PHP 2° PARTIE :
FONCTIONS ET FORMULAIRE

1. Introduction
2. Syntaxe de déclaration
3. Utilisation des paramètres
4. Variables globales
5. Variables statiques
6. Récursivité
7. Fonctions dynamiques
8. Nombre variable de paramètres
9. Formulaire

1 . Introduction

On distingue 2 types de fonctions : les fonctions intégrées ou *built-in* qui sont inclues par défaut avec les distributions de PHP comme **`print`**, **`echo`** et les fonctions définies par le programmeur.

Les fonctions ont plusieurs buts :

- Éclaircir le code en regroupant dans une même fonction certaines fonctionnalités d'un programme qui se répètent.
- Pouvoir créer des fonctions génériques qui pourront être utilisées dans d'autre programme ce qui évite de répéter pour chaque projet le même code
- Possibilité d'évolution du code plus facile (modification du contenu d'une fonction)

2 . Syntaxe de déclaration

Une fonction se déclare et s'utilise de la manière suivante :

```
<?php
//déclaration
function ma_fonction($params1, $params2)
{
     // code de la fonction
     // …
     return($une_variable); // facultatif
}

$retour = ma_fonction(2, 5); //appel
?>
```

3 . Utilisation des paramètres (valeur ou référence)

Les paramètres d'une fonction peuvent être passés de 2 façons différentes :

- **Par valeur**, c'est a dire que s'ils ont une valeur à l'extérieur de la fonction, seule la valeur est transmise à la fonction, si la variable subit des modifications à l'intérieur de la fonction, ces modifications ne seront pas perçues dans le programme principal.

- **Par référence**, avec le signe & avant la variable (ex : `&$cpt`).Dans ce cas la, l'adresse mémoire de la variable dans le programme est passée à la fonction et toute modification de cette variable dans la fonction aura des répercutions à l'extérieur du programme.

3 . Utilisation des paramètres (exemple)

```
<?php
function modif_tab($tab)
{
          $tab[1] = "j'aime le C mais je préfère le php.";
}
$tab = array("salut,", "j'aime MySQL");

// passage de $tab par valeur, la boucle for affichera
// salut, j'aime MySQL

modif_tab($tab);
for ($i=0;$tab[$i];$i++)
          echo "$tab[$i] ";

// passage de $tab par référence, la boucle for affichera
// salut, j'aime le C mais je préfère le php.

modif_tab(&$tab);
for ($i=0;$tab[$i];$i++)
          echo "$tab[$i] ";

?>
```

4 . Variables globales

- **Les variables globales déclarées dans un script sont visibles sur l'ensemble du script, mais pas au sein des fonctions. Dans ce cas, pour utiliser une variable au sein d'une fonction, on la déclare comme globale, à l'aide de l'instruction `global`, pour faire référence à la variable globale du même nom.**

Rappel : les variables déclarées et utilisées au sein d'une fonction sont locales à la fonction.

- Une variable globale déclarée à l'intérieur d'une fonction permet à une variable d'être accessible en dehors de la fonction. Dans ce cas, la position de l'appel de la fonction est importante.

4 . Variables globales (exemple)

```
<?php
$a = 0; // a est une variable globale
$b = 1; // b est une variable globale
test_global($b); // passage par valeur
echo "a = " .$a. " b = " .$b. " c = " .$c;

function test_global($b) // b est ici une variable locale
{
      global $a; // permet l'accès à la variable globale a
      global $c; // c est une variable globale

      $a++;
      $c = 2;
      $b++; // c'est la variable locale qui est incrémentée
}
?>
```

5 . Variables statiques

- Une variable `static` déclarée à l'intérieur d'une fonction à l'aide de l'instruction **`static`** permet à une variable de garder sa valeur à chaque appel de la fonction. L'initialisation d'une variable `static` se fait au début de la fonction et à chaque appel de la fonction dans le script elle gardera la valeur du dernier appel.

```
<html><body>
<?php

function stat_fonc()
{
          static $cpt = 0;

          $cpt++;
          echo $cpt;
}

stat_fonc();
// affiche 1 (le premier appel initialise $cpt à 0 une seule fois
// et l'incrémente de 1

stat_fonc();
// affiche 2, $cpt a gardé la valeur précédente et l'incremente de 1.
?>
</body></html>
```

6 . Récursivité

Le langage PHP supporte les fonctions récursives. Une fonction récursive est une fonction qui s'appelle elle-même.

