eVOGUE

Liam TARDIEU

www.evogue.fr

SUPPORT DE COURS / PHP – PARTIE 1

Sommaire

➢ Présentation

DEFINITION

PHP (PHP : Hypertext Preprocessor) est un langage de scripts généralistes et Open Source, spécialement conçu pour le développement d'applications web. Il peut être intégré facilement au HTML.

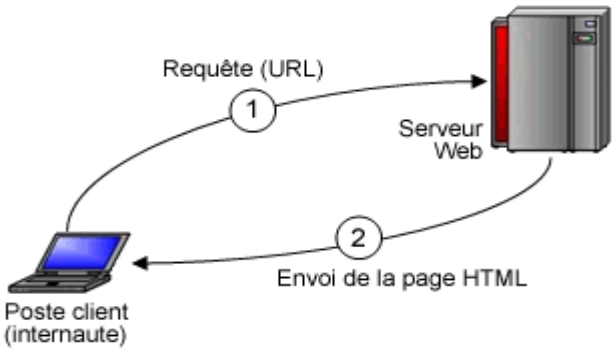
HISTORIQUE

Le langage PHP fut créé en 1994 par Rasmus Lerdorf pour son site web. Au fur et à mesure qu'il ajoutait de nouvelles fonctionnalités, Rasmus a transformé la bibliothèque en une implémentation en langage C, capable de communiquer avec des bases de données et de créer des applications dynamiques et simples pour le Web. Rasmus décida alors en 1995 de publier son code, pour que tout le monde puisse l'utiliser et en profiter. En 1997, deux étudiants, Andi Gutmans et Zeev Suraski, redéveloppèrent le cœur du langage. Ce travail aboutit un an plus tard à la version 3 de PHP, devenu alors PHP: Hypertext Preprocessor. Peu de temps après, ils commencèrent la réécriture du moteur interne de PHP. Ce fut ce nouveau moteur, appelé Zend Engine - qui servit de base à la version 4 de PHP. La version actuelle est la version 5, sortie le 13 juillet 2004.
En 2007, PHP est utilisé par plus de 20 millions de sites Web à travers le monde.

FONCTIONNEMENT

Architecture web minimale

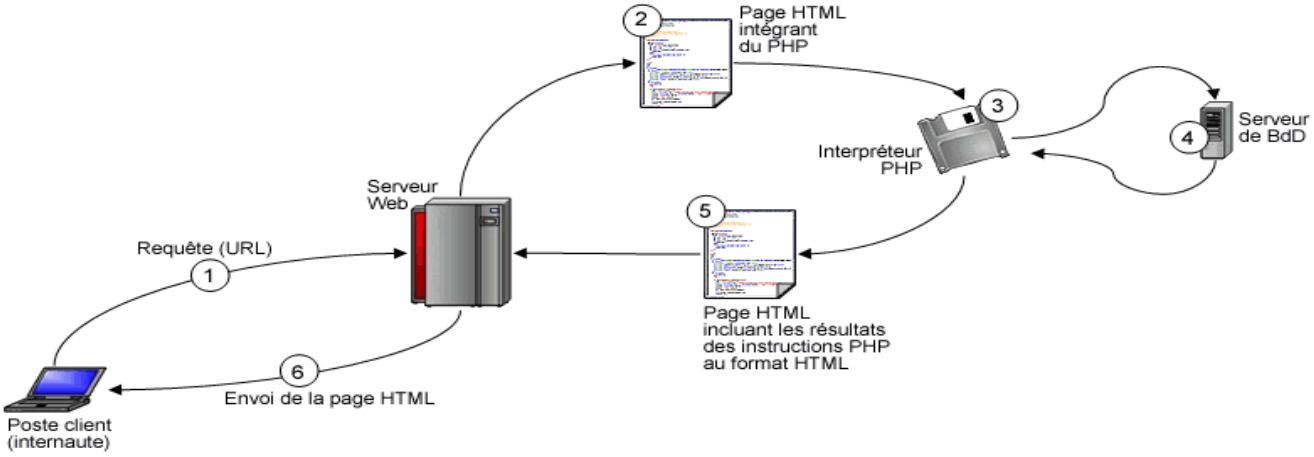
Pour rappel, voici le fonctionnement d'une navigation web simple (HTML, et éventuellement JavaScript et CSS) :



- l'internaute effectue une requête (depuis un lien hypertexte, ou en saisissant une URL) (1)
- la requête arrive sur le serveur web, qui retourne la page HTML correspondante (2)
- la page est lue par le navigateur de l'internaute

Architecture web avec PHP

Notez que cette architecture est semblable pour tous les langages de développement web.



- l'internaute effectue une requête (depuis un lien hypertexte, ou en saisissant une URL) (1)
- la requête arrive sur le serveur web, qui transmet la page PHP à l'interpréteur PHP (2)
- l'interpréteur décode les instructions (3)
- si nécessaire, l'interpréteur lance des requêtes sur le serveur de Base de Données (4)
- l'interpréteur retourne la page au serveur web. Cette page ne contient plus que du code HTML (et JavaScript, CSS) (5)
- le serveur retourne la page à l'internaute (6)
- la page est lue par le navigateur de l'internaute

SERVEUR WEB

Nous venons de le voir, PHP a besoin de s'exécuter sur un serveur et nécessite un interpréteur PHP.
Un serveur Web peut être :

- un ordinateur tenant le rôle de serveur informatique sur lequel fonctionne un logiciel serveur HTTP
- le serveur HTTP lui-même ;
- un ensemble de serveurs permettant le fonctionnement d'applications Web.

Le plus souvent, un serveur Web fait fonctionner plusieurs logiciels qui fonctionnent en parallèle. Nous retrouvons la combinaison Apache (serveur HTTP), MySQL (serveur de base de données) et PHP, tous libres. Sous Linux, cette combinaison s'appelle **LAMP** (sigle de « Linux, Apache, MySQL, PHP ») ; sous Windows, **WAMP** (« Windows, Apache, MySQL, PHP ») ; et sous Mac, **MAMP** (« Macintosh, Apache, MySQL, PHP »). Il contient en plus le module PHPMyAdmin, qui permet la gestion des bases de données. Comme il n'est pas toujours très aisé de disposer en permanence d'un serveur de test distant, nous allons en installer un de façon locale. Son installation est indispensable pour développer nos pages PHP.

➢ Syntaxe de PHP

INSTRUCTIONS ET INTEGRATION DANS UNE PAGE WEB

Pour intégrer du PHP dans une page web, il est nécessaire de préciser à l'interpréteur les parties de scripts qui lui sont dévolues. Voici un exemple permettant d'intégrer un passage de code PHP.
Il est important que cette page possède l'extension « .php ».

Exemple

```
<?php  echo  phpinfo();  ?>
```

< ?php : permet d'ouvrir un passage d'instructions PHP.

?> : permet de fermer un passage d'instructions PHP (il n'est pas obligatoire).

echo : permet d'effectuer un affichage.

phpinfo() : est une fonction prédéfinie qui nous renseigne sur la configuration PHP du serveur.

ASSSOCIATION DU PHP AVEC D'AUTRES LANGAGES

En PHP, comme dans d'autres langages, toutes les instructions se terminent par un point-virgule «**;**».

Exemple

```
<?php
      echo 'Bonjour';
      echo '<br />';
      echo 'Bienvenue';
?>
<hr />
```

Le code HTML peut être intégré à l'intérieur de l'instruction « echo » ou en dehors du passage PHP.
Le point virgule de la dernière instruction est facultatif néanmoins il est préférable de toujours l'écrire.

COMMENTAIRES

Les commentaires sont des parties de texte ignoré par l'interpréteur et permettent de documenter le code source afin de l'expliquer.

Exemple

```
echo 'Texte'; // Ceci est un commentaire sur une seule ligne
echo 'Texte'; /* Ceci est un commentaire sur
                          plusieurs lignes */
echo 'Texte'; # Ceci est un commentaire sur une seule ligne
```

Les commentaires sur une seule ligne commencent au signe // ou # et se terminent à la fin de la ligne.

Effectivement un code bien documenté permet de rendre le code source plus simple à lire et à comprendre. Il faut aussi penser qu'un autre développeur (étranger au projet) peut être amené à découvrir le code-source et les commentaires pourront lui faciliter la tâche.

➢ Variables

DEFINITION

« Emplacement de stockage nommé qui peut contenir des données pouvant être modifiées pendant l'exécution du script. Chaque variable a un nom qui l'identifie de façon unique à l'intérieur de son niveau de portée. »

Plus simplement, une variable est une zone en mémoire du système, qui sert à conserver une valeur.

