**PHP**

PHP : Personal Home Page -> PHP : Hypertext Preprocessor

**PHP, un langage pour le Web, mais pas seulement …**

PHP est avant tout un langage dédié à la génération de pages Web. Le client (un navigateur par exemple) demande l'exécution d'une "page PHP" (en fournissant d'éventuels paramètres) à un serveur, ce dernier sous-traite la demande à un processeur PHP qui interprète le code et retourne un résultat au serveur qui le renvoie au client.

PHP peut cependant être utilisé en ligne de commande, au même titre que d'autres langages de programmation.

Nous utiliserons PHP en mode *interaction directe avec l'utilisateur* (ligne de commande) et non en mode *client/serveur Web* (via HTTP).

**Début <?php Fin ?>**

Voici un programme PHP qui affiche "Bonjour." (avec un retour à la ligne)

<?php

print("Bonjour.\n");

?>

**Affichage print ou echo**

On peut inclure des caractères spéciaux (retours chariot, des sauts de ligne, des tabulations...) avec le caractère d'échappement antislash :

print("Bonjour.\n"); // \n permet un saut de ligne

**Commentaires // ou # ou /\* \*/**

Un commentaire est un bout de texte qui ne sera pas interprété comme une instruction. On utilise les commentaires suivants :

// commentaire sur une ligne

# encore un commentaire sur une ligne

/\* commentaire

sur plusieurs

lignes \*/

**Saisie trim(fgets(STDIN))**

Elle se fait avec l'instruction :

$saisie = trim(fgets(STDIN));

// cela affecte la valeur saisie à la variable $saisie

La fonction **trim()** supprime les espaces avant et après la chaîne Cela permet de supprimer le retour chariot qui a été saisi par l'utilisateur lorsqu'il a appuyé sur la touche entrée.

**Variables $ =**

**Il n'est pas nécessaire de déclarer au préalable les variables qui seront utilisées, ni de préciser leur type, qui est défini par défaut et de façon transparente par PHP en fonction de la valeur affectée à la variable.**

Tous les noms de variable commencent obligatoirement par le signe **$**

L'affectation d'une valeur à une variable s'effectue par le signe **=**

Elle se lit de droite à gauche.

Attention, la comparaison s'effectue avec **==**.

$titre\_article = "Une image exceptionnelle de la végétation mondiale";

echo $titre\_article ; // affichage de la variable

echo "\n"; // saut de ligne

$nb = 139;

echo $nb ;

echo "<br>";

**Evaluation des chaînes de caractères entre guillemets doubles**

Lorsqu'une variable est placée dans une chaîne de caractères entre guillemets doubles, elle est évaluée par PHP, c'est à dire que PHP la remplace par sa valeur. Cela ne fonctionne pas avec les simples cotes.

<?php

$annee\_actu = "2014";

echo "Année : $annee\_actu";

?>

Résultat :

Année : 2014

Cela ne fonctionne pas avec les simples cotes :

<?php

$annee\_actu = "2014";

echo 'La variable $annee\_actu vaut '.$annee\_actu ;

?>

Résultat :

La variable $annee\_actu vaut 2014

**Principaux types de variables**

|  |  |  |  |
| --- | --- | --- | --- |
| Type | en français | Plage de valeurs | Exemple |
| integer | Entier | -2 147 483 648 à + 2 147 483 647 | $var = 12; |
| double | réel | 1.7E-308 à 1.7E+308 | $var = 3.1415957; |
| string | chaîne de caractères | caractère alphanumérique | $var = "bienvenue"; |
| boolean | booléen | true (vrai) ou false (faux) | $var = true; |

**Constantes define**

Il est par exemple utile, dans un site, de définir une constante pour une valeur qui se trouve répétée un certain nombre de fois. Comme par exemple la dénomination de la société éditrice d'un site internet, du titre du site ou de son auteur, si ces informations sont récurrentes.

<?php

define ("PI", 3.14);

echo "Le nombre Pi vaut : ". PI;

// affiche: Le nombre Pi vaut 3.14

?>

**Opérateurs numériques + - \* / %**

|  |  |  |
| --- | --- | --- |
| **Exemple** | **Nom** | **Description** |
| $a + $b | somme | rend la somme de la valeur de $a et de la valeur de $b |
| $a - $b | différence | rend la différence de la valeur de $a et de la valeur de $b |
| $a \* $b | multiplication | rend le produit de la valeur de $a par la valeur de $b |
| $a / $b | division | rend le résultat de la division de la valeur de $a par la valeur de $b |
| $a % $b | modulo | rend le reste de la division entière de la valeur de $a par la valeur de $b |

**Concaténation .**

L'opérateur point « . » permet de concaténer deux chaînes de caractère.

<?php

$date\_angl = "2003-10-12";

$ladate = "La date au format " . "anglais est " . $date\_angl;

echo "$ladate";

// affiche : La date au format anglais est 2003-10-12

?>

**Si... Sinon if…. else…**

Il est très fréquent que l'exécution d'une séquence d’instructions dépende d'une condition.