Exemple simple présentant le principe de la récursivité : Affichage à l'envers d'une chaîne de caractère

```
<html><body>
<?php
$str = "Hello World !";
reverse_r($str);
function reverse_r($str)
{
    if(strlen($str) > 0)
        //substr retourne le premier caractère
      reverse_r(substr($str, 1)); //appel récursif
    echo substr($str, 0, 1); //affiche un caractère
    return;
}
?>
</body></html>
```

6 . Récursivité (suite)

- Les fonctions récursives sont principalement utilisées pour naviguer dans les structures de données dynamiques (listes et arbres).

- Sinon, dans de nombreux cas, la récursivité est équivalente à une répétition (itération). Les fonctions récursives sont plus lentes et consomment plus de mémoire que les itérations.

- Le même exemple réalisé par itération :

```
function reverse_i($str)
{
  for($i=1;$i<=strlen($str);$i++)
     echo substr($str, -$i, 1); //affiche un caractère en partant de la fin
  return;
}
```

7 . Fonctions dynamiques

Vous pouvez vous trouver dans le cas ou vous ne savez pas quelle fonction devra être appelée à un moment précis du script.
Pour cela, il suffit de placer dans une variable le nom d'une fonction, puis d'utiliser cette variable comme une fonction.

Un exemple :

```
<?php
function write($text)
{
        print($text);
}

function writeBold($text)
{
        print("<b>$text</b>");
}

$fonction_var = "write";
$fonction_var("toto"); // affiche toto

$fonction_var = "writeBold";
$fonction_var("toto"); // affiche toto en gras
?>
```

8 . Nombre variable de paramètres

Il existe 3 fonctions utiles dans la gestions des arguments passés à une fonction dans php4. Il s'agit de :

- **`func_get_arg`** qui permet de lire un argument spécifique.
- **`func_get_args`** pour obtenir l'ensemble des arguments sous forme d'un tableau.
- **`func_num_args`** pour connaître le nombre d'arguments reçus par la fonction.

Cela permet d'écrire des fonctions qui acceptent un nombre variable d'arguments.

```
<?php
function foo()
{ $numargs = func_num_args();
  echo "Nombre d'arguments: $numargs\n";
}
foo (1, 2, 3); // affiche 'Nombre d'arguments: 3'
?>
```

9 . Formulaire

Pour utiliser un formulaire capable d'envoyer des informations à un script php, il suffit de mettre le nom du fichier php qui réceptionnera les informations en tant que valeur de l'attribut *ACTION* de la balise *FORM*.

Voici ce à quoi peut ressembler un formulaire en HTML, permettant d'envoyer les coordonnées d'une personne à un fichier php nommé *test.php* :

```
<FORM method="GET" action="test.php">
Nom :                <INPUT type=text size=20 name=nom><BR>
Prénom :             <INPUT type=text size=20 name=prenom><BR>
Age :                <INPUT type=text size=2 name=age><BR>
                     <INPUT type=submit value=Envoyer>
</FORM>
```

Lorsque l'on soumet un formulaire à un fichier php, toutes les données du formulaires lui sont passées en tant que variables, c'est-à-dire chacun des noms associés aux champs (ou boutons) du formulaires précédés du caractère *$*.

Ainsi, dans l'exemple précédent, le fichier *test.php* reçoit les variables :

$nom **$prenom** **$age**

Si jamais un des champs du formulaire n'est pas rempli, il possède la valeur "", c'est-à-dire une chaîne vide.

On utilisera généralement les fonctions **`IsSet()`** ou **`Empty()`** pour faire des tests sur les variables.

Notes