LES BASES DU LANGAGE PHP

Ajouter du code PHP dans un fichier sur le serveur

Le code PHP est exécuté sur le serveur, et n'est pas transmis au navigateur.

Par convention, les fichiers qui contiennent du PHP auront l'extension ".php".

Le code php est encadré par les balises

```
<?php
    // Votre code PHP ici
?>
```

La balise finale (celle en fin de fichier) est facultative.

Tout ce qui n'est pas entre ces 2 balises est envoyé au navigateur tel quel (donc traité comme du HTML)

Dans le code PHP, l'instruction "echo" permet de générer du code qui s'inclut dans le flux HTML.

Par exemple, le fichier suivant :

```
<p>Paragraphe
<?php echo "important"; ?>
</p>
```

Envoie au navigateur la texte suivant :

```
<p>Paragraphe
important
</p>
```

Il existe une alternative simplifiée : **<?=** *expression* **?>**

Qui est équivalente à **<?php echo** *expression* **; ?>**

L'exemple ci-dessus aurait donc pu être noté :

```
<p>Paragraphe
<?= "important" ?>
</p>
```

La syntaxe PHP

PHP est un langage de programmation : il s'agit donc de donner des ordres, des consignes à la machine (en l'occurrence le serveur).

Alors que HTML est un langage de description de contenu (avec des balises pour le structurer), et CSS un langage de description de mise en forme.

La syntaxe générale est proche de celle de javascript.

Un programme est donc un mode opératoire, il est composé d'**instructions**.

Il y a quelques éléments généraux à connaître :

Commentaires : on peut ajouter des commentaires (donc des informations qui ne sont utiles que pour les programmeurs, la machine n'en fait rien). Il y a deux manière de mettre un commentaire :

- le commencer par **//** : tout ce qui est après le //, jusqu'à la fin de la ligne est un commentaire
- l'encadrer par **/* */** (comme en CSS) : tout ce qui est entre les 2 marqueurs est du commentaire (cela peut donc s'étendre sur plusieurs lignes/

Séparateurs : comme en HTML et en CSS, les sauts de lignes ne sont pas significatifs (sauf en fin de commentaire simple).

Pour terminer les instructions, on utilisera donc le **point-virgule** (comme en CSS).

Par contre pour des raisons de lisibilité, on ne placera en général qu'une instruction par ligne.

Il existe également des situations ou on doit créer des blocs d'instructions : on les encadre alors par **{ }** (la notion est assez similaire aux { } en CSS).

Par convention et lisibilité, on indente vers la droite tout ce qui se trouve dans un bloc.

Exemple :

```
/*
Ceci est une exemple de mise en forme
 */

var $nbPersonnes = 4 ;        // Nombre de personnes
if ( $x > $nbPersonnes) {
   echo "Il y a trop de monde" ;
   $flag = true ;
}
```

Les variables

En programmation, une variable est un moyen de stocker et de nommer une information pour l'utiliser dans les instructions. C'est donc une sorte de boite, à laquelle on donne un nom, et dans laquelle on peut ranger quelque chose.

Une variable doit être nommée.

NOM DE VARIABLE

Un nom de variable commence par le caractère **$**, puis une lettre (majuscule ou minuscule) ou un underscore (_), suivi d'autant de lettres, underscores et chiffres que l'on veut (on ne peut pas commencer par un chiffre).

Tous les autres caractères sont exclus.

La différence avec javascript est donc le $.

Par convention, et pour des raisons de lisibilité :

- on donne des noms significatifs (un nom et un adjectif, par exemple *menuPrincipal*)
- on met la première lettre des mots (sauf le premier) en majuscules, et tout le reste en minuscules
- on évite les chiffres, sauf quand 2 veut dire explicitement "le second", par exemple

AFFECTAION D'UNE VALEUR À UNE VARIABLE

Une variable n'a pas besoin d'être déclarée, mais il faut lui affecter une valeur avant de l'utiliser :

```
$nomUtilisateur = "Durand" ;
```

Le signe **égal** après un nom de variable est une instruction d'affectation. Cela remplace le contenu de la variable dont le nom est à gauche ($*nomUtilisateur* dans l'exemple) par le résultat de l'expression qui est à droite (la chaîne de caractère *"Durand"* dans l'exemple).

