    unlink("fichier.txt");
}


?>
```

a pour effet de supprimer le fichier "fichier.txt" s'il existe.

- La fonction **`rename()`** permet de renommer un fichier. Cette fonction prend en paramètres une chaîne de caractères contenant l'ancien nom du fichier et une autre chaîne de caractères contenant le nouveau nom du fichier. Il faut également penser à vérifier l'existence du fichier avant de le renommer, sinon PHP génère une erreur.

Par exemple :

```
<?php


if(file_exists("fichier.txt")){
    rename("fichier.txt","nouveau_fichier.txt");
}


?>
```

a pour effet de renommer le fichier "fichier.txt" en "nouveau_fichier.txt" s'il existe.

- La fonction **`filesize()`** retourne la taille d'un fichier. Cette fonction prend en paramètre une chaîne de caractères contenant le nom du fichier.

Par exemple :

```
<?php


echo filesize("fichier.txt");


?>
```

Affiche :

8. Manipulation de répertoires

Ce chapitre utilise quelques notions de POO (programmation orientée objet). Le chapitre L'objet est donc un prérequis pour la bonne compréhension des syntaxes.

- La fonction **`dir()`** retourne une instance de la classe Directory. C'est-à-dire que cette fonction va mettre un pointeur sur un répertoire ce qui permettra ensuite de lire dans ce répertoire. Cette fonction prend en paramètre une chaîne de caractères contenant le chemin du répertoire.

Par exemple, si vous voulez lire les fichiers contenus dans le répertoire C:\Program Files\EasyPHP-DevServer-15.9VC11\data\localweb :

```
<?php


$directory = dir("/Program Files/EasyPHP-DevServer-15.9VC11\data\localweb");
echo "Pointeur : " . $directory->handle . "<br />";
echo "Chemin : " . $directory->path . "<br />";
while ($entry = $directory->read()) {
   echo $entry . "<br />";
}
$directory->close();


?>
```

Affiche :

```
Pointeur : Resource id #3
Chemin : /Program Files/EasyPHP-DevServer-15.9VC11\data\localweb
```

.

..

fichier.txt

test.php

En effet, la fonction **dir()** crée un objet `$directory` qui contient des propriétés handle et path ayant pour valeur le chemin du dossier. Nous verrons dans un prochain chapitre les notions d'objet et de propriété. Souvenez-vous pour l'instant qu'il existe une méthode **read()** permettant de lire les fichiers d'un répertoire.

- La fonction **is_dir()** retourne `true` si le répertoire existe et `false` sinon. Cette fonction prend en paramètre une chaîne de caractères contenant le nom du répertoire.

Par exemple :

```
<?php


if (is_dir("/Program Files/EasyPHP-DevServer-15.9VC11\data\localweb")) {
   echo "Le dossier existe.";
}
else {
   echo "Le dossier n'existe pas.";
}


?>
```

Affiche :

Le dossier existe.

- La fonction **opendir()** permet d'ouvrir un répertoire. Cette fonction prend en paramètre une chaîne de caractères contenant le nom du répertoire. Elle retourne un pointeur sur ce répertoire. Cette fonction est souvent associée à **readdir()** qui permet de lire tous les fichiers et les sous-répertoires d'un répertoire.

Par exemple :

```
<?php


if ($pointeur = opendir('.')) { //ouverture du répertoire courant
                                //(C:\Program Files\EasyPHP-DevServer-15.9VC11\
                                //data\localweb)
    while ($fichier = readdir($pointeur)) { //tant qu'il existe
                                            //un fichier dans le répertoire
        if ($fichier != "." && $fichier != "..") { //ne pas afficher les dossiers
```

```
                                                    //. et .. propres à Windows
            echo "$fichier <br />";
        }
    }
    closedir($pointeur);
}

