Matière PHP

Les bases

Les pages web contenant du PHP ont l'extension .php.
Une page PHP est une simple page HTML qui contient des instructions en langage PHP.
Les instructions PHP sont placées dans une balise **<?php ?>**.
Pour afficher du texte en PHP, on utilise l'instruction **echo**.

Une page PHP peut inclure une autre page (ou un morceau de page) grâce à l'instruction include : **<?php include("maPage.php"); ?>**. Cette instruction ordonne à l'ordinateur : « Insère ici le contenu de la page maPage.php ». En gros c'est le chemin où se trouve le fichier. L'instruction include sera remplacée par le contenu de la page demandée. Cette technique, très simple à mettre en place, permet par exemple de placer les menus de son site dans un fichier menus.php que l'on inclura dans toutes les pages. Cela permet de centraliser le code des menus alors qu'on était auparavant obligé de le copier dans chaque page sur nos sites statiques en HTML et CSS !

Les variables

$ devant une chaîne de caractères signifie que nous parlons d'une variable, on la nomme comme on veut ensuite, mais jamais d'accent ni d'espace dans un nom de variable.
Une variable est typée : elle peut prendre 4 types (dans l'ordre ci-dessous) :
Les types des variables
- string (chaîne de caractères)
- integer (nombre entier)
- bool (booléen qui signifie logique binaire genre vrai ou faux)
- float (nombre réel)

Toutes les instructions (toutes) se terminent par un point-virgule.
• Seule la variable string exige les guillemets (simples ou doubles, préférez les apostrophes pour l'instant, nous verrons pourquoi un peu plus loin)...
• La booléenne prend la valeur true, mais sans guillemets, elle a pour contraire... false.
• Le nombre réel prend un point à l'anglo-saxonne et non une virgule...

Une chaîne est une suite de caractères. Elle peut être délimitée par des guillemets simples ou doubles. **Attention**, leur utilisation ne produira pas le même effet: les chaines entourées de guillemets doubles sont sujettes aux substitutions de variables et au traitement des séquences d'échappement, contrairement aux chaines entourées de guillemets simples.

```
<?
$chaine = "(mais pas à CS)";
echo "Je joue à Quake 3 $chaine\tQ3 rocks!";
echo 'Je joue à Quake 3 $chaine\tQ3 rocks!';
?>
```

Le premier echo va afficher "Je joue à Quake 3 (mais pas à CS) Q3 rocks!",

tandis que le deuxième va afficher "Je joue à Quake 3 $chaine\tQ3 rocks!".

Opérateurs arithmétiques

`$a + $b`	Addition	Somme de $a et $b.
`$a - $b`	Soustraction	Reste de la différence de $b et $a.
`$a * $b`	Multiplication	Produit de $a par $b.
`$a / $b`	Division	Dividende de $a par $b.
`$a % $b`	Modulo	Reste de la division entière de $a par $b.

Operateur sur les chaines

Il n'existe qu'un seul opérateur en PHP sur les chaînes, c'est l'opérateur de concaténation. Cependant PHP dispose en standard de toutes les fonctions nécessaires au maniement des chaînes de caractères.

```
$a = "Hello ";
$b = $a . "World!"; // maintenant $b contient "Hello World!";
```

Il est à noter que vous pouvez spécifier une chaîne en l'encadrant entre des simples ou des doubles quottes. Dans le second cas, PHP tentera d'interpréter le contenu de la chaîne. La différence au niveau performance existe, mais est mineure.

```
$val='Pengo';
echo 'Hello $val';   // Affiche  Hello $val
echo "Hello $val";   // Affiche  Hello Pengo
```

L'utilisation de l'opérateur arithmétique d'addition en vue de concaténer deux chaînes de caractères **est interdite**.

Opérateurs logique

`$a and $b`	Et	Résultat vrai si $a ET $b sont vrais
`$a or $b`	Ou	Résultat vrai si $a OU $b est vrai (ou les deux)
`$a xor $b`	Ou exclusif	Résultat vrai si $a OU $b est vrai, mais pas si les deux sont vrais
`! $a`	Non	Résultat vrai si $a est faux, et réciproquement
`$a && $b`	Et	Résultat vrai si $a ET $b sont vrais
`$a || $b`	Ou	Résultat vrai si $a OU $b est vrai (ou les deux)

Opérateurs de comparaison

`$a == $b`	égal	Résultat vrai si $a est égal à $b
`$a != $b`	Différent	Résultat vrai si $a est différent de $b
`$a < $b`	Inférieur	Résultat vrai si $a est strictement inférieur à $b
`$a > $b`	Supérieur	Résultat vrai si $a est strictement supérieur à $b
`$a <= $b`	Inf ou égal	Résultat vrai si $a est inférieur ou égal à $b
`$a >= $b`	Sup ou égal	Résultat vrai si $a est supérieur ou égal à $b

Les conditions IF et SWITCH

L'instruction if est une des structures de contrôle de base que l'on retrouve dans la plupart des langages.

```
If (expression){
    Instructions
}

If(expression){
    Instructions
} else {
    Instructions
}

if(expression) {
    instructions
}
elseif(expression) {
    instructions
}
else {
    instructions
}
```

Notez que dans la première syntaxe, on peut se passer des accolades s'il n'y a qu'une seule instruction à exécuter.