Expressions

Une **expression** est une formule qui donne un résultat.

Cela peut donc être un nombre, une chaîne de caractères, une variable (dont on utilise alors le contenu), un appel de fonction (dont on prend le retour, le résultat, voir le paragraphe sur les fonctions), et une combinaison d'opérations sur tous ces éléments.

Les types de valeurs utilisables dans les expressions (donc les types de valeurs pour les variables) peuvent être :

- un nombre entier : 4
- un nombre décimal : 5.5 (le séparateur est le point, c'est la notation anglaise)
- une chaîne de caractères, c'est à dire du texte : "Virginie"

  Pour que PHP comprenne que ce n'est pas un mot clé du langage ou un élément créé, mais le texte, le mot, on entoure par des guillemets simples ou doubles
- un booléen (vrai ou faux) : **true** ou **false**

OPÉRATIONS SUR LES NOMBRES

Il est possible de faire des opérations sur les nombres, en particulier :

Somme :	a **+** b	
Différence :	a **-** b	
Produit (multiplication) :	a * b	(le signe "multiplier" est l'étoile)
Division :	a / b	

Bien sûr il est possible de combiner tout cela, et d'affecter le résultat à une variable.

Les parenthèse permettre de regrouper des opérations.

```
$total =  ( $prixArticle1 * 2 + $prixArticle2 * 3 ) * 1.2;
```

OPÉRATIONS SUR LES CHAINES DE CARACTÈRES :

Sur les chaines de caractères l'opérateur . est un opérateur de **concaténation** (on met les chaines bout à bout).

Exemple :

```
$message =  "Bienvenue " . $prenomUtilisateur ; // message contient alors "Bienvenue Virginie"
```

Fonctions

Les fonctions permette de réaliser des "morceaux" de programme, qui vont réaliser une action donnée, à partir de données en entrée (les paramètres).

Une fonction comporte donc les caractéristiques suivantes :

- un **rôle** (ce qu'elle fait)
- des **paramètres** : les données que l'on doit lui donner pour qu'elle fasse l'action dans la bonne situation, avec les bonnes informations
- un **retour** (facultatif) : une valeur qu'elle renvoie

APPEL D'UNE FONCTION

Pour utiliser une fonction (on dit appeler une fonction), on doit connaître son nom, ces paramètres, et ce qu'elle retourne.

Par exemple, pour la fonction intégrée **print_r**:

- Le rôle est d'afficher en clair le résultat d'un expression ou le contenu d'une variable
- Elle a un paramétre : l'expression
- Elle ne retourne rien

Pour l'utiliser :

```
print_r("Bonjour") ;
```

On donne le nom de la fonction, suivi d'une parenthèse, de la valeur de chaque paramètre (séparés par une virgule), puis on ferme la parenthèse. La valeur de chaque paramètre est une **expression**.

```
print_r( $prenomUtilisateur ) ;
print_r( "Bonjour" .  $prenomUtilisateur) ;
```

Si la fonction retourne une valeur, on l'utilise dans une expression :

```
$z = somme( 4, $b ) ;
$x = somme($a,2) + somme($b,7) ;
print_r ( somme(4,$b) ) ;
```

FONCTIONS UTILISATEUR

On peut utiliser les multiples fonctions intégrées, ou des fonctions fournies par des librairies, mais on peut aussi fabriquer ses propres fonctions.

Pour réaliser une fonction, on doit lui choisir un **nom** : ce sont les même règles que pour les variables (sans le $)

Puis définir son rôle, ses paramètres (ce dont elle a besoin pour faire le traitement demandé avec les bonnes valeurs, dans le bon contexte), et son retour (ce qu'elle renvoie si elle doit renvoyer une valeur)

Pour l'ajouter dans le programme (donc la créer pour pouvoir l'utiliser a), on utilise le mot-clé **function** :

```
function nomFonction( $nomParametre1, $nomParametre2 )  {
   // Role : on explicite le rôle de la fonction
   // Paramètres :
   //    $nomParametre1 : ce qu'il faut mettre en premier paramètre quand on appelle la fonction
   //    $nomParametre2 : ce qu'il faut mettre en second paramètre quand on appelle la fonction
   // Retour : ce que retourne la fonction

   // Code de la fonction ici
   // .....
}
```

Le code qui permet à la machine d'exécuter la fonction est mis entre accolades (c'est un bloc de code).