NOMMAGE

Pour donner un nom à vos variables, il est nécessaire de respecter les règles suivantes :

- le nom de la variable commence toujours par le signe dollar « **$** »
- il est obligatoirement une chaîne de caractères alphanumériques
- les caractères spéciaux (espaces, signes de ponctuation, opérateurs, caractères accentués) ne sont pas acceptés, sauf le tiret-bas (underscore _)
- Une variable ne peut pas commencer par un chiffre mais peut en contenir
- le nom est sensible à la casse

DECLARATION ET AFFECTATION

PHP ne nécessite pas de déclaration préalable de variables. La simple utilisation de celle-ci suffit.

Pour affecter une valeur à une variable, nous utilisons le signe égal « **=** » .

Exemple

```
$pays = "France" ;
```

La variable nommée « $pays » contient la valeur : « France ».

PORTEE

Une variable est systématiquement globale, sauf si elle est déclarée dans une fonction.

Dans ce cas la variable est locale à la fonction.

AFFECTATION DE VARIABLES PAR REFERENCE

Depuis PHP 4, il est possible d'affecter les valeurs aux variables par référence via le signe « **&** ». Cela signifie que la nouvelle variable ne fait que référencer (en d'autres termes, "devient un alias de", ou encore "pointe sur") la variable originale. Les modifications de la nouvelle variable affecteront l'ancienne et vice versa.

TYPES DE DONNEES

Les variables, comme toutes les données en PHP, sont de plusieurs types possibles.

Nombre (donnée numérique)

Une variable peut être un nombre entier (integer) ou un nombre à virgule flottante (double).

```
$a = 127; // un entier
echo gettype($a);
$b = 1.5; // un nombre à virgule
echo gettype($b);
```

Chaîne de caractères (donnée alphanumérique)

Une variable peut être une chaîne de caractères (string). Une chaîne de caractères est délimitée par des guillemets (double quotes ") ou des apostrophes (simple quotes ').

```
$a = "une chaîne";
echo gettype($a); // chaine de caractères (string)
$b = '127';
echo gettype($b); // chaine de caractères (string)
```

Booléen

Une variable peut être un booléen, et prendre une des deux valeurs possibles : vrai ou faux.
En PHP ces valeurs s'écrivent : TRUE ou FALSE.

```
$a = true;
echo gettype($a);
$b = false;
echo gettype($b);
```

Nous verrons plus tard que les variables peuvent aussi être de type : array, object, ressource, null, etc.

➢ Constantes

DEFINITION

« Élément nommé qui conserve une valeur constante pendant toute l'exécution d'un script ; peut être utilisé n'importe où dans le code à la place de valeurs réelles. Une constante est une chaîne ou une valeur numérique. »

NOMMAGE

Pour donner un nom à vos constantes, il est nécessaire de respecter les règles suivantes :

- le nom de la constante n'est pas préfixé (nous n'utilisons jamais le signe $)
- il est obligatoirement une chaîne de caractères alphanumériques
- les caractères spéciaux (espaces, signes de ponctuation, opérateurs, caractères accentués) ne sont pas acceptés, sauf le tiret-bas (underscore _)
- le premier caractère ne peut être un chiffre
- le nom est sensible à la casse. Par convention, nous utilisons uniquement des majuscules.

DECLARATION ET AFFECTATION

La déclaration des constantes se fait grâce à la fonction define() .

Exemple

```
define("CAPITALE","Paris");
echo CAPITALE;
```

CONSTANTES MAGIQUES

PHP propose des constantes prédéfinies :

__FILE__ renvoie le nom du fichier en cours d'interprétation.

__LINE__ renvoie le numéro de la ligne courante.

__FUNCTION__ renvoie le nom de la fonction courante.

__CLASS__ renvoie le nom de la classe courante.

__METHOD__ renvoie le nom de la méthode courante.

Ces constantes peuvent être utiles pour le débogage.

DIFFERENCE : CONSTANTE ET VARIABLE

Comme les variables, une constante est une zone en mémoire du système, qui sert à stocker une valeur. Mais contrairement à une variable, la valeur d'une constante ne peut jamais être modifiée durant l'exécution du script.

➢ Instruction de sortie

ECHAPPEMENT

Dans certains cas, il est possible (voir nécessaire !) d'utiliser le caractère d'échappement. Ce caractère est la barre oblique inversée (anti-slash \).

Exemple

```
echo 'aujourd\'hui' ;
echo "aujourd'hui" ;
```

APOSTROPHES ET GUILLEMETS

```
$ecole = 'ifocop' ;
echo "aujourd'hui nous allons chez " . $ecole;
echo "aujourd'hui nous allons chez $ecole";
echo 'aujourd'hui nous allons chez ' . $ecole;
echo 'aujourd'hui nous allons chez $ecole';
```

Sur la dernière ligne, en utilisant les apostrophes, la variable `$ecole` n'est pas évaluée, et s'affiche comme une chaîne ordinaire. En utilisant les guillemets, la variable `$ecole` est évaluée, et c'est son contenu qui est pris en compte dans la chaîne.

AFFICHAGE : ECHO ET PRINT

Instruction echo, Instruction print

Affiche une chaîne de caractères. Si le paramètre de la fonction echo (ou print) est une variable, la fonction affichera le contenu de celle-ci. L'utilisation des parenthèses est facultative.

Exemple

```
$a = "Pomme ";
$b = "Poire ";
$c = "Abricot ";
echo $a;              // print $a;
echo $b;              // print $b;
echo $c;              // print $c;
```

Différences

Il existe peu de différence entre les instructions print et echo : pas de différence de temps d'exécution majeure, en revanche, la fonction print peut-être évaluée contrairement à la fonction echo mais ces différences sont si peu significatives que nous pourrons utiliser l'une ou l'autre indifféremment.

AFFICHAGE : PRINT_R ET VAR_DUMP

Les fonctions « ***print_r/var_dump*** » permettent d'afficher des informations lisibles pour une variable (utile pour les variables de type « ***array*** » ou « ***object*** »).
L'utilisation des parenthèses est obligatoire.

```
$a = 'bonjour';
print_r($a); var_dump($a);
echo "<hr />";
$b = array("a","b","c","d");
print_r($b); var_dump($b);
```

Différences

Print_r : réalise une sortie mise en forme de la variable.

Var_dump : affiche diverses informations disponibles sur une variable (type, longueur et valeur).

➢ Les opérateurs

OPERATEURS ARITHMETIQUES

Addition, Soustraction, Multiplication et Division

```
$a = 10 ; $b = 2 ;
echo $a + $b ; // affiche 12

$a = 10 ; $b = 2 ;
echo $a - $b ; // affiche 8

$a = 10 ; $b = 2 ;
echo $a * $b ; // affiche 20

$a = 10 ; $b = 2 ;
echo $a / $b ; // affiche 5
```

Modulo

Le modulo (%) est le reste de la division entière de l'opérande de gauche par l'opérande de droite.

```
$a = 13 ; $b = 4 ;
echo $a % $b ; // affiche 1, puisque le reste de 13/4 est 1
(13=4*3+1)
```

OPERATEUR DE CONCATENATION

La concaténation est une opération pouvant servir, par exemple, avec les chaînes de caractères.

La concaténation de deux chaînes est la chaîne formée par ces deux chaînes mises bout à bout.

```
$a = 'salut ' ;
$b = 'à tous !' ;
$c = $a.$b ;
echo $c ; // affiche "salut à tous !"
```

L'opérateur de concaténation peut se faire via le signe point « **.** » ou le signe virgule « **,** ».

OPERATEURS D'AFFECTATION

Affectation simple

L'opérateur d'affectation de base est le signe égal « **=** ».

Attention, contrairement aux mathématiques, cet opérateur n'est pas commutatif. C'est-à-dire que l'affectation peut se traduire par : « la valeur de l'opérande de droite affecte l'opérande de gauche. »

Opération-affectation

Il existe une forme raccourcie pour exprimer une opération suivie d'une affectation.

```
$a = 10 ; $b = 2 ;
$a += $b ; // équivaut à $a = $a + $b (ici $a vaut 12, voir ci-dessus)
$a -= $b ; // équivaut à $a = $a - $b (ici $a vaut 8)
$a *= $b ; // équivaut à $a = $a * $b (ici $a vaut 20)
$a /= $b ; // équivaut à $a = $a / $b (ici $a vaut 5)

$c = 'France' ; $d = 'Italie' ;
$c .= $d ; // équivaut à $c = $c . $d
echo $c ; // affiche FranceItalie
```

OPERATEURS D'INCREMENTATION

L'opérateur d'incrémentation est le signe « **++** », il permet d'augmenter la valeur d'une variable de 1 (incrémenter).

L'opérateur de décrémentation est le signe « **--** » , il permet de diminuer cette valeur de 1 (décrémenter).

