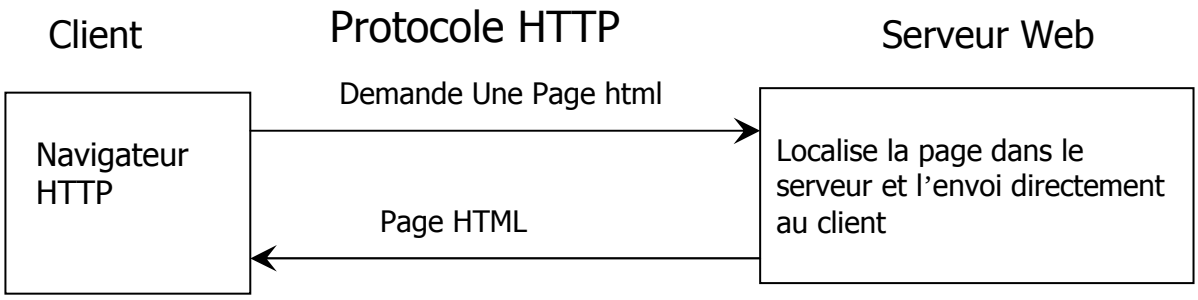
PHP
Développement Web dynamique

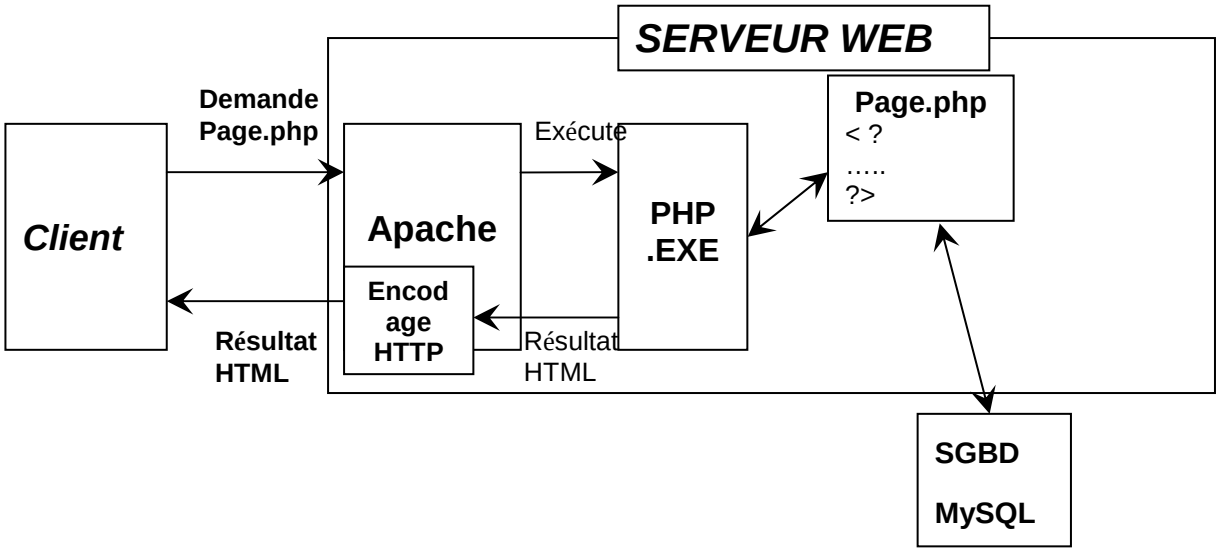
I- Concepts de base des sites web

1- Principe d'un site web statique



Lorsqu'un client demande une page HTML au serveur, en cliquant sur un lien hypertexte ou en saisissant l'URL de cette page, le serveur web la cherche dans son disque dur, et l'envoi telle quelle au client. Dans ce cas, le serveur n'effectue aucun traitement, ce qui fait que le client reçoit le même code de la page qui est hébergée dans le serveur.

2- cas d'un site web dynamique.