Attention : ce code ne sera exécuté que lors de l'appel de la fonction, et à chaque appel de la fonction.

Les noms donnés aux paramètres sont les noms de variable **dans la fonction** qui reçoivent la valeur des paramètres.

Dans le code de la fonction, pour mettre fin à la fonction (en sortir, ne pas exécuter la suite) on utilise l'instruction **return**, suivie d'une expression si on doit retourner une valeur ;

Exemple : déclaration d'une fonction qui calcule un prix TTC :

```
   function prixTTC( $prixHT, $tauxTVA )  {
      // Role : calcule et retourne le prix TTC à partir du prix hors taxes et du taux de TVA
      // Paramètres :
      //    $prixHT : prix hors taxes
      //    $tauxTVA: taux de TVA (en%)
      // Retour : prix TTC

      return $prixHT * ( 1 + $tauxTVA / 100) ;
   }
```

Une fonction qui retourne une valeur peut être utilisée dans une expression.

VARIABLES LOCALES

Les paramètres d'une fonction, ainsi que les variables utilisées dans les fonctions, sont locales : cela signifie qu'elles n'existent qu'à l'intérieur de la fonction, et n'existent plus quand on sort de la fonction (et si on appelle plusieurs fois la fonction, c'est une nouvelle variable à chaque fois.

Les expressions conditionnelles : if

L'instruction **if** permet d'exécuter des blocs de code en fonction de conditions.

La syntaxe est donc :

```
if ( condition1 ) {
   // Code exécuté si la condition 1 est vraie
} else if ( condition2 ) {
   // Code exécuté si la condition 2 est vraie
} else {
   // Code exécuté si aucune des conditions n'est vraie
}
```

Les clauses **else if** et **else** sont facultatives.

Il y a au maximum un **else**, mais autant de **else if** que l'on souhaite.

Les tests peuvent être imbriqués, sans limitation : un bloc de code peut contenir des if.

LES CONDITIONS

Une condition est une expression qui après évaluation, vaut **true** (vrai) ou **false** (faux).

On peut donc utiliser des opérateurs de comparaisons :

expression1 **===** *expression2* : le résultat de l'expression 1 est strictement égal au résultat de l'expression 2

exemple : `( 1 + 1 === 2 )` vaut **true**, par contre `( 1 + 1 === "2" )` vaut **false**

expression1 **!==** *expression2* : le résultat de l'expression 1 est différent du résultat de l'expression 2

expression1 **>** *expression2* : le résultat de l'expression 1 est strictement supérieur au résultat de l'expression 2

expression1 **>=** *expression2* : le résultat de l'expression 1 est supérieur ou égal au résultat de l'expression 2

expression1 **<** *expression2* : le résultat de l'expression 1 strictement inférieur au résultat de l'expression 2

expression1 **<=** *expression2* : le résultat de l'expression 1 est est inférieur ou égal au résultat de l'expression 2

Et les conditions peuvent être combinées :

`(` *condition1* **&&** *condition2* `)` est vrai si les deux conditions sont vraies

`(` *condition1* **and** *condition2* `)` est vrai si les deux conditions sont vraies

`(` *condition1* **||** *condition2* `)` est vrai si au moins une des deux conditions sont vraies

`(` *condition1* **or** *condition2* `)` est vrai si au moins une des deux conditions sont vraies

`(` **!** *condition* `)` est vrai si la condition est fausse

Sites de référence

Le site de référence pour PHP est : **php.net**

Il est très complet, et indique la compatibilité avec les navigateurs.

Je conseille la version en anglais pour ceux qui sont à l'aise avec cette langue.