$a++ équivaut à $a=$a+1 ;

Cependant :

- l'addition (resp. soustraction) peut être effectuée avant l'affectation, c'est la pré-incrémentation (resp. pré-décrémentation), ou
- l'addition (resp. soustraction) peut être effectuée après l'affectation : c'est la post-incrémentation (resp. post-décrémentation).

OPERATEURS DE COMPARAISON

Les opérateurs de comparaison présentés ici permettent de comparer deux valeurs.

Pour les exemples suivants, nous considérerons $a=10 ; $b=5 ; $c=2 ;

L'égalité se teste avec le signe « **==** ».

```
$a == $b                //est faux
$a == $b*$c             //est vrai
```

La différence se teste avec le signe « **<>** » ou le signe « **!=** ».

```
$a <> $b                //est vrai
$a != $b*$c             //est faux
```

La stricte supériorité se teste avec le signe « **>** », la supériorité simple avec le signe « **>=** ».

```
$a > $b                 //est vrai
$a > $b*$c              //est faux
$a >= $b*$c             //est vrai
```

La stricte infériorité se teste avec le signe « **<** », l'infériorité simple avec le signe « **<=** ».

```
$a < $b                 //est faux
$a < $b*$c              //est faux
$a <= $b*$c             //est vrai
```

Comparaison du type et de la valeur

L'égalité peut également se tester avec le signe « **===** ». Cela à pour but de comparer le type en plus de la valeur de la variable.

```
$a = 1;
$b = "1";
if($a === $b)
{
      echo "il s'agit de la même chose";
}
```

Avec la présence du triple égal, le teste ne fonctionne pas car les types des variables sont différents. La valeur « 1 » et la valeur « 1 » sont identiques mais le type Int (entier) est égal au type String (chaine de caractère).

OPERATEURS LOGIQUES

Comme les opérateurs de comparaison, les opérateurs logiques seront utiles dans le cadre de tests.

Pour les exemples suivants, nous considérerons $a=10 , $b=5 , $c=2 .

Opérateur ET

- signe : (AND) ou (**&&**)
- sert à effectuer deux comparaisons simultanées

Si les deux comparaisons renvoient Vrai, alors l'opération globale renverra Vrai.

Si une comparaison renvoie Faux, l'autre Vrai, alors l'opération globale renverra Faux.

Si les deux comparaisons renvoient Faux, alors l'opération globale renverra Faux.

```
($a > $b) && ($b > $c)     est vrai (on a vrai ET vrai)
($a > $b) && ($c > $a)     est faux (on a vrai ET faux)
```

Opérateur OU

- signe : (OR) ou (**||**)
- sert à effectuer deux comparaisons simultanées

Si les deux comparaisons renvoient Vrai, alors l'opération globale renverra Vrai.

Si une comparaison renvoie Faux, l'autre Vrai, alors l'opération globale renverra Vrai.

Si les deux comparaisons renvoient Faux, alors l'opération globale renverra Faux.

```
($a > $b) || ($c > $a)     est vrai (on a vrai OU faux)
($b > $a) || ($c > $a)     est faux (on a faux OU faux)
```

Opérateur OU exclusif

- signe : (XOR)
- sert à effectuer deux comparaisons simultanées

Si les deux comparaisons renvoient Vrai, alors l'opération globale renverra Faux.

Si une comparaison renvoie Faux, l'autre Vrai, alors l'opération globale renverra Vrai.

Si les deux comparaisons renvoient Faux, alors l'opération globale renverra Faux.

```
($a > $b) XOR ($b > $c)    est faux (on a vrai XOR vrai)
($a > $b) XOR ($c > $a)    est vrai (on a vrai XOR faux)
```

Opérateur de Négation (not)

- signe : (**!**)
- sert à effectuer une comparaison négative.

Si une comparaison renvoie Vrai, sa négation renverra Faux, et inversement.

```
($a < $b)  est faux
!($a < $b) est vrai
```

OPERATEUR TERNAIRE

L'opérateur ternaire « **?** » est une version simplifiée.

Il teste une condition et réalise l'une ou l'autre des instructions selon le résultat de ce test.

➢ Structures Conditionnelles

DEFINITION

Nous appelons structure conditionnelle, la structure qui permet de réaliser une (ou plusieurs) instruction(s) sous certaines conditions. Elle nécessite l'utilisation des opérateurs de comparaisons et parfois des opérateurs logiques.

CONDITION : IF

Définition

L'instruction IF permet l'exécution conditionnelle d'une partie de code.

- la condition doit être entre parenthèses « **()** »
- les accolades « **{}** » ne sont pas obligatoires s'il n'y a qu'une instruction.
- il est possible de cumuler les conditions grâce aux opérateurs logiques

Syntaxe

```
if (condition) {
      instructions si condition vraie ;
}
$a = 10 ;
if($a > 2){
      echo 'la condition est vraie ';
}
echo 'suite du code';
// affiche "la condition est vraie suite du code"
```

CONDITION : ELSE

Définition

Il est parfois nécessaire d'exécuter une instruction si une condition est remplie, et une autre si cette condition n'est pas remplie.

L'instruction ELSE fonctionne après un IF et exécute les instructions correspondantes au cas où la condition du IF est Fausse.

Syntaxe

```
if (condition) {
      instructions si condition vraie ;
} else {
      instructions si condition fausse ;
}
$a = 2 ;
if($a > 5){
      echo 'la condition est vraie ';
}
else{
      echo 'la condition est fausse ';
}
// affiche "la condition est fausse"
```

CONDITION : ELSEIF

Définition

L'instruction ELSEIF, comme son nom l'indique, est une combinaison de ELSE et de IF.
Comme l'instruction ELSE, elle permet d'exécuter une instruction au cas où la condition du IF est Fausse. Mais, à la différence de l'expression ELSE, l'instruction ne sera exécutée que si l'expression conditionnelle ELSEIF est évaluée comme Vraie.

Syntaxe

```
if (condition1)
{
      instructions si condition1 vraie ;
}
elseif (condition2)
{
      instructions si condition1 fausse et condition2 vraie ;
}
else
{
      instructions si condition1 et condition2 fausse ;
}
$a = 2 ;
if($a > 5)
{
      echo 'A supérieur à 5 ';
}
elseif($a == 5)
{
      echo 'A égal à 5';
}
else
{
      echo 'A n\'est ni égal ni supérieur à 5';
} // affiche "A n'est ni égal ni supérieur à 5".
```

Variante

Nous pouvons aussi écrire ELSE IF (avec un espace) au lieu de ELSEIF (sans espace), le résultat est strictement identique.

En utilisant l'opérateur ternaire, nous pouvons simplifier l'écriture du if/else.

Syntaxe

```
(condition) ? instruction si vrai : instruction si faux ;
$a = 'France';
echo ($a=='France') ? 'Vive la France' : 'il ne s\'agit pas de la France';
// affiche "Vive la France"
```

CONDITION : SWITCH

Définition

La condition SWITCH permet de faire plusieurs tests d'égalité sur le contenu d'une même variable.

Elle fait partie des structures conditionnelles.

Syntaxe

```
switch ($variable) {
	case 'valeur 1' :
		instructions si $variable vaut 'valeur 1' ;
		break ;
	case 'valeur 2' :
		instructions si $variable vaut 'valeur 2' ;
		break ;
	case 'valeur n' :
		instructions si $variable vaut 'valeur n' ;
		break ;
	default :
		instructions par défaut ;
		break ;
}
```

Fonctionnement

La variable présente entre parenthèses, après l'instruction switch, est testée.

Pour chacun des cas (case), si la valeur de la variable est strictement égale à la valeur exprimée (entre guillemets), alors les instructions qui suivent sont exécutées.

Dès que le script rencontre l'instruction break, il passe au test suivant.

Si la valeur de la variable n'est égale à aucun des cas proposés, ce sont les instructions par défaut (default) qui sont effectuées.

```
$couleur = 'vert';
switch ($couleur) {
	case 'bleu':
		echo 'Vous aimez le bleu';
		break;
	case 'rouge':
		echo 'Vous aimez le rouge';
		break;
	case 'vert':
		echo 'Vous aimez le vert';
		break;
	default:
		echo 'Vous n\'aimez ni le bleu, ni le vert, ni le
rouge...';
		break;
}
// affiche 'vous aimez le vert'
```

➢ Structures itératives

DEFINITION

Nous appelons structure itérative (ou récursive), la structure qui permet d'exécuter plusieurs fois les mêmes instructions. Elle permet de faire "en boucle" un bloc d'instructions.

BOUCLE : WHILE

Définition

L'instruction while permet d'exécuter plusieurs fois une série d'instructions, aussi longtemps qu'une condition est réalisée.