?>
```

Affiche :

fichier.txt

test.php

C'est-à-dire le contenu du répertoire C:\Program Files\EasyPHP-DevServer-15.9VC11\data\localweb sans les dossiers **.** et **..** propres à Windows.

- La fonction **`filetype()`** retourne le type du contenu d'un répertoire, c'est-à-dire un fichier ou un sous-répertoire. Cette fonction prend en paramètre une chaîne de caractères contenant le nom du répertoire ou du fichier.

Par exemple, si dans notre répertoire localweb nous avons les deux fichiers test.php et fichier.txt et un sous-répertoire appelé image :

```
<?php

if ($pointeur = opendir('.')) { //ouverture du répertoire courant
                                //(C:\Program Files\EasyPHP-DevServer-15.9VC11\
                                //data\localweb)
    while ($fichier = readdir($pointeur)) { //tant qu'il existe
                                            //un fichier dans le répertoire
       if ($fichier != "." && $fichier != "..") { //ne pas afficher les dossiers
                                                  //. et .. propres à Windows
            echo $fichier." de type ".filetype($fichier)."<br />";
        }
    }
    closedir($pointeur);
}

?>
```

Affiche :

fichier.txt de type file

image de type dir

test.php de type file

- La fonction **`glob()`** retourne un tableau contenant tous les fichiers d'un répertoire respectant un masque donné. Cette fonction prend en paramètre une chaîne de caractères contenant le masque des fichiers à retourner.

Par exemple, si dans notre répertoire localweb nous avons les deux fichiers test.php et fichier.txt et un sous-répertoire appelé image :

```
<?php


$fichier = glob('./*.txt');
print_r($fichier);


?>
```

Affiche :

Array ([0] => ./fichier.txt)

En effet, le masque ./*.txt signifie : tous les fichiers texte du répertoire courant.

Si vous voulez rechercher tous les fichiers .jpg du répertoire image situé dans le répertoire courant, c'est-à-dire localweb, il faut taper le code suivant :

```
<?php


$fichier = glob('./image/*.jpg');
print_r($fichier);


?>
```

Contrairement à Windows, la fonction glob fait la distinction entre les minuscules et les majuscules donc si vous aviez des fichiers avec l'extension JPG, le code précédent ne les aurait pas trouvés.

- La fonction **`getcwd()`** retourne le répertoire courant.
- La fonction **`chdir()`** permet de se déplacer dans un répertoire.

Par exemple, si dans le répertoire localweb il y a les deux fichiers test.php et fichier.txt et un sous-répertoire appelé image :

```
<?php
```

```
echo getcwd()."<br />";
chdir('image');
echo getcwd();

?>
```

Affiche :

C:\Program Files\EasyPHP-DevServer-15.9VC11\data\localweb

C:\Program Files\EasyPHP-DevServer-15.9VC11\data\localweb\image

- La fonction **`mkdir()`** permet de créer un répertoire.

Par exemple, si vous voulez créer un répertoire css dans notre répertoire localweb :

```
<?php

echo getcwd()."<br />";
mkdir('css'); //création du répertoire css
chdir('css'); //déplacement dans le répertoire css
echo getcwd();

?>
```

Affiche :

C:\Program Files\EasyPHP-DevServer-15.9VC11\data\localweb

C:\Program Files\EasyPHP-DevServer-15.9VC11\data\localweb\css

- La fonction **`rmdir()`** permet de supprimer un répertoire.

Par exemple, si vous voulez supprimer le répertoire css dans le répertoire localweb :

```
<?php

echo getcwd()."<br />";
rmdir('css'); //suppression du répertoire css
echo "Suppression effectuée.";

?>
```

Affiche :

C:\Program Files\EasyPHP-DevServer-15.9VC11\data\localweb

Suppression effectuée.

- La fonction **`dirname()`** retourne le chemin parent du nom du répertoire, une chaîne de caractères passée en paramètre.

Par exemple :

```
<?php


$chemin = "/EasyPHP-DevServer-15.9VC11/data/localweb/image";
$repertoire = dirname($chemin);
echo $repertoire;