Dans le cas d'un site web dynamique avec PHP, le client demande une page PHP au serveur web (Apache). Ce dernier cherche la page dans son disque dur, si elle la trouve, Il charge le module PHP et demande à celui-ci d'exécuter l'ensemble des instructions qui se trouve dans cette page. Tout script compris entre les balises < ? et ?> est considéré comme étant script php et sera exécuté dans le serveur. Dans le cas contraire, il sera considéré comme étant script Client (HTML, javascript, CSS..) et sera envoyé au navigateur. Le serveur envoi

en fin de compte une page HTML comme réponse à la requête. Généralement l'exécution du script php entraîne la connexion à une base de données pour consulter et mettre à jour des données dans celle-ci.

Il est important de noter que les scripts php sont exécuté coté serveur et que le résultat envoyé au navigateur web est formé par du code HTML, javascript, css ou XML et sera interprété par le client (Browser)

3- Caractéristiques d'un script PHP.

- Généralement les scripts php sont stockés dans des fichiers texte ayant pour extension .php. Toute fois cette extension peut être changée en modifiant la configuration du serveur web Apache. Par fois les hébergeurs PHP préfèrent spécifier la version du php dans l'extension. Ainsi, on trouve des fichiers à extension .php3, ?php4 ou .php5.
- Les scripts php sont stockés dans des fichiers texte qui peuvent contenir du code HTML et du code PHP. Tout script qui se trouve entre les balises < ? et ?> est considéré comme script PHP. Il sera alors exécuté par le serveur. Et tout code en dehors de ces balise est considéré comme étant un script statique (HTML) et sera envoyé au client.

4- Requête http.

Le navigateur Web et un serveur Web communique en utilisant le protocole HTTP. Chaque requête client est associée à un script dans le serveur. L'appel à une page web par le client peut être effectué par les manières suivantes :

- Ecrire directement l'adresse de la page dans la zone URL du navigateur.
- En cliquant sur un lien hypertexte de la page courante.
- En cliquant sur le bouton submit d'un formulaire de la page courante.
- Provoquée par un timer défini dans la page courante.

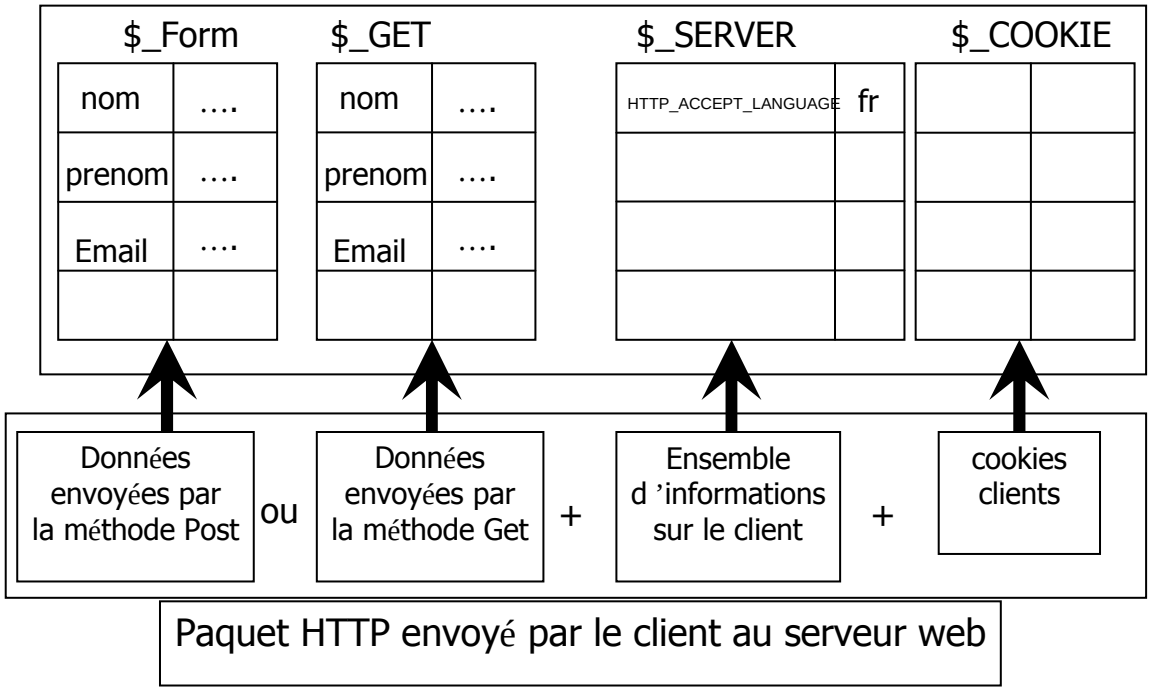
La requête http contient des informations envoyées par le client. Ces informations peuvent être:

1. Des données envoyées par la méthode GET : un formulaire peut utiliser la méthode GET. Dans ce cas toutes les données du formulaire sont envoyées avec l'URL de la requête (http://localhost/affiche.php?nom=a&prenom=b&email=c.....). La méthode GET est également employée par le navigateur lorsque la requête est associée à un lien hypertexte. Cette méthode présente deux limites. La première c'est que avec GET, on ne peut pas envoyer plus de 1000 caractères (longueur maximale de l'URL). Le deuxième inconvénient est que toutes les données envoyées sont visible à l'utilisateur dans l'url du navigateur, ce qui présente un problème au niveau de sécurité.
2. Des données envoyées par la méthode POST : la méthode POST est généralement utilisées par les formulaires. L'avantage de cette méthode par rapport à GET, c'est qu'on peut envoyer autant de caractères que l'on souhaite. En plus les données sont encapsulées dans le corps de la requête http. Ce qui fait que les données envoyées ne sont pas visibles au niveau de l'URL du navigateur.
3. Dans la requête HTTP, le client envoie toujours un ensemble d'informations sur sa machine (adresse IP, langue, type de navigateur….). ces informations, standardisées par le protocole http, permettent au serveur de connaître mieux le client qui a envoyé la requête.
4. Le client envoie également au serveur des données enregistrées préalablement, par le serveur, dans le poste du client sous forme de COOKIES. Ces cookies servent aux serveurs web de stocker des informations utiles, relatives à chaque client, dans le poste de celui-ci. Ces informations seront lues par le serveur lorsque le client se reconnecte au même site. Le serveur peut également stocker des informations relatives à chaque client durant toute la durée de sa connexion au site, cette fois ci dans la

RAM du serveur, en utilisant ce qu'on appel des variables sessions. Nous reviendrons à ce concept très important plus tard.

4- Récupérer les données de la requête client avec PHP.

PHP dispose des collections suivantes pour récupérer les données de la requête



- La collection $_POST permet de stocker les données envoyées par la méthode post. Si on suppose, par exemple, qu'un client a envoyé un formulaire qui contient les champs nom, prenom et email, en utilisant la méthode POST, nous pouvons récupérer, coté serveur avec un script php en écrivant :

```
< ?
 $N=$_POST['nom'];
 $P=$_POST['prenom'];
 $E=$_POST['email'];
?>
```

- La collection $_GET permet de stocker les données envoyées par la méthode GET. Si on suppose, par exemple, qu'un client a envoyé une requête GET de type http://localhost/affiche.php?nom=azerty, nous pouvons écrire l'instruction php suivante pour récupérer la valeur du paramètre nom de l'URL.

```
< ?
 $N=$_GET['nom'];
?>
```

- La collection $_SERVER peut être utilisée pour récupérer la valeur des entêtes http envoyées dans la requête du client. Ces informations permettent de décrire les caractéristiques de l'environnement du client et du serveur. Exemples :

- $langue=$_SERVER['HTTP_ACCEPT_LANGUAGE']; : pour stocker dans la variable $lanhue, le code de la langue du navigateur client.
- $ip= $_SERVER['REMOTE_ADDR'] ; : pour stocker dans la variable ip, l'adresse IP du client.