Syntaxe

```
while (condition)
{
      instructions ;
}
```

Fonctionnement

1. *la condition est testée*
2. *elle renvoie :*
   - 2.1. *vrai : exécution du bloc d'instructions et retour à 1*
   - 2.2. *faux : non-exécution du bloc d'instructions et suite du script*

Prenez garde à ne pas créer de structure itérative sans fin !

En effet, si la condition renvoie vrai mais que rien dans les instructions ne modifie les paramètres de cette condition, celle-ci renverra toujours vrai : on obtient une boucle infinie et un plantage certain !

Exemple

```
$i = 0;
while ($i<3)
{
      echo "$i---";
      $i++; // incrémentation du "compteur"
}
// affiche "0---1---2---"
```

BOUCLE : DO-WHILE

Définition

L'instruction do while permet d'exécuter plusieurs fois une série d'instructions, aussi longtemps qu'une condition est réalisée.

Syntaxe

```
do
{
      instructions ;
} while (condition) ;
```

Fonctionnement

1. *exécution préalable du bloc d'instructions*
2. *la condition est testée*
3. *elle renvoie*
    - 3.1. *vrai : retour à 1*
    - 3.2. *faux : suite du script*

```
$i = 0;
do{
      echo "$i---";
      $i++;
} while ($i<3);
// affiche "0---1---2---"
```

Différences

La différence entre les structures while et do while se situe au niveau de la chronologie des événements.

Dans le cas de la boucle while, la condition est examinée avant la boucle tandis que pour la boucle do-while elle est examinée à la fin. Ainsi, même si cette condition n'est pas vérifiée, la boucle s'exécutera une fois.

BOUCLE : FOR

Définition

L'instruction for permet d'exécuter plusieurs fois une série d'instructions, selon des conditions prévues à l'avance.

Syntaxe

```
for (initialisation; condition; évolution)
{
      instructions ;
}
```

- *initialisation* : indique la variable qui servira de compteur ainsi que sa valeur initiale.
- *condition* : condition pour continuer les boucles. L'instruction s'achève dès que la condition renvoie faux.
- *Evolution* : évolution du compteur à chaque boucle (généralement une incrémentation).

Fonctionnement

1. *le compteur est créé (et affecté d'une valeur) au début de l'instruction*
2. *la condition est testée*
3. *elle renvoie :*
   - 3.1. *vrai : exécution du bloc d'instructions, modification du compteur et retour à 2*
   - 3.2. *faux : non-exécution du bloc d'instructions et suite du script*

```
for ($i=0; $i<3; $i++)
{
      echo "$i---";
}
// affiche "0---1---2---"
```

BOUCLE : FOREACH

Définition

L'instruction foreach permet d'exécuter plusieurs fois une série d'instructions, selon des conditions prévues à l'avance.

Foreach fonctionne uniquement sur les tableaux (array) et les objets (object), elle retournera une erreur si vous tentez de l'utiliser sur une variable d'un autre type ou non initialisée.

Syntaxe

```
foreach()
{
      instructions ;
}
```

Fonctionnement

Dans cet exemple, l'instruction foreach est un moyen simple de passer en revue un tableau.

```
$liste = array("gregoire","nathalie","emilie","françois","georges");
Foreach($liste as $indice => $element)
{
      echo $element ;
}
```

➢ Inclusions

DEFINITION

Voici quelques fonctions très simples mais très utiles !

Une inclusion est un fichier contenant un morceau de code (HTML ou PHP) qui peut être utilisé à tout moment et dans toutes les pages d'un site par un simple appel de ce fichier…

Soit un fichier inclusion nommé *exemple.inc.php* :

```
<?php
      echo "Voici du texte contenu dans l'inclusion<br />\n";
?>
```

Un fichier d'inclusion n'est pas différent en soi d'un autre fichier php. C'est pourquoi il n'existe aucune règle concernant les noms de ces fichiers ; l'extension .inc.php est seulement un usage. Attention, on trouve parfois des scripts où les inclusions ont l'extension .inc (sans .php). Il est conseillé de les renommer : en effet, un fichier .inc ne garantit aucune sécurité, les serveurs PHP renvoyant parfois directement le code de l'inclusion sans l'interpréter…

FONCTION INCLUDE()

Pour utiliser notre inclusion dans une autre page, on peut utiliser la fonction include() .

Les parenthèses sont optionnelles.

```
<?php
      echo "Avant <br />\n";
      include "exemple.inc.php";
      include("exemple.inc.php");
      echo "Après <br />\n";
?>
```

Si le chargement de l'inclusion pose un problème, PHP retourne un warning et continue l'exécution du script.

FONCTION REQUIRE()

On peut aussi utiliser la fonction require() .

Là aussi, les parenthèses sont optionnelles.

```
<?php
      echo "Avant <br />\n";
      require "exemple.inc.php";
      require("exemple.inc.php");
      echo "Après <br />\n";
?>
```

Si le chargement pose un problème, PHP retourne une erreur fatale et arrête l'exécution du script.

FONCTIONS INCLUDE_ONCE() ET REQUIRE_ONCE()

Ces fonctions sont équivalentes aux précédentes, sauf qu'elles évitent qu'un même fichier soit inclus deux fois.

Dans l'exemple le texte de l'inclusion n'apparaît qu'une fois, bien que nous l'ayons demandé 2 fois.

```
<?php
      echo "Première fois : <br />\n";
      include_once "exemple.inc.php";
      echo "Deuxième fois : <br />\n";
      include_once "exemple.inc.php";
      echo "Fin";
?>
```

En cas d'un include_once suivi d'un require_once, les doublons seront tout de même évités !

INCLUSION DE FICHIERS POUR COMPOSER UN SITE

Les sites internet dynamiques ou applications ont besoin de réutiliser des parties de code identique à plusieurs endroits, cela serait une perte de temps considérable de réécrire ou copier/coller le code à chaque endroit ayant ce besoin. De plus cela donne d'avantage de lignes de code sur chaque page.

Il peut s'agir d'une partie d'un site (par exemple : les entêtes, une zone, etc.), ou bien des librairies de fonctions utilisateurs ou encore des fichiers de configuration, etc.

Plus tard, si une modification devait avoir lieu il faudrait rouvrir chaque fichier contenant ce code pour les modifier un à un.

C'est précisément là qu'interviennent les fonctions include() et require(), elles permettent d'importer et d'exécuter le code à réutiliser dans la page.

De nos jours, ces fonctions sont incontournables et représentent un avantage certain, en plus de l'économie réalisée en ligne de code sur la page (tout le code ne se trouve pas sur la page même mais il est importé d'un autre fichier), une modification ne se fait plus qu'à un seul endroit. En effet, le changement s'effectue sur un fichier et c'est ce fichier qui est importé sur tout le site, la modification prend effet immédiatement.

De nombreux sites utilisent les fonctions include/require et grâce à ses fonctions, les systèmes de fichiers appelés « Template » ont fait leur apparition.

Il s'agit d'un modèle de mise en page où l'on place images et textes. Aussi, il est souvent utilisé de manière répétitive pour créer des documents présentant une même structure. Plusieurs contenus peuvent partager le même Template (modèle de page).

DIFFERENCE : INCLUDE ET REQUIRE

Pourquoi existe-t-il deux fonctions différentes si elles accomplissent la même tâche ?

Leur fonctionnement est strictement le même mais la différence qui les sépare réside dans la gestion des erreurs.

La fonction include() renverra une erreur de type WARNING si elle n'arrive pas à ouvrir le fichier en question. De ce fait l'exécution du code qui suit dans la page sera exécutée. En revanche, la fonction require() affichera une erreur de type FATAL qui interrompt l'exécution du script.

Le choix d'utiliser l'une ou l'autre dépend surtout du contexte. Si le fichier doit obligatoirement être présent pour que le reste du programme fonctionne, alors require() est à préférer sinon un include() sera suffisant.

➢ Tableaux

DEFINITION

Un tableau (array en anglais) est une structure de données de base qui regroupe un ensemble d'éléments (des variables ou autres entités contenant des données), auquel nous avons accès à travers un numéro d'index (ou indice).
Là où une variable stocke une unique valeur, un tableau peut en stocker une multitude. Chaque valeur est alors repérée par un indice (ou encore : une clé, un index).