?>
```

Affiche :

/EasyPHP-DevServer-15.9VC11\data\localweb

- La fonction **`pathinfo()`** retourne un tableau contenant les informations sur le chemin d'une chaîne de caractères passée en paramètre.

Par exemple :

```
<?php


$tableau_info = pathinfo("/EasyPHP-DevServer-15.9VC11/data/localweb");
echo $tableau_info["dirname"]; //chemin parent
echo $tableau_info["basename"]; //répertoire
echo $tableau_info["filename"]; //fichier


?
```

Affiche :

/EasyPHP-DevServer-15.9VC11/data

localweb

localweb

Les includes

La fonction **`include()`** est très utile car elle permet d'appeler une autre page PHP dans une page PHP. Si vous marquez **`include('fonctions.php')`**, cela revient à coller le code contenu dans la page fonctions.php à l'endroit où vous appelez la fonction **`include()`**. Cette fonction reçoit donc en paramètre le nom de la page PHP à inclure.

Par exemple, vous avez une page PHP appelée variable.php contenant le code suivant :

```php
<?php


$nom = "DUPONT";

$prenom = "Robert";


?>
```

Dans votre page PHP courante, vous appelez l'include de cette façon :

```php
<?php


include("variable.php");

echo $prenom." ".$nom;


?>
```

Affiche :

Robert DUPONT

Si votre fichier variable.php se trouvait dans le répertoire inc, le code pour appeler cette page serait :

```php
<?php


include("inc/variable.php");

echo $prenom." ".$nom;


?>
```

La page PHP peut être indiquée avec un chemin relatif ou absolu. Un chemin absolu est marqué en partant de la racine du disque dur (par exemple C:\Program Files\easy PHP) alors qu'un chemin relatif est marqué en partant de l'endroit où se situe votre fichier PHP. Vous verrez dans un prochain chapitre que le fichier **php.ini** contient une directive **include_path** contenant le chemin de recherche pour les fichiers d'inclusion.

Le fichier à inclure peut porter l'extension inc et ainsi s'appeler variable.inc au lieu de variable.php. Il peut contenir aussi uniquement du HTML. Typiquement si vous avez un menu à insérer dans toutes vos pages de votre site web, vous créez un fichier menu.php contenant votre menu en HTML et vous faites un include("menu.php") dans toutes les pages de votre site web. Ceci est très important pour la maintenance de votre site, car si vous voulez apporter une modification à votre menu, vous ne le faites qu'une fois et ce sera visible partout.

- La fonction **`include_once()`** évite que l'inclusion soit répétée plusieurs fois. En effet il peut arriver que votre fichier include soit appelé involontairement dans un autre fichier include. Sa syntaxe est la même que pour la fonction **`include()`**.

- La fonction **`require()`** est semblable à la fonction **`include()`** à une différence près. La fonction **`require()`** provoque une erreur fatale si elle n'arrive pas à exécuter le code du fichier passé en paramètre. Alors que la fonction **`include()`** provoque uniquement un avertissement (warning). Sa syntaxe est la même que pour l'include, c'est-à-dire : **`require("page_includephp")`**
- La fonction **`require_once()`** évite que l'inclusion de type require soit répétée plusieurs fois.

Exercices sur les fichiers

1. Énoncés

Exercice 1 (facile) : création d'un fichier compteur de page

Créer un fichier texte qui stocke le nombre de fois qu'une page a été vue.

Exercice 2 (moyen) : création d'un fichier de renseignement d'images

Placer trois images dans un répertoire images puis créer une page PHP qui crée un fichier texte contenant le nom et la taille de ces images puis qui copie ces images dans un répertoire archive au même niveau que le répertoire image.

Exercice 3 (difficile) : création d'un fichier log de traces

Créer une page qui écrit dans un fichier log.txt la date et l'heure courante et qui affiche le temps en microsecondes pour déplacer trois images du répertoire images au répertoire archive.