- La collection $_COOKIE permet de lire et de créer les cookies. Pour lire la valeur de la clé nom d'un cookie nommé « utilisateur », on peut écrire :
  - $n=$_COOKIE['utilisateur']['nom'];

4- Envoyer une réponse http au client avec PHP.

La réponse http envoyée par le serveur client se compose de deux parties :

- La première partie est formée par les entêtes http. Ces entêtes permettent de fournir au client des informations diverses. Ces informations peuvent, par exemple, donner des indications sur le serveur, sur le type du document envoyé par le serveur, faire une redirection, créer des cookies, gestion du cache client etc…

  PHP dispose de plusieurs manières pour définir les entêtes de la réponse http. L'instruction la plus générale est header(). Cette fonction reçoit en argument une chaîne de caractères qui indique les noms et les valeurs des entêtes http. Par exemple, pour envoyer une réponse http qui indique au client de faire une redirection vers une autre page nommée affiche.php, on peut écrire :

  ```
  header("location:affiche.php");
  ```

  Location représente le nom de l'entête http et affiche.php indique la valeur de cette enête. En recevant, dans l'entête de la réponse http cette information, le client peut comprendre qu'il doit envoyer automatiquement une autre requête vers affiche.php.
- La deuxième partie de la réponse http est formée par le contenue de la page web envoyée par le serveur. C'est dans cette partie qu'on trouverait tout le code HTML de la page qui sera affichée par le navigateur.

  Pour stocker des informations dans le corps de la réponse, on utilise la fonction echo (). Par exemple, pour envoyer le texte « Bonjour » affiché en gras au navigateur, on écrit :

  ```
  echo "<b>Bonjour</b>" ;
  ```

II- Bases du langage PHP.

Généralement, la syntaxe du langage php ressemble à celle du langage C.

1- Commentaires

Comme dans le langage C, une ligne de commentaire est précédée par le double caractère slash //. Et pour mettre en commentaire un ensemble de ligne, on les délimite par /* et */

Exemples :

```
// Une ligne de commentaire
/*
 Plusieurs lignes
de commantaires
*/
```

2- Variables :

Une variable est un objet repéré par son nom, pouvant contenir des données, qui pourront être modifiées lors de l'exécution du programme. Les variables en langage PHP peuvent être de trois types:

- scalaires

- tableaux
- tableaux associatifs

Quelque soit le type de variable, son nom doit obligatoirement être précédé du caractère dollar ($). Contrairement à de nombreux langages de programmation, comme le langage C, les variables en PHP n'ont pas besoin d'être déclarées, c'est-à-dire que l'on peut commencer à les utiliser sans en avoir averti l'interpréteur précédemment, ainsi si la variable existait précédemment, son contenu est utilisé, sinon l'interpréteur lui affectera la valeur en lui assignant 0 par défaut. De cette façon si vous ajoutez 3 à une nouvelle variable (non définie plus haut dans le code), sa valeur sera 3

Nommage des variables

Avec PHP, les noms de variables doivent répondre à certains critères:

- un nom de variable doit commencer par une lettre (majuscule ou minuscule) ou un "_" (pas par un chiffre)
- un nom de variable peut comporter des lettres, des chiffres et le caractère _ (les espaces ne sont pas autorisés!)

Nom de variable correct	Nom de variable incorrect	Raison
$Variable	$Nom de Variable	comporte des espaces
$Nom_De_Variable	$123Nom_De_Variable	commence par un chiffre
$nom_de_variable	$toto@mailcity.com	caractère spécial @
$nom_de_variable_123	$Nom-de-variable	signe - interdit
$nom_de_variable	nom_de_variable	ne commence pas par $

Les noms de variables sont sensibles à la casse (le langage PHP fait la différence entre un nom en majuscule et un nom en minuscules), il faut donc veiller à utiliser des noms comportant la même casse! Toutefois, les noms de fonctions font exception à cette règle...