NOMMAGE

Les règles de nommage pour les tableaux sont les mêmes que pour les variables, pour rappel :

- le nom du tableau commence toujours par le signe dollar « **$** »
- il est obligatoirement une chaîne de caractères alphanumériques
- les caractères spéciaux (espaces, signes de ponctuation, opérateurs, caractères accentués) ne sont pas acceptés, sauf le tiret-bas (underscore _)
- le premier caractère ne peut être un chiffre mais peut en contenir
- le nom est sensible à la casse

DECLARATION

Pour créer un tableau, il suffit d'écrire le nom de la variable suivi de crochets « [] »
Il n'est pas obligatoire de définir l'indice ; si aucun indice n'est indiqué, l'indice suivant disponible sera utilisé. Le premier indice commence à zéro.

```
$tab[] = 'Pomme'; // indice 0
$tab[] = 'Poire'; // indice 1
$tab[] = 'Banane'; // indice 2
$tab[] = 'Fraise'; // indice 3
```

Il est également possible de créer un tableau de façon plus explicite, en utilisant l'instruction array()

```
$tab = array('Pomme','Poire','Banane','Fraise');
```

Un tableau est toujours composé d'indices et de valeurs, voici un exemple dans notre cas :

Indice	*Valeur*
0	Pomme
1	Poire
2	Banane
3	Fraise

Un tableau permet de conserver un ensemble de valeurs et l'indice nous permettra de ressortir l'une d'entre elles.

Exemple (pour ressortir « Poire ») :

```
echo $tab[1] ;
```

TABLEAUX INDEXES ET ASSOCIATIFS

Tableaux indexés

Comme nous l'avons vu dans un tableau indexé, chaque valeur d'un tableau est repérée par un indice (index).

Le premier indice d'un tableau est toujours 0. Ainsi pour un tableau de longueur n (longueur = nombre d'éléments), le premier indice est 0, le dernier indice est n-1.

Tableaux associatifs

Pour mieux s'y retrouver dans les tableaux, il est parfois utile de remplacer l'indice par un nom significatif.

```
$tab['Nom'] = 'Einstein';
$tab['Prenom'] = 'Albert';
```

Ou bien :

```
$tab = array('Nom' => 'Einstein', 'Prenom' => 'Albert');
```

TABLEAUX MULTIDIMENSIONNELS

Un tableau multidimensionnel désigne un ou plusieurs tableaux à l'intérieur d'un tableau principal ou non. Nous parlons alors d'un tableau à plusieurs dimensions.

Exemple

```
$tab = array(0 => array("prenom" => "Amandine", "nom" => "Thoyer"),
1 => array("prenom" => "Jean", "nom" => "Dafard"));
```

Indice	Valeur
0	Array
1	Array

Indice	Valeur
prenom	Jean
nom	Dafard

Indice	Valeur
prenom	Amandine
nom	Thoyer

Affichages des Tableaux

La balise PRE en html affiche le texte contenu dans ses balises tel qu'il est écrit en respectant les espaces, les tabulations et les sauts de ligne. print_r affiche les valeurs du tableau qui seront présentées dans un format affichant également les clés et les valeurs.

```
print "<pre>"; print_r($tab); print "</pre>";
```

De cette manière, nous pouvons visualiser l'intégralité des valeurs contenues dans le tableau.
Pour n'afficher qu'un prénom, il faudra utiliser le code suivant :

```
print  $tab[0]['prenom'] ;
```

Pour afficher tous les prénoms d'un tableau multidimensionnel, il serait judicieux d'utiliser une boucle :

```
for($i=0;  $i<sizeof($tab);  $i++) // tant que $i est inferieur au
nombre d'éléments du tableau...
    {
           print $tab[$i]['prenom']  . '<br />'; // on affiche
l'élément du tableau d'indice $i
    }
```

TRAITEMENT SUR LES TABLEAUX

Les tableaux sont très utilisés en PHP, surtout avec les bases de données ; Bien souvent, les retours de requêtes sur une Base de données se font sous forme de tableaux.

Taille

La fonction suivante renvoie le nombre d'éléments d'un tableau :

```
count (tableau); // similaire à la fonction sizeof().
$tab = array(0 => "un", 1 => "deux", 2 => "trois");
echo count($tab); // renvoie 3
```

Itérations pour les tableaux

PHP propose une structure itérative spécifique aux tableaux. Cette structure permet de réaliser un bloc d'instructions pour chaque élément du tableau.

```
foreach ($tableau as $valeur) {
      instructions ;
}
```

ou, pour récupérer à la fois une clé (ou un indice) associé à sa valeur :

```
foreach ($tableau as $cle => $valeur) {
      instructions ;
}
$tab = array(0 => "un", 1 => "deux", 2 => "trois");
foreach($tab as $val){
      echo "$ val <br />";
}
foreach($tab as $cle => $val){
      echo "$cle : $val <br />";
}
```

Ordre croissant

Pour classer les éléments d'un tableau selon l'ordre croissant de leurs valeurs, nous utilisons la fonction : sort (tableau) ;

```
$tab = array("janvier","février","mars","avril","mai","juin");
print_r($tab);
sort($tab);
print_r($tab); /* affiche Array ( [0] => avril [1] => février [2] =>
janvier [3] => juin [4] => mai [5] => mars ) */
```

Ordre décroissant

Pour classer les éléments dans l'ordre décroissant de leurs valeurs :

```
rsort ($tableau) ;
$tab = array("janvier","février","mars","avril","mai","juin");
rsort($tab);
print_r($tab); /* affiche Array ( [0] => mars [1] => mai [2] => juin
     [3] => janvier [4] => février [5] => avril ) */
```

Tri par clé

Il est également possible de classer les éléments selon l'ordre des clés en utilisant les fonctions *ksort () ; ou krsort () ;*

Affichage en chaine de caractères

```
<?php
$tab = array('arbre', 'maison', 'soleil');
$separation= implode(",", $tab);
echo $separation; // arbre,maison,soleil
```

La fonction *IMPLODE* retourne la représentation en chaîne de caractères de tous les éléments du tableau, dans le même ordre avec la séparation choisie placée entre deux valeurs.
La fonction contraire *EXPLODE* permet de rassembler les éléments d'une chaine de caractère en un tableau *array*.

Naturellement, il existe beaucoup de fonction prédéfinie permettant d'effectuer des traitements sur les tableaux *array*. Ces fonctions peuvent être consultées dans la documentation de PHP.

➢ Superglobales

DEFINITION

Les Superglobales sont des variables internes qui sont toujours disponibles, quel que soit le contexte. Ces variables un peu « spéciale » véhicule sous forme de tableau tout ce qui concerne respectivement, les informations côté serveur, les arguments passés dans l'url, les valeurs passées via un formulaire, les cookies, le transfert de fichiers, les valeurs des variables d'environnement et les sessions.

Les superglobales sont des tableaux associatifs, c'est-à-dire qu'une clé est associée à chaque valeur.

Plusieurs variables prédéfinies en PHP sont "superglobales", en voici la liste :

Nom	Description
○ **$GLOBALS**	Contient toutes les variables disponibles dans l'environnement d'exécution global.
○ **$_SERVER**	Contient les variables fournies par le serveur web.
○ **$_GET**	Contient les variables fournies en paramètre au script via la méthode GET du protocole HTTP.
○ **$_POST**	Contient les variables fournies par un formulaire via la méthode POST du protocole HTTP.
○ **$_FILES**	Contient les variables fournies suite à un chargement de fichier par un formulaire via la méthode POST du protocole HTTP.
○ **$_COOKIE**	Contient les variables fournies par les cookies via le protocole HTTP.
○ **$_SESSION**	Contient les variables de la session en cours dans le script.
○ **$_REQUEST**	Contient les variables fournies au script par n'importe quel mécanisme.
○ **$_ENV**	Contient les variables fournies par l'environnement. Ce peut être des variables du Shell sous lequel s'exécute PHP, les variables CGI...

Il n'est pas possible de créer soi-même des superglobales, nous ne pouvons qu'utiliser celles déjà existantes.

CREATION D'UN FORMULAIRE

Utiliser les formulaires avec PHP permet d'échanger des données avec l'internaute.

C'est-à-dire, lui envoyer des données, mais aussi en recevoir de sa part.

Dès lors, le formulaire est une porte ouverte sur le serveur… Ce qui signifie qu'une personne mal intentionnée peut potentiellement exploiter des failles de sécurité…

De plus, les formulaires exposent vos scripts à de nombreux bogues : certaines données peuvent en effet avoir un comportement handicapant pour votre script. C'est à vous de prévoir les envois de l'internaute et de protéger vos scripts.

Comme tout langage de développement, PHP n'apprécie pas l'approximation.

La création d'un formulaire se fait via le langage HTML et CSS si besoin (pour la mise en forme).