L'instruction switch peut élégamment remplacer une longue série de if peu lisibles. Voici les deux syntaxes:

```
switch(expression) {
case expression:
   instructions
   break;
default:
   instructions
   break;
}
```

L'expression de chaque instruction case est comparée à l'expression du switch. Si elles sont égales, le bloc d'instructions suivant le case est exécuté. Le mot-clef break permet de sortir du switch. Sans lui, toutes les exécutions suivant le case concerné (y compris celles des case suivants) s'exécuteraient. Si aucun des case ne correspond à l'expression switch, c'est le bloc d'instructions suivant default qui sera exécuté.

```
<?
$i = 3;

switch($i)
{
   case 2:  echo "La variable i est égale à 2";  echo "<br>le php ressemble au C";  break;
   case 3:  echo "La variable i est égale à 3";  break;
   default:  echo "La variable i n'est égale ni à 2, ni à 3";  break;
}
?>
```

Remarquez que, malgré son omniprésence, break n'est pas requis par l'instruction switch. En effet, on peut très sciemment s'en passer. On pourrait très bien contracter ce code en:

```
<?
$i = 3;

switch($i)
{
   case 3:  echo "La variable i est égale à 3";  break;
   case 2:  echo "La variable i est égale à 2";
   default:  echo "<br>le php ressemble au C";  break;
}
?>
```

La boucle While

L'instruction while permet de répéter le bloc d'instructions qu'il contient aussi longtemps qu'une expression est vraie. Voici les deux syntaxes:

```
while(expression) {
   instructions
```

```
}
```

```
while(expression):
   instructions
endwhile;
```

De la même façon que pour le if, la première syntaxe peut se passer des accolades s'il n'y a qu'une instruction à exécuter. L'expression while est évaluée avant chaque itération (une itération = un tour de boucle). Ce qui veut dire que si votre expression est fausse, le programme ne passera même pas une fois dans votre boucle. Si l'expression est vraie, le code sera exécuté, puis l'expression réévaluée. Si elle se révèle vraie une fois de plus, le code sera exécuté une seconde fois, etc, jusqu'à ce que l'expression soit fausse. Si l'expression est fausse, l'exécution du programme reprend immédiatement après la fermeture de la boucle.

```
<?
$i = 0;

while( $i < 10 )  // tant que $i est plus petit que 10
{
   echo "La variable i est égale à $i<br>";
   $i++;       // on incrémente $i
}
?>
```

Dans ce script on met une variable $i à 0, puis, dans un while, on l'incrémente après avoir affiché sa valeur. La condition d'accomplissement de la boucle est que $i soit strictement inférieur à 10.

Les boucles while sont souvent sources d'erreurs aux conséquences assez catastrophiques. Afin de minimiser ce risque, prenez garde aux choses suivantes:

- N'oubliez pas d'initialiser toute variable faisant partie de l'expression while, sous peine de résultats aléatoires...
- De même, si vous faîtes une boucle telle que celle ci-dessus, n'oubliez pas de faire varier la variable qui fait partie de l'expression. Vous risquez sinon de créer une boucle infinie.

La boucle for

Une boucle for, c'est un peu comme un while, mais en automatique. Voici les deux syntaxes:

```
for(expr_de_départ; expr_conditionnelle; expr_itérative) {
   instructions
}
                  for(expr_de_départ; expr_conditionnelle; expr_itérative):
   instructions
endfor;
```

Une boucle for contient trois expressions. La première est l'expression de démarrage: elle sera évaluée une et une seule fois au début de la boucle. On l'utilise en général pour initialiser un compteur. La seconde expression est l'expression conditionnelle. L'expression est évaluée avant chaque itération, et, si elle est vraie, le code contenu dans la boucle est

exécuté. Sinon l'exécution du programme se poursuit à partir de l'instruction suivant la fermeture de la boucle. La troisième et dernière expression est l'expression itérative. Elle est évaluée à la fin de chaque itération. On l'utilise en général pour incrémenter le compteur qu'on a initialisé dans l'expression de démarrage. Dans la première syntaxe, on peut se passer des accolades si la boucle for ne contient qu'une instruction. Voici le même script que précédemment, cette fois avec un boucle for:

```
<?
for($i = 0; $i < 10; $i++)
{
    echo "La variable i est égale à $i<br>";
}
?>
```

Même si ça parait moins clair de prime abord, on s'aperçoit vite que c'est bien plus pratique et lisible qu'un while.

