## La notion de fonction

On appelle *fonction* un sous-programme qui permet d'effectuer un ensemble d'instructions par simple appel de la fonction dans le **corps** du programme principal. Les fonctions permettent d'exécuter dans plusieurs parties du programme une série d'instructions, cela permet une simplicité du code et donc une taille de programme minimale. D'autre part, une fonction peut faire appel à elle-même, on parle alors de fonction récursive (il ne faut pas oublier de mettre une condition de sortie au risque sinon de ne pas pouvoir arrêter le programme...).

## La déclaration d'une fonction

PHP recèle de nombreuses fonctions intégrées permettant d'effectuer des actions courantes. Toutefois, il est possible de définir des fonctions, dites *fonctions utilisateurs* afin de simplifier l'exécution de séries d'instructions répétitives. Contrairement à de nombreux autres langages, PHP nécessite que l'on définisse une fonction avant que celle-ci puisse être utilisée, car pour l'appeler dans le corps du programme il faut que l'interpréteur la connaisse, c'est-à-dire qu'il connaisse son nom, ses arguments et les instructions qu'elle contient. La définition d'une fonction s'appelle "*déclaration"* et peut se faire n'importe où dans le code. La déclaration d'une fonction se fait grâce au mot-clé *function*, selon la syntaxe suivante :

function Nom\_De\_La\_Fonction(argument1, argument2, ...) {  
 liste d'instructions  
}

**Remarques:**

* le nom de la fonction suit les mêmes règles que les [noms de variables](http://www.commentcamarche.net/contents/php/phpvar.php3) :
  + le nom doit commencer par une lettre
  + un nom de fonction peut comporter des lettres, des chiffres et les caractères \_ et & (les espaces ne sont pas autorisés!)
  + le nom de la fonction, comme celui des variables est sensible à la casse (différenciation entre les minuscules et majuscules)
* Les arguments sont facultatifs, mais s'il n'y a pas d'arguments, les parenthèses doivent rester présentes
* Il ne faut pas oublier de refermer les accolades
* Le nombre d'accolades ouvertes (fonction, boucles et autres structures) doit être égal au nombre d'accolades fermées!
* La même chose s'applique pour les parenthèses, les crochets ou les guillemets!

Une fois cette étape franchie, votre fonction ne s'exécutera pas tant que l'on ne fait pas appel à elle quelque part dans la page!

## Appel de fonction

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom (une fois de plus en respectant la casse) suivie d'une parenthèse ouverte (éventuellement des arguments) puis d'une parenthèse fermée :

Nom\_De\_La\_Fonction();

**Remarques:**

* le point virgule signifie la fin d'une instruction et permet à l'interpréteur de distinguer les différents blocs d'instructions
* si jamais vous avez défini des arguments dans la déclaration de la fonction, il faudra veiller à les inclure lors de l'appel de la fonction (le même nombre d'arguments séparés par des virgules!)
* Nom\_De\_La\_Fonction(argument1, argument2);

## Renvoi d'une valeur par une fonction

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

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

La syntaxe de l'instruction return est simple :

return valeur\_ou\_variable;

## Les arguments d'une fonction

Il est possible de passer des arguments à une fonction, c'est-à-dire lui fournir une valeur ou le nom d'une variable afin que la fonction puisse effectuer des opérations sur ces arguments ou bien grâce à ces arguments.

Le passage d'arguments à une fonction se fait au moyen d'une liste d'arguments (séparés par des virgules) entre parenthèses suivant immédiatement le nom de la fonction. Les arguments peuvent être de simple variables, mais aussi des tableaux ou des objets. A noter qu'il est possible de donner une valeur par défaut à ces arguments en faisant suivre le nom de la variable par le signe "=" puis la valeur que l'on affecte par défaut à la variable.

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

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

Ainsi cette fonction peut être appelée de deux façons différentes :

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

## Travailler sur des variables dans les fonctions

Lorsque vous manipulerez des variables dans des fonctions, il vous arrivera de constater que vous avez beau modifier la variable dans la fonction celle-ci retrouve sa valeur d'origine dès que l'on sort de la fonction...

Cela est du à la portée des variables, c'est-à-dire si elles ont été définies comme **variables globales ou locales**.

Il existe plusieurs niveaux de définition de variables :

* Une variable précédée du mot clé **global** sera visible dans l'ensemble du code, c'est-à-dire que sa portée ne sera pas limitée à la fonction seulement. Ainsi, toutes les fonctions pourront utiliser et modifier cette même variable
* Le niveau **static** permet de définir une variable locale à la fonction, qui persiste durant tout le temps d'exécution du script
* Par défaut, la variable possède le niveau **local**, c'est-à-dire que la variable ne sera modifiée qu'à l'intérieur de la fonction et retrouvera la valeur qu'elle avait juste avant l'appel de fonction à la sortie de celle-ci

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

## Passage de paramètre par référence

Une autre méthode pour modifier une variable consiste à la faire précéder du caractère *&*, précisant qu'il s'agit alors d'un alias: la valeur de la variable est modifiée à la sortie de la fonction. On parle alors de passage par référence. Dans ce cas on passe la référence (adresse mémoire) de la variable à la fonction, ce qui permet de modifier sa valeur.

<?  
function dire\_texte($qui, &$texte)  
{  
  
 $texte = "Bienvenue $qui";  
  
}  
  
$chaine = "Bonjour ";  
  
dire\_texte("cher phpeur",$chaine);  
  
echo $chaine; // affiche "Bienvenue cher phpeur"  
?>

## Retourner plusieurs variables

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

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

## La récursivité

Les fonctions récursives sont des fonctions qui s'appellent elles-mêmes. Ce type de fonction se révéle indispensable pour parcourir une arborescence par exemple.

Voici un exemple simple.

<?  
function fonction\_recursive($n=0)  
{  
  
 $n++;  
  
 echo "$n <br>";  
  
 if($n < 10){ // si n est inférieur à 10 on continue  
 fonction\_recursive($n);  
  
 }  
}  
  
fonction\_recursive(); // affiche les nb de 1 à 10  
?>