Néanmoins, voici un bref rappel de ce que doit être un formulaire…

Action

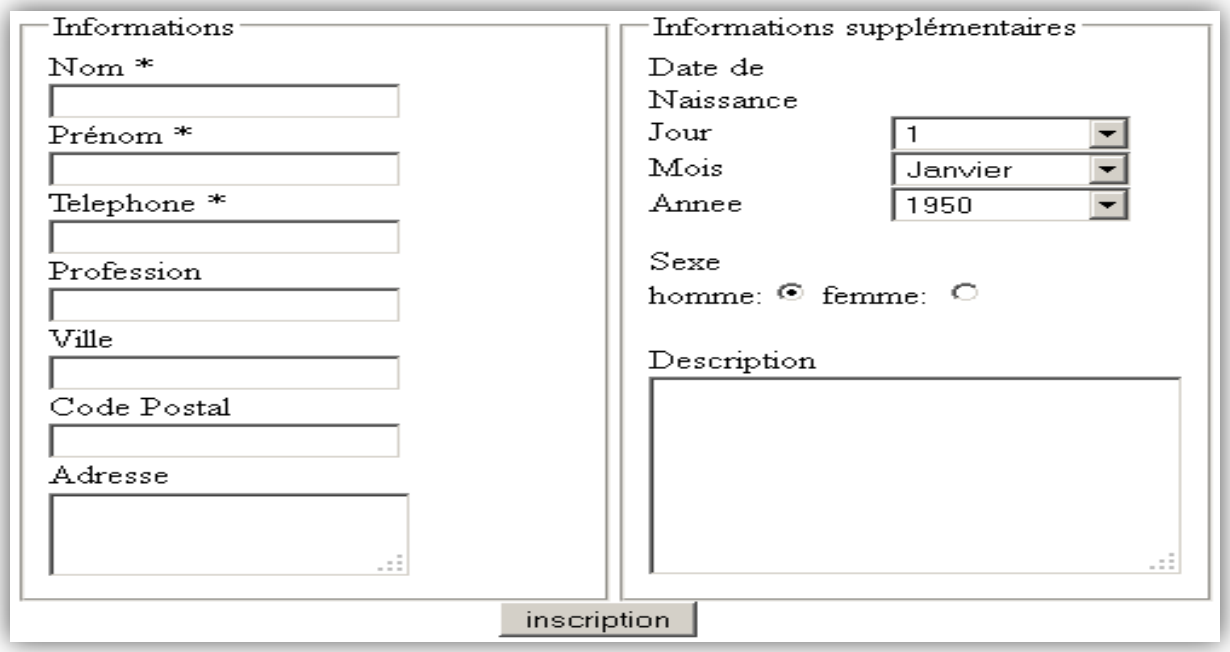
Vous devez indiquer dans un formulaire où vous envoyez les données et comment vous les envoyez.

La balise <form> possède donc un attribut « action », dont la valeur indique une destination (où ?), et un attribut « method », dont la valeur indique la méthode d'envoi des données (comment ?).

L'action du formulaire sera la plupart du temps l'adresse d'une page PHP. La soumission du formulaire renverra alors l'internaute vers cette page PHP qui sera chargée du traitement des données. Il est également possible de faire en sorte que le formulaire envoie les données à la page même qui le contient.

Il ne faut pas oublier l'attribut « name » sur chaque élément du formulaire pour les identifier distinctement.

Voici un formulaire correctement codé en (X)HTML :



RECUPERATION DE DONNEES AVEC LA SUPERGLOBALE POST

Définition

La méthode POST contient les variables fournies par un formulaire via la méthode du protocole HTTP. La variable super globale intégrée $_POST est employée pour récupérer des valeurs dans un formulaire.

Les variables $_POST sont des tableaux (array) de données associatifs et superglobaux. Voici leurs principales caractéristiques :

- Ils sont générés à la volée par PHP avant même que la première ligne du script ne soit exécutée.
- Ce sont des tableaux associatifs comme ceux que nous déclarons traditionnellement. Leur manipulation est exactement semblable à ces derniers. Les clés correspondent aux noms des variables transmises et les valeurs à celles associées à ces variables.
- Ils sont superglobaux, c'est à dire visibles de partout dans le programme (même à l'intérieur d'une fonction utilisateur).
- Ils sont accessibles en lecture et en écriture. Il est donc possible de les modifier.
- L'information envoyée d'un formulaire avec la méthode POST est invisible à l'utilisateur et n'a aucune limite sur la quantité de l'information à envoyer

Exemple

```
<?php
if(isset($_POST['inscription']))
{
   echo "Pseudo : " . $_POST["pseudo"] . "<br />";
   echo "Mot de passe : " . $_POST["mdp"] . "<br />";
}
?>
```

RECUPERATION DE DONNEES AVEC LA SUPERGLOBALE GET

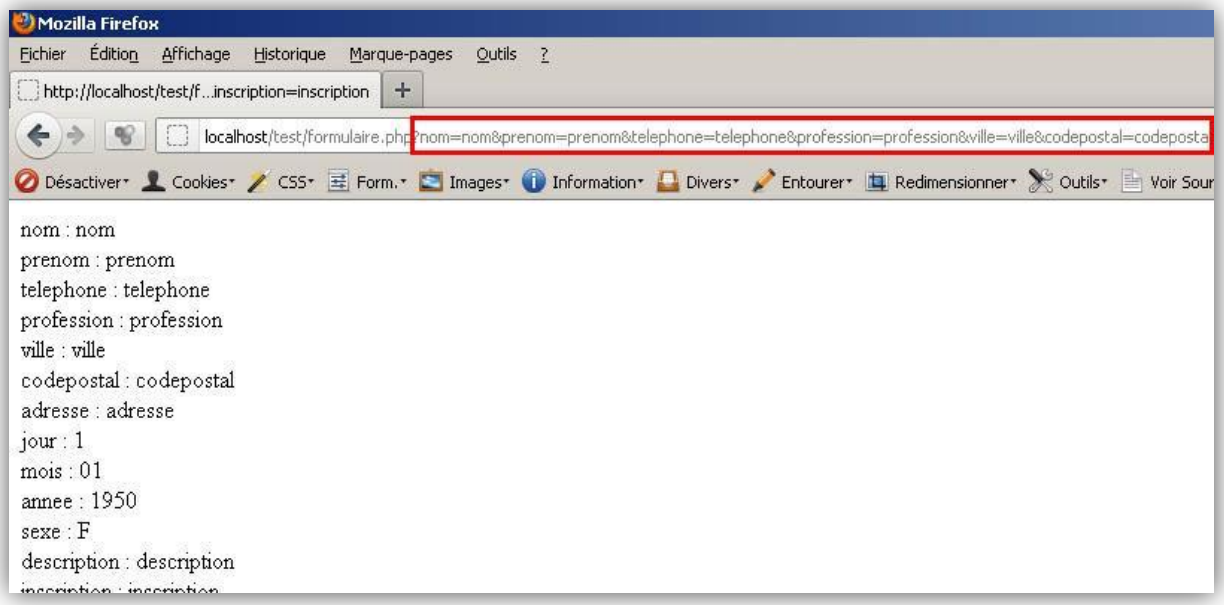
Définition

Cette méthode transmet les données dans l'URL par le protocole HTTP (ce que nous voyons dans la barre d'adresse du navigateur). *$_GET* contient donc les variables fournies en paramètre au script via la méthode GET du protocole HTTP

Exemple

```
<?php
if(isset($_GET['inscription']))
{
	echo "Nom : " . $_GET["nom"] . "<br />";
	echo "Prenom : " . $_GET["prenom"] . "<br />";
	echo "Ville : " . $_GET["ville"] . "<br />";
}
?>
```

Nous obtenons après traitement des données un résultat semblable à celui-ci :



Les données envoyées par le formulaire peuvent transiter d'une page à l'autre par le biais de variables superglobales. Il s'agit de tableaux qui ont une portée supérieure à celle des pages (d'où leur nom).

Utilisation

Cette méthode est surtout utilisée pour passer des informations d'une page à une autre via un lien par exemple.

Pour créer manuellement une chaîne de GET, il suffit de respecter la syntaxe observée précédemment :

- débuter la chaîne par un point d'interrogation « **?** »
- indiquer, pour chaque variable : son nom, suivi du signe égal « **=** », suivi de sa valeur
- séparer chaque variable par une esperluette « **&** ».

Ce qui donne, par exemple :

```
page.php?nom1=valeur1&nom2=valeur2&nom3=valeur3
```

Il est également possible de faire de la réécriture d'url (UrlRewriting) pour éviter d'avoir des chaines trop longues. Pour cela, il est nécessaire de le préciser dans un fichier .htaccess.

LA SUPERGLOBALE $_REQUEST

La superglobale $_REQUEST contient le contenu des deux superglobales $_GET, $_POST, et de la superglobale $_COOKIE.