Les Tableaux

Ces tableaux sont très simples à imaginer. Regardez par exemple celui-ci, contenu de la variable $prenoms :

Clé Valeur
0 François
1 Michel
2 Nicole
3 Véronique
4 Benoît
… …

Construire un tableau numéroté

Pour créer un tableau numéroté en PHP, on utilise généralement la fonction array.
Cet exemple vous montre comment créer l'array $prenoms :

```
// La fonction array permet de créer un array
<?php
        $prenoms = array ('François', 'Michel', 'Nicole', 'Véronique','Benoît');
?>
```

L'ordre a beaucoup d'importance. Le premier élément (« François ») aura le n°0, ensuite Michel le n°1, etc.

Afficher un tableau numéroté

Pour afficher un élément, il faut donner sa position entre crochets après $prenoms. Cela revient à dire à PHP : « Affiche-moi le contenu de la case n°1 de $prenoms. » Pour faire cela en PHP, il faut écrire le nom de la variable, suivi du numéro entre crochets.
Pour afficher « Michel », on doit donc écrire :

```
<?php
        echo $prenoms[1];
?>
```

Un tableau, c'est un moyen de stocker plusieurs variables, selon un plan qui vous paraît logique. C'est comparable à un meuble avec ses tiroirs. Dans le tiroir 0, (oui, le tableau commence par le tiroir zéro), vous rangez la variable lundi par exemple, dans le tiroir 1, vous rangez la variable mardi etc. En informatique, on appelle **index ou indice** le numéro de tiroir (**la position de la variable dans le tableau**), et **valeur** la **valeur de la variable entreposée**.
Voici une autre exemple de la syntaxe d'un tableau tout simple

On construit le tableau des jours de la semaine

```php
<?php
$semaine=array('lundi','mardi','mercredi','jeudi','vendredi','samedi','dimanche');
?>
```

Par cette simple ligne, vous venez de construire un tableau (qui vous le remarquerez, est une variable en lui-même puisqu'il commence par $, mais une variable complexe, organisée). **$semaine est le nom du tableau entier.** Vous lui avez affecté des valeurs, (via la **commande array**, il sait que c'est un tableau). Et ici, par défaut, **l'index commence à 0**, donc dimanche aura pour index.. 6 et non 7. Une fois construit ce tableau, comment convoquer une valeur ? **$semaine[2] sera...mercredi** et ainsi de suite... selon la règle : **$semaine[index]**

Bien sûr ici, nous avons construit **un tableau contenant des variables string** (les jours de la semaine) **et des indices numériques** (0, 1, 2 etc...).
On peut tout-à-fait ranger des **valeurs numériques dans un tableau**.
Ainsi ce tableau qui stocke quelques années marquantes de l'histoire de France...

Histoire de France

```php
<?php
$dates=array(1789,1830,1848,1851,1871,1914,1918,1936,1939,1945,1958,1968);
echo $dates[3];
?>
```

La commande echo renverra... 1851 ici.

Les tableaux associatifs

Ils fonctionnent sur le même principe, sauf qu'au lieu de numéroter les cases, on va les étiqueter en leur donnant à chacune un nom différent. Par exemple, supposons que je veuille, dans un seul array, enregistrer les coordonnées de quelqu'un (nom, prénom, adresse, ville, etc.). Si l'array est numéroté, comment savoir que le n°0 est le nom, le n°1 le prénom, le n°2 l'adresse… ? C'est là que les tableaux associatifs deviennent utiles.

Construire un tableau associatif

Pour en créer un, on utilisera la fonction array comme tout à l'heure, mais on va mettre « l'étiquette » devant chaque information :

```php
<?php
// On crée notre array $coordonnees
    $coordonnees = array (
        'prenom' => 'François',
        'nom' => 'Dupont',
        'adresse' => '3 Rue du Paradis',
        'ville' => 'Marseille');
?>
```

Vous remarquez qu'on écrit une flèche (=>) pour dire « associé à ». Par exemple, on dit « *ville*» associée à « *Marseille* ».Nous avons créé un tableau qui ressemble à la structure suivante :

Clé Valeur
Prenom François
nom Dupont
adresse 3 Rue du Paradis
ville Marseille

Afficher un tableau associatif

Pour afficher un élément, il suffit d'indiquer le nom de cet élément entre crochets, ainsi qu'entre guillemets ou apostrophes puisque l'étiquette du tableau associatif est un texte. Par exemple, pour extraire la ville, on devra taper :

```php
<?php
    echo $coordonnees['ville'];
?>
```

Ce code affiche : « Marseille ».