La syntaxe PHP est la suivante (les parties entre crochets sont facultatives) :

**if (condition [ == TRUE ] )**

**{ ... }**

**[ else { ... } ]**

Exemple : un programme qui affiche "Vous êtes majeur" ou "Vous êtes mineur" en fonction de l'âge saisi.

<?php

$age = trim(fgets(STDIN));

if ($age >= 18) {

echo "Vous êtes majeur.\n";

}

else {

echo "Vous êtes mineur.\n";

}

}

?>

Le mot clé **if** suivi d'une **condition** permet d'isoler et de conditionner l'exécution d'un **bloc d'instructions** délimité par des accolades ({ et }).

Un bloc d'instructions peut être composé de *0* (zéro) à *n* instructions, et peut lui-même inclure d’autres structures (condition, itération, exception).

Une condition est une **expression booléenne**. Une expression est qualifiée de *booléenne* si sa valeur ne peut être que soit vrai (true) soit faux (false), exclusivement. Elle tient son nom du mathématicien et logicien anglais George Boole (1815-1864).

**Opérateurs de comparaison**

* == : égalité
* < : inférieur
* > : supérieur
* <= : inférieur ou égal
* >= : supérieur ou égal
* != : différence

**Opérateurs logiques && || ^ !**

Lorsqu'un bloc d'instructions dépend de **plusieurs conditions**, ces dernières peuvent être conjuguées par l'intermédiaire des opérateurs logiques, les plus connus étant ET, OU et NON.

Les opérateurs logiques évaluent des expressions booléennes. Le résultat est toujours une valeur booléenne (TRUE ou FALSE).

|  |  |  |
| --- | --- | --- |
| **Expression booléenne** | **Nom** | **est VRAI** |
| (argv(1)==argv(2)  **&&** argv(1)=="toto") | **ET** | si **les deux** expressions, à gauche et à droite de l'opérateur, sont vraies. |
| (argv(1)==argv(2)  **||** argv(1)=="toto") | **OU** | si **une des deux ou les deux** expressions, à gauche et à droite de l'opérateur, est vraie. |
| (argv(1)==argv(2)  **^** argv(1)=="toto") | **OU exclusif** | si **seulement une des deux** expressions, mais pas les deux, à gauche et à droite de l'opérateur, est vraie. |
| **!** (argv(1)==argv(2)) | **Négation** | si l'expression est fausse. |

**Selon switch… case…**

Cette structure permet d'évaluer une seule fois une condition, et d'effectuer de multiples tests de comparaisons d'égalité.

<?php

// on a saisi $modulo2

switch ($modulo2) {

case 0: echo "Pair\n";

break;

case 1: echo "Impair\n";

}

echo "Bye.\n";

?>

Le **switch** a remplacé le **if**. Attention, une fois entré dans un cas, il faut **placer l'instruction** break; **en fin de bloc**. En effet, une fois entré dans un cas, les instructions des cas suivants sont TOUTES exécutées, d'où l'intérêt de l'instruction break.

Enfin il existe une clause **default** pour traiter les cas non filtrés.

Le switch s'avère pratique lorsqu'il y a plus de deux cas à traiter.

**Opérateur ternaire (…) ?… :…**

Dans le cas d'une alternative de type binaire (si la condition est vrai alors... sinon...), certains langages, comme PHP, proposent un opérateur ternaire (attendant 3 arguments).

Sa syntaxe est :

(<expression booléenne>) ? <expression si vrai> : <expression si faux>

Exemple :

<?php

echo "Entrez un nombre : ";

$x = trim(fgets(STDIN));

$modulo2 = $x % 2;

$res = ($modulo2==0) ? "Pair\n" : "Impair\n";

echo $res;

echo "Bye.\n";

?>

L'instruction :

$res = ($modulo2==0) ? "Pair\n" : "Impair\n";

se lit : *si $modulo2==0 alors la valeur de l'expression à affecter à la variable* $res *est "Pair\n" sinon ce sera "Impair\n".*

**Pour for**

Syntaxe :

**for (initialisation; condition de poursuite; instruction finale)**

**{corps de la boucle}**

Voici un exemple d'utilisation de cette structure. Ce programme construit, de façon itérative, une chaîne de caractères composée d'étoiles pour, finalement, l'afficher.

1: <?php

2: for ($i= 0, $res=""; $i < 10 ; $i++){

3: $res = $res . "\*";

4: }

5: echo $res . "\n";

6: ?>

La structure for comporte 4 parties :

* **initialisation** : de la ou des variables utilisées au sein de la boucle.
* **condition de poursuite** : doit être vérifiée pour passer à l'étape suivante.
* **corps de la boucle**: contient les instructions générales devant être répétées.
* **instruction finale** : une action (ou une suite d'actions - des instructions séparées par une virgule) à exécuter systématiquement **après le corps**. Typiquement, c'est ici que l'on fait évoluer (augmenter -**incrémentation**- ou diminuer -décrémentation-) la valeur de la **variable de boucle** $i.