La variable globale $_REQUEST peut donc être employée pour récupérer des données de formulaire envoyées avec les deux méthodes GET et POST.

➢ Traitement des chaînes

TAILLE

Il est possible de compter le nombre de caractères contenus dans une chaîne. Nous utilisons pour cela la fonction : strlen ("chaîne").

Exemple

```
$a = 'Cette fonction compte le nombre de caractères d\'une chaîne';
echo strlen($a); // affiche 58
```

Note : bien entendu, le caractère d'échappement n'est pas comptabilisé ici.

POSITION

PHP permet de vérifier la présence d'un caractère ou d'une « sous-chaîne » dans une chaîne (cette fonction est sensible à la casse) :

```
$a = "test@site.fr";
$b = "test-at-site.fr";
$c = "@site.fr";
var_dump(strpos($a,"@"));
var_dump(strpos($b,"@"));
var_dump(strpos($c,"@"));
```

Le premier appel à la fonction retourne l'entier 4, car le caractère @ est à l'indice 4 dans la chaîne (nous comptons à partir de 0). Le second appel renvoie le booléen FALSE, car le caractère @ n'existe pas dans la chaîne. Pour le troisième appel, il ne faut pas confondre 0 (présent en première position) et FALSE (absent). PHP dispose d'une fonction identique mais insensible à la casse : stripos ("chaîne", "sous-chaîne");

CHANGEMENT DE CASSE

Pour convertir en casse haute (majuscule) : strtoupper ("chaîne");

Pour convertir en casse basse (minuscule) : strtolower ("chaîne");

Pour mettre en majuscule le premier caractère de la chaîne : ucfirst ("chaîne");

Pour mettre en majuscule le premier caractère de chaque mot : ucwords ("chaîne");

```
$a = "casse haute OU casse basse";
echo strtoupper($a)."<br />"; // CASSE HAUTE OU CASSE BASSE
echo strtolower($a)."<br />"; // casse haute ou casse basse
```

COUPER UNE CHAINE

Nous utilisons pour cela la fonction : substr qui retourne un segment de chaîne.

Quelques exemples :

```
$rest = substr("abcdef", -1);    // retourne "f"
```

```
$rest = substr("abcdef", 0, -1);  // retourne "abcde"
```

```
$texte = "Lorem Ipsum is simply dummy text of the printing and
typesetting industry. Lorem Ipsum has been the industry's standard
dummy text ever since the 1500s.”;

$coupe = substr($texte, 0, 20) . "...<a href=''> Lire la suite
</a>";

print $coupe; // Retourne une partie du texte seulement, avec un
lien pour lire la suite de l’article.
```

Il existe beaucoup d'autres fonctions prédéfinies permettant d'effectuer des traitements sur les chaines de caractères. Ces fonctions peuvent être consultées dans la documentation de PHP.

➢ Fonctions prédéfinies

DEFINITION

En informatique, une fonction est une portion de code représentant un sous-programme, qui effectue une tâche ou un calcul relativement indépendant du reste du programme. En programmation impérative, une fonction est une séquence d'instructions réalisant une certaine tâche. En programmation fonctionnelle, la fonction est l'objet de base, qui permet de découper le problème global en plus petits calculs. Nous utilisons parfois le synonyme routine, notamment à propos des fonctions bas-niveau des systèmes d'exploitation. Une fonction a une entrée (les arguments, ou paramètres que nous lui donnons), exécute un travail sur ces arguments, et, généralement, retourne une valeur (la sortie). Quand une fonction prend des arguments en entrée, elle en prend une copie, ce ne sont pas les véritables variables qui le sont. Pour ce faire, les fonctions demandent l'adresse mémoire des variables, plutôt que leurs valeurs. C'est un pointeur (ou référence en C++ ou PHP), dans ce cas, la fonction modifie la variable à l'intérieur d'elle-même et retourne généralement un code d'erreur, ou 0 si tout c'est bien terminé. Une fonction peut aussi se passer de renvoyer une valeur (elle est de type 'void' en C/C++).

NOMBRE

Pour arrondir un nombre à l'entier le plus proche, nous utilisons la fonction round().
Cette fonction accepte un paramètre supplémentaire, afin de spécifier le nombre de décimales pour l'arrondi (ce paramètre peut être négatif pour un arrondi à l'unité, à la dizaine, à la centaine, etc.) :

Exemple

```
$a = 1252.43587;
echo round($a); // affiche 1252
echo round($a,2);     // affiche 1252,44
echo round($a,-2);    // affiche 1300
echo ceil($a);        // affiche 1253
echo floor($a); // affiche 1252
```

VALEUR ALEATOIRE

Obtenir une valeur aléatoire est utile à de nombreux scripts. Par exemple, pour afficher une bannière publicitaire choisie aléatoirement, pour proposer un style CSS alternatif, etc.

La fonction suivante renvoie un entier compris entre les paramètres "minimum" et "maximum" (inclus) :

Exemple

```
$a = mt_rand(1,10);
echo $a;
```

AFFICHER UNE DATE

Pour afficher une date au format voulu, nous utilisons la fonction :

```
print date("Y-m-d");
```

TIMESTAMP

Le timestamp est un nombre qui indique le nombre de secondes qui se sont écoulées depuis le premier janvier 1970. Pour afficher le timestamp de la seconde actuelle, nous utilisons la fonction time() de PHP.

Exemple

```
<?php
     echo time(); //Affiche le nombre de secondes écoulées depuis le
1er janvier 1970
?>
```

ENVOI DE MAIL

L'envoi de mail s'effectue en PHP de manière très simple, grâce à la fonction mail() :

mail ("destinataire", "sujet", "message", "entête") ;

Son utilisation se présente essentiellement de la façon suivante :

Exemple

```
$email="nom@ifocop.fr";
$sujet="bonjour";
$message="comment allez-vous ?";
mail($email,$sujet,$message);
```

Avant d'utiliser la méthode « mail » de php, il faut configurer le fichier php.ini de wamp de manière à renseigner le smtp entre autre.

Le Simple Mail Transfer Protocol (littéralement « Protocole simple de transfert de courrier »), généralement abrégé SMTP, est un protocole de communication utilisé pour transférer le courrier électronique (courriel) vers les serveurs de messagerie électronique.

CREATION DE FICHIERS, LECTURE, ECRITURE

PHP permet de créer des fichiers « à la volée » via ses fonctions prédéfinies, cela peut être intéressant pour conserver une information, ou un ensemble de données quand nous n'avons pas la possibilité de se connecter à une base de données pour le faire.

Ecriture

```
$f = fopen("liste.txt","a");
     fwrite($f," test ");
$f = fclose($f);
```

fopen permet d'ouvrir un fichier en écriture, le mode « a » se charge de créer le fichier s'il n'existe pas.

fwrite permet d'écrire à l'intérieur du fichier (représenté par la variable $f).

fclose permet de fermer le fichier représenté par le pointeur.

Lecture

```
$nom_fichier="liste.txt";
$fichier=file($nom_fichier);
for( $i = 0 ; $i < count($fichier) ; $i++ )
{
    echo $fichier[$i]."</br>";
}
```

File permet de lire un fichier et renvoie le résultat dans un tableau (array). Si une erreur survient (fichier inexistant), *file()* retournera FALSE.

DIVERS

La fonction isset détermine si une variable est affectée (« si elle existe », « si elle est définie »).
Ses valeurs de retour sont TRUE si la variable a une valeur autre que null, sinon elle renverra FALSE.

Exemple

```
$a = 'Hello';
if(isset($a)) echo "a existe";
if(isset($b)) echo "b existe";
```

La fonction empty détermine si une variable contient une valeur nulle. Cette instruction renvoie vrai ou faux.

Exemple

```
$var1 = 0;
$var2 = 10;
$var3 = ""; // chaîne vide
if (empty($var1)) echo 'var1 vaut soit 0, vide, ou non définie';
if (empty($var2)) echo 'var2 vaut soit 0, vide, ou non définie';
if (empty($var3)) echo 'var3 vaut soit 0, vide, ou non définie';
if (empty($var4)) echo 'var4 vaut soit 0, vide, ou non définie';
```

➢ Fonctions utilisateur

DEFINITION

PHP met à notre disposition un ensemble très vaste de fonctions prédéfinies, couvrant les besoins de base propres à toute programmation. Nous en avons déjà vu un certain nombre.
A l'inverse et en complément, une fonction utilisateur est une fonction écrite par le développeur.
Une fonction est un ensemble d'instructions exécutées à l'appel de celle-ci.
L'intérêt des fonctions est de permettre la réutilisation à l'envi d'un ensemble d'instructions. Cela diminue alors le nombre de lignes de codes (donc le poids des fichiers) et surtout cela simplifie la vie du développeur.
Une fonction se fait toujours en deux étapes : la déclaration, puis l'exécution.