Quand utiliser un array numéroté et quand utiliser un array associatif ?

Comme vous l'avez vu dans mes exemples, ils ne servent pas à stocker la même chose…

- Les arrays numérotés permettent de stocker une série d'éléments du même type, comme des prénoms. Chaque élément du tableau contiendra alors un prénom.
- Les arrays associatifs permettent de découper une donnée en plusieurs sous-éléments. Par exemple, une adresse peut être découpée en nom, prénom, nom de rue, ville…

Parcourir un tableau

Lorsqu'un tableau a été créé, on a souvent besoin de le parcourir pour savoir ce qu'il contient. Nous allons voir un des moyens d'explorer un array :

- la boucle for ;

La boucle for

Il est simple de parcourir un tableau numéroté avec la boucle for. Puisqu'il est numéroté à partir de 0, on peut faire une boucle for qui incrémente un compteur à partir de 0 :

```php
<?php
// On crée notre array $prenoms
	$prenoms = array ('François', 'Michel', 'Nicole', 'Véronique','Benoît');
// Puis on fait une boucle pour tout afficher :
	for ($numero = 0; $numero < 5; $numero++)
	{
		echo $prenoms[$numero] . '<br />'; // affichera $prenoms[0],
			$prenoms[1] etc.
	}
?>
```

Quand on écrit $prenoms[$numero], la variable $numero est d'abord remplacée par sa valeur. Par exemple, si $numero vaut 2, alors cela signifie qu'on cherche à obtenir ce que contient $prenoms[2], c'est-à-dire… Nicole.

Les fonctions

Les fonctions permettent d'éviter d'avoir à répéter du code PHP que l'on utilise souvent. Une fonction est un bloc de code PHP destiné généralement à être réutilisé plusieurs fois. Plutôt que d'écrire X fois le morceau de code, on le met dans une fonction, et c'est cette fonction que l'on appellera dès qu'on l'aura décidé. Une fonction est une série d'instructions qui effectue des actions et qui retourne une valeur. En général, dès que vous avez besoin d'effectuer des opérations un peu longues dont vous aurez à nouveau besoin plus tard, il est conseillé de vérifier s'il n'existe pas déjà une fonction qui fait cela pour vous. Et si la fonction n'existe pas, vous avez la possibilité de la créer.

```php
<?php
	function Addition($nb1, $nb2){
		$somme=$nb1 + $nb2;
		echo ' La somme est de : '.$somme;
	}
?>
```

Dans ce cas, pour la convoquer, on pourra écrire :

```
<?php
        Addition(5 , 10) ;
?>
```

Programmation Orientée Objet

PHP dispose des concepts de POO (Programmation Orientée Objet) au travers des classes. Rappelons d'abord qu'un objet possède des attributs et des méthodes, et doit utiliser les mécanismes d'héritage et de polymorphisme.
Attribut = caractéristique d'un objet.
Méthode = action qui s'applique à un objet.
Héritage = définition d'un objet comme appartenant à la même famille qu'un autre objet plus général, dont il hérite des attributs et des méthodes.

Les classes

Une classe est la description complète d'un objet. Elle comprend la déclaration des attributs ainsi que l'implémentation des méthodes de cet objet. La création d'un objet est déclenchée par celle d'une instance de la classe qui le décrit.

Déclaration

La déclaration d'une classe s'appuie sur le mot clé class. La syntaxe est comparable à celle de la déclaration des fonctions.

```
class ma_classe { … }
```

Affectation

Pour exploiter les méthodes et les propriétés d'un objet, on utilise un accesseur dont la syntaxe est constituée des caractères « - » et « > » côte à côte : « -> »

```
$objet_test -> ma_méthode() ; // appelle la méthode
```

Opérateur de la classe courante

$this-> est l'opérateur de self-référence.

```
$this -> nb_roues = 4 ; Les méthodes se déclarent comme des fonctions.
```

Constructeur

Le constructeur se déclare comme une méthode. Il doit porter le nom de la classe. Il est appelé automatiquement lors de l'instanciation de la classe. ($uneClass = New MaClasse();)

```
class Véhicule {
        var $nb_roues;
        function Véhicule( $nb_roues ) { $this-> nb_roues= $nb_roues; }
```