**Tant que while**

Syntaxe :

**while (condition)**

**{ ... }**

Exemple :

<?php

$i=0;

$res="";

**while** ($i < 10) {

$res = $res . "\*";

$i++;

}

echo $res . "\n";

?>

**Répéter... tant que do... while**

Syntaxe :

**do**

**{ ... }**

**while (condition)**

Exemple :

<?php

$max=10;

...

$i=0;

$res="";

**do** {

$res = $res . "\*";

$i++;

}

**while** ($i < $max);

echo $res . "\n";

?>

**Fonctions function… return…**

**Définition**

Une fonction peut recevoir des données d'entrée, elle réalise un traitement et génère un résultat. Les termes *sous-programme*, *routine* et *fonction* sont synonymes.

D'un **point de vue utilisateur**, une fonction est une sorte de « boîte noire », à laquelle on fournit des données (les **paramètres réels**) et qui rend un résultat (la **valeur de la fonction**).

D'un **point de vue concepteur**, une fonction est un sous-programme qui reçoit des valeurs (les **paramètres formels**), et qui, à partir de ceux-ci, détermine un résultat. Ce résultat est retourné au programme utilisateur (dit **programme appelant**) par l'intermédiaire de l'instruction **return <une valeur>;**

**Utilisation**

Pour utiliser une fonction, il est nécessaire de connaître son **interface** et son **mode d'emploi**. Ces deux informations sont généralement fournies sous la forme d'une **documentation de l'API** (*Application Programming Interface*).

**Exemple de description d'une fonction**

|  |  |
| --- | --- |
| **Interface documentée** | **Exemple** |
| int strlen ( string $str )  strlen retourne la taille (sous forme d’un entier) de la chaîne de caractères $str passée en argument | $str = "abcde";  echo strlen($str);  // retourne 5 |

**Passage de paramètres**

Certaines fonctions ne prennent aucun argument et d'autres plusieurs. Lorsqu'on appelle la fonction, il faut lui fournir le bon **nombre** d'arguments, dans le bon **ordre** et dans les **types** demandés.

string substr ( string $chaine, int $start [, int $length] )

$rest = substr("abcdef", 1, 3); // retourne "bcd"

Le fait de marquer, dans la documentation, le dernier argument entre crochets, signifie que ce dernier est **facultatif**, c'est à dire que cette fonction peut très bien s'utiliser avec 2 ou 3 arguments.

**Conception**

Concevoir une fonction c'est :

* Identifier et documenter une **interface** (entête de la fonction)
* Établir un **test** d'utilisation
* Fournir un **corps**

L'interface s'occupe du QUOI. Elle présente la **syntaxe d'utilisation** associée à des commentaires (la sémantique).

Le test donne un exemple du POURQUOI. Il décrit une façon d'utiliser la fonction.

Le corps explique le COMMENT. Il décrit l'algorithme, il réalise le contrat défini par l'entête.

function racine\_carree($x) {

$r = $x;

while ($r\*$r>$x){

$r--;

}

return($r);

}

Le mot clé **function** permet de déclarer et définir une fonction.

Le mot clé **return** fournit une **valeur** à la fonction et permet de quitter immédiatement le corps de la fonction.

**Tableaux à une dimension array…**

**Déclaration**

$tabJeau = **array();** // tableau vide

$tab = array(1,2, « chaine »,45.3);

**Tableaux associatifs**

**$tab['cle'] =valeur;**

$tab ['nom'] = 'dupont';

$tab['prenom'] = 'pierre ';

**Ajouter des éléments**

$tabJeau[] = 20; // ajoute 20 à la fin du tableau

**array\_push**($tabJeau,$j); // ajoute l'élément $j à la fin du tableau

$tab[1] = 10; // remplace la 2nde case du tableau

**Créer des suites dans un tableau**

$tabSL = **range**('a','z');

$tabSN = range(1,10);

**Créer un tableau à partir d'une chaîne**

**array explode ( string separateur,string chaineADecouper[,(int nbMax])**

$tabt =explode ('a',"tagazout");

// donne Array ( [0] => t [1] => g [2] => zout )

$tabd = explode('/',"31/12/2014");

// donne Array ( [0] => 31 [1] => 12 [2] => 2014 )

$tabc = explode(' ',"Comment ça va?");

// donne Array ( [0] => Comment [1] => ça [2] => va? )

**Nombre d'éléments**

**count($tab);** // pour compter le nombre d'éléments d'un tableau

**sizeof($tab);**

**array array\_count\_values($tab)** retourne un tableau associatif comportant pour chaque valeur différente, son nombre d'occurrences dans le tableau

Voir la liste des **fonctions PHP sur les tableaux**, sur www.w3schools.com