DECLARATION

Le nommage des fonctions suit les mêmes règles que pour les variables.
Les arguments sont traités un peu plus loin ; ils sont optionnels, cependant la présence des parenthèses, même vides, est obligatoire.

```
function nom_de_fonction ($argument1, ..., $argumentN) {
      instructions ;
}
```

EXECUTION

Une fonction déclarée n'est pas active par défaut, pour effectuer le traitement des instructions de la fonction, il est nécessaire de l'exécuter (l'appeler).
Il est possible d'invoquer plusieurs fois une fonction (c'est même l'intérêt).
Une fonction déclarée sans argument sera exécutée sans argument néanmoins, rappelez-vous que la présence des parenthèses est obligatoire.

```
nom_de_fonction ($argument1, ..., $argumentN) ;
```

FONCTION SANS ARGUMENT

Voici une fonction simpliste qui permet lors de son appel de tirer un trait sur la page web grâce à la balise « <hr /> » du html.

Exemple

```
function separation() // sans argument
{
      echo "<hr /><hr /><hr />";
}
separation(); // execution n°1
separation(); // execution n°2
```

FONCTION AVEC ARGUMENTS

Lorsqu'on déclare une fonction, celle-ci semble calibrée pour agir toujours de façon identique. Il est pourtant parfois utile, voire nécessaire, de lui permettre de s'adapter… cette flexibilité révèle toute la richesse des fonctions.

Exemple : une fonction de conversion nous sert à convertir une somme. Lorsque je l'utilise, j'ai besoin de lui fournir le taux de change du jour. Le taux de change, variable selon le jour, sera alors un argument de la fonction.

Les arguments sont des paramètres fournis à la fonction et lui permettent de compléter ou modifier son comportement initialement prévu.

Exemple

```
function bonjour($qui)
{
      echo "Bonjour ".$qui."<br />\n";
}
$prenom = "Etienne";
bonjour("Pierre"); // execution n°1
bonjour("Jacques"); // execution n°2
bonjour($prenom); // execution n°3
```

Lors de la première exécution de la fonction, la variable *$qui* prend la valeur : « Pierre ».

Lors de la seconde exécution de la fonction, la variable *$qui* prend la valeur : « Jacques ».

Lors de la dernière exécution de la fonction, la variable *$qui* prend la valeur : « Etienne » transmise par la variable *$prenom*.

FONCTION AVEC ARGUMENTS FACULTATIFS

Si vous désirez créer une fonction avec certains paramètres facultatifs, il suffit d'attribuer aux paramètres une valeur par défaut.

```
function bonjour($qui=null)
{
      echo "Bonjour ".$qui."<br />";
}
bonjour("Jean"); // affiche "Bonjour Jean"
bonjour(); // affiche "Bonjour"
```

FONCTION AVEC PLUSIEURS ARGUMENTS

Dans le cas où la fonction est destinée à recevoir plusieurs arguments, il est impératif de respecter le nombre et l'ordre des arguments qu'elle attend.

Exemple

```
meteo("hiver","2"); // exécution n°1
function meteo($saison, $temperature) // déclaration
{
      echo "Nous sommes en ".$saison." et il fait: " . $temperature
." degre(s)<br /> \n";
}
meteo("été","26"); // exécution n°2
```

Le \n dans une instruction echo permet de faire un retour chariot dans le code HTML (ne pas confondre avec un retour chariot en HTML :
)

RETOUR DE VALEURS

Une fonction peut retourner une valeur ; il faut pour cela recourir à l'instruction return suivie de la valeur ou de la variable à retourner.

Attention : Le mot-clé return met immédiatement fin à l'exécution de la fonction.

Exemple

```
function joursemaine()
{
      $jour = "lundi"; // variable locale.
      return $jour; // valeur de retour (à ce moment là nous quittons la fonction)
}
// echo $jour; // ne fonctionne pas car cette variable n'est connue qu'à l'intérieur de la fonction
$recup = joursemaine(); // on récupèrons la valeur de retour soit "lundi"
echo $recup; // on affiche "lundi"
```

VARIABLES GLOBALES, VARIABLES LOCALES

Définition

Les variables définies à l'intérieur d'une fonction sont dites locales. C'est-à-dire que les variables déclarées ou modifiées à l'intérieur d'une fonction ne seront pas accessibles à l'extérieur de celle-ci. Les variables définies dans le corps du script sont dites globales. Elles sont accessibles dans le script courant, mais étant déclarées en dehors des fonctions, elles ne seront pas accessibles à l'intérieur de celles-ci.

Rendre globale une variable locale

Ainsi, pour pouvoir utiliser une variable globale à l'intérieur d'une fonction, il est nécessaire de la déclarer comme étant globale (global) en début de fonction.

Il n'est pas nécessaire d'utiliser le mot-clé global devant chaque variable à déclarer comme globale, celui-ci permet de déclarer plusieurs variables en une seule fois :

Exemple

```
function aireRectangle() {
      global $largeur, $hauteur, $surface;
      $surface = $largeur * $hauteur;
}
$largeur = 12;
$hauteur = 20;
aireRectangle();
echo $surface; // affiche "240"
```

A vrai dire, la logique (la largeur et la hauteur sont des paramètres, la surface une valeur retournée) voudrait que l'on écrive ce dernier exemple sous la forme :

```
function aireRectangle($largeur, $hauteur) {
      return $largeur * $hauteur;
}
$surface = aireRectangle(12, 20);
echo $surface;
```

➢ Classes et Objets

QU'EST-CE QU'UN OBJET ?

Un objet est un conteneur symbolique, qui possède sa propre existence et englobe des caractéristiques, états et comportements et qui, par métaphore, représente quelque chose de tangible du monde réel manipulé par informatique.
En programmation orientée objet, un objet est créé à partir d'un modèle appelé classe, duquel il hérite les comportements et les caractéristiques. Les comportements et les caractéristiques sont typiquement basés sur celles propres aux choses qui ont inspiré l'objet : une personne, un dossier, un produit, une lampe, une chaise, un bureau. Tout peut être objet.

QU'EST-CE QU'UNE CLASSE

Les classes sont présentes pour « fabriquer » des objets.
En programmation orientée objet, un objet est créé sur le modèle de la classe à laquelle il appartient.

Exemple

Prenons l'exemple le plus simple du monde : les gâteaux et leur moule. Le moule, il est unique. Il peut produire une quantité infinie de gâteaux. Dans ces cas-là, les gâteaux sont les objets et le moule est la classe. La classe est présente pour produire des objets. Elle contient le plan de fabrication d'un objet et nous pouvons nous en servir autant que nous le souhaitons afin d'obtenir une infinité d'objets.

Autre Exemple

Etant donné qu'une classe est le modèle de quelque chose que nous voudrions construire.
Le plan de construction d'une maison qui réunit les instructions destinées à la construction est donc une classe. Cependant le plan n'est pas une maison.
La maison est un objet qui a été « fabriqué » à partir de la classe (le plan).
A partir du plan (la classe) nous pouvons construire une ou plusieurs maisons (l'objet).

Le vocabulaire diffère légèrement en Programmation Orientée Objet, nous ne parlerons plus de fonctions mais de méthodes, nous ne parlerons plus de variables mais d'attributs (ou propriétés).

Une classe est une entité regroupant un certain nombre d'attributs et de méthodes que tout objet issu de cette classe possèdera.

DIFFERENCE ENTRE UNE CLASSE ET UN OBJET

- La classe est un plan, une description de l'objet. Sur le plan de construction d'une voiture nous y retrouverons le moteur ou encore la couleur de la carrosserie.
- L'objet est une application concrète du plan. L'objet est la voiture. Nous pouvons créer plusieurs voitures basées sur un plan de construction.

Nous pouvons donc créer plusieurs objets à partir d'une classe.

MANIPULATION

```
class Contenant
{
	public $prenom = "Julien";
	public $age = 25;
	public function pays()
	{
		return "France";
	}
}
$objet = new Contenant();
print "<pre>"; var_dump($objet); print "</pre>";
echo $objet->prenom . "<br />";
echo $objet->age . "<br />";
echo $objet->pays() . "<br />";
```

Sur cet exemple, la class Contenant est instanciée via le mot-clé « new ».

Nous faisons appel à la fonction var_dump() pour en visualiser le contenu.

L'opérateur « -> » permet d'accéder à une propriété ou une méthode de l'objet.

* Ce support « PHP – partie 1 » constitue l'introduction au PHP pour être en adéquation avec le module de cours dispensé à l'IFOCOP. D'autres supports de cours sont disponibles et approfondissent d'avantage le langage ainsi que ses possibilités.