Variables scalaires

Le langage PHP propose trois types de variables scalaires:

- entiers: nombres naturels sans décimale (sans virgule)
- réels: nombres décimaux (on parle généralement de type *double*, car il s'agit de nombre décimaux à double précision)
- chaines de caractères: ensembles de caractères

Il n'est pas nécessaire en PHP de typer les variables, c'est-à-dire de définir leur type, il suffit de leur assigner une valeur pour en définir le type:

- entiers: nombre sans virgule
- réels: nombres avec une virgule (en réalité un point)
- chaines de caractères: ensembles de caractères entre guillemets simples ou doubles

Instruction	Type de la variable
$Variable = 0;	type entier
$Variable = 12;	type entier

$Variable = 0.0;	type réel
$Variable = 12.0;	type réel
$Variable = "0.0";	type chaîne
$Variable = "Bonjour tout le monde";	type chaîne

Il existe des caractères repérés par un code ASCII spécial permettant d'effectuer des opérations particulières. Ces caractères peuvent être représentés plus simplement en langage PHP grâce au caractère '\' suivi d'une lettre, qui précise qu'il s'agit d'un caractère de contrôle:

Caractère	Description
\"	guillemet
\\	barre oblique inverse (antislash)
\r	retour chariot
\n	retour à la ligne
\t	tabulation

En effet, certains de ces caractères ne pourraient pas être représentés autrement (un retour à la ligne ne peut pas être représenté à l'écran). D'autre part, les caractères \ et " ne peuvent pas faire partie en tant que tel d'une chaîne de caractère, pour des raisons évidente d'ambigüité...

Variables tableaux

Les variables, telles que nous les avons vues, ne permettent de stocker qu'une seule donnée à la fois. Or, pour de nombreuses données, comme cela est souvent le cas, des variables distinctes seraient beaucoup trop lourdes à gérer. Heureusement, PHP propose des structures de données permettant de stocker l'ensemble de ces données dans une "variable commune". Ainsi, pour accéder à ces valeurs il suffit de parcourir la variable de type complexe composée de « variables » de type simple.

Les tableaux stockent des données sous forme de liste. Les données contenues dans la liste sont accessibles grâce à un index (un numéro représentant l'élément de la liste). Contrairement à des langages tels que le langage C, il est possible de stocker des éléments de types différents dans un même tableau.

Ainsi, pour désigner un élément de tableau, il suffit de faire suivre au nom du tableau l'indice de l'élément entre crochets:

```
$Tableau[0] = 12;
$Tableau[1] = "CCM";
```

Avec PHP, il n'est pas nécessaire de préciser la valeur de l'index lorsque l'on veut remplir un tableau, car il assigne la valeur 0 au premier élément (si le tableau est vide) et incrémente les indices suivants. De cette façon, il est facile de remplir un tableau avec des valeurs. Le code précédent est équivalent à:

```
$Tableau[] = 12;
$Tableau[] = "CCM";
```

- Les indices de tableau commencent à zéro
- tous les types de variables peuvent être contenus dans un tableau

Lorsqu'un tableau contient d'autres tableaux, on parle de tableaux multidimensionnels. Il est possible de créer directement des tableaux multidimensionnels en utilisant plusieurs paires de crochets pour les index (autant de paires de crochets que la dimension voulue). Par exemple, un tableau à deux dimensions pourra être déclaré comme suit:

```
$Tableau[0][0] = 12;
$Tableau[0][1] = "CCM";
$Tableau[1][0] = 1245.652;
$Tableau[1][1] = "Au revoir";
```

Variables tableaux associatifs

PHP permet l'utilisation de chaînes de caractères au lieu de simples entiers pour définir les indices d'un tableau, on parle alors de *tableaux associatifs*. Cette façon de nommer les indices peut parfois être plus agréable à utiliser:

```
$Toto["Age"] = 12;
$Toto["Adresse"] = "22 rue des bois fleuris";
```

Portée (visibilité) des variables

Selon l'endroit où on déclare une variable, celle-ci pourra être accessible (visible) de partout dans le code ou bien que dans une portion confinée de celui-ci (à l'intérieur d'une fonction par exemple), on parle de *portée* (ou *visibilité*) d'une variable.

Lorsqu'une variable est déclarée dans le code même, c'est-à-dire à l'extérieur de toute fonction ou de tout bloc d'instructions, elle est accessible de partout dans le code (n'importe quelle fonction du programme peut faire appel à cette variable). On parle alors de **variable globale**

Lorsque l'on déclare une variable à l'intérieur d'un bloc d'instructions (entre des accolades), sa portée se confine à l'intérieur du bloc dans lequel elle est déclarée.

- Une variable déclarée au début du code, c'est-à-dire avant tout bloc de donnée, sera globale, on pourra alors les utiliser à partir de n'importe quel bloc d'instructions
- Une variable déclarée à l'intérieur d'un bloc d'instructions (dans une fonction ou une boucle par exemple) aura une portée limitée à ce seul bloc d'instructions, c'est-à-dire qu'elle est inutilisable ailleurs, on parle alors de variable locale

D'une manière générale il est préférable de donner des noms différents aux variables locales et globales pour des raisons de lisibilité et de compréhension du code.

Définition de constantes

Une constante est une « variable » dont la valeur est inchangeable lors de l'exécution d'un programme. Avec PHP, les constantes sont définies grâce à la fonction *define()*. la syntaxe de la fonction *define()* est la suivante:

```
define("Nom_de_la_variable", Valeur);
```

3- les opérateurs :

Opérateurs arithmétiques

Les opérateurs de calcul permettent de modifier mathématiquement la valeur d'une variable

Opérateur	Dénomination	Effet	Exemple	Résultat (pour *x=7*)

+	opérateur d'addition	Ajoute deux valeurs	$x+3	10
-	opérateur de soustraction	Soustrait deux valeurs	$x-3	4
*	opérateur de multiplication	Multiplie deux valeurs	$x*3	21
/	plus: opérateur de division	Divise deux valeurs	$x/3	2.3333333
=	opérateur d'affectation	Affecte une valeur à une variable	$x=3	Met la valeur 3 dans la variable $x
%	opérateur modulo	Donne le reste de la division entière entre 2 nombres	$x%3	1

Les opérateurs d'assignation

Ces opérateurs permettent de simplifier des opérations telles que *ajouter une valeur dans une variable et stocker le résultat dans la variable*. Une telle opération s'écrirait habituellement de la façon suivante par exemple: *$x=$x+2*
Avec les opérateurs d'assignation il est possible d'écrire cette opération sous la forme suivante: *$x+=2*
Ainsi, si la valeur de x était 7 avant opération, elle sera de 9 après...

Les autres opérateurs du même type sont les suivants:

Opérateur	**Effet**
+=	addition deux valeurs et stocke le résultat dans la variable (à gauche)
-=	soustrait deux valeurs et stocke le résultat dans la variable
***=**	multiplie deux valeurs et stocke le résultat dans la variable
/=	divise deux valeurs et stocke le résultat dans la variable
%=	donne le reste de la division deux valeurs et stocke le résultat dans la variable
|=	Effectue un OU logique entre deux valeurs et stocke le résultat dans la variable
^=	Effectue un OU exclusif entre deux valeurs et stocke le résultat dans la variable
&=	Effectue un Et logique entre deux valeurs et stocke le résultat dans la variable
.=	Concatène deux chaînes et stocke le résultat dans la variable

Les opérateurs d'incrémentation

Ce type d'opérateur permet de facilement augmenter ou diminuer d'une unité une variable. Ces opérateurs sont très utiles pour des structures telles que des boucles, qui ont besoin d'un compteur (variable qui augmente de un en un).

Un opérateur de type *$x++* permet de remplacer des notations lourdes telles que *$x=$x+1* ou bien *$x+=1*

Opérateur	Dénomination	Effet	Syntaxe	Résultat (avec x valant 7)
++	Incrémentation	Augmente d'une unité la variable	$x++	8
--	Décrémentation	Diminue d'une unité la variable	$x--	6

Les opérateurs de comparaison :

Opérateur	Dénomination	Effet	Exemple	Résultat
= = **A ne pas confondre avec le signe d'affectation (=)!!**	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	$x==3	Retourne 1 si $x est égal à 3, sinon 0
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	$x<3	Retourne 1 si $x est inférieur à 3, sinon 0
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	$x<=3	Retourne 1 si $x est inférieur ou égale à 3, sinon 0
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur	$x>3	Retourne 1 si $x est supérieur à 3, sinon 0
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	$x>=3	Retourne 1 si $x est supérieur ou égal à 3, sinon 0
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	$x!=3	Retourne 1 si $x est différent de 3, sinon 0

Les opérateurs logiques (booléens) :

Ce type d'opérateur permet de vérifier si plusieurs conditions sont vraies:

Opérateur	Dénomination	Effet	Syntaxe
|| ou **OR**	OU logique	Vérifie qu'une des conditions est réalisée	((condition1)||(condition2))

&& ou **AND**	ET logique	Vérifie que toutes les conditions sont réalisées	((condition1)&&(condition2))
XOR	OU exclusif	Vérifie qu'une et une seule des conditions est réalisée	((condition1)XOR(condition2))
!	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	(!condition)

4- Structures de contrôle

La notion de bloc

Une expression suivie d'un point-virgule est appelée instruction. Par exemple *a++;* est une instruction. Lorsque l'on veut regrouper plusieurs instructions, on peut créer ce que l'on appelle un **bloc**, c'est-à-dire un ensemble d'instructions (suivies respectivement par des point-virgules) et comprises entre les accolades { et }.

Les instructions *if, while* et *for* peuvent par exemple être suivies d'un bloc d'instructions à exécuter...

L'instruction if

L'instruction if est la structure de test la plus basique, on la retrouve dans tous les langages (avec une syntaxe différente...). Elle permet d'exécuter une série d'instruction si jamais une condition est réalisée.

La syntaxe de cette expression est la suivante:

```
if (condition réalisée) {

   liste d'instructions

}
```

Remarques:

- la condition doit être entre des parenthèses
- il est possible de définir plusieurs conditions à remplir avec les opérateurs ET et OU (&& et ||) par exemple l'instruction suivante teste si les deux conditions sont vraies :
  if ((condition1)&&(condition2))
  L'instruction ci-dessous exécutera les instructions si l'une ou l'autre des deux conditions est vraie :
  if ((condition1)||(condition2))
- s'il n'y a qu'une instruction, les accolades ne sont pas nécessaires...

L'instruction if ... else

L'instruction *if* dans sa forme basique ne permet de tester qu'une condition, or la plupart du temps on aimerait pouvoir choisir les instructions à exécuter **en cas de non réalisation de la condition**...
L'expression *if ... else* permet d'exécuter une autre série d'instruction en cas de non-réalisation de la condition.

La syntaxe de cette expression est la suivante:

```
if (condition réalisée) {

   liste d'instructions
```

```
}
else {
     autre série d'instructions

}
```

L'instruction if ... elseif ... else

L'instruction *if ... else* ne permet de tester qu'une condition, or il est parfois nécessaire de tester plusieurs conditions de façon exclusive, c'est-à-dire que sur toutes les conditions une seule sera réalisée ...
L'expression *if ... elseif ... else* permet d'enchaîner une série d'instructions et évite d'avoir à imbriquer des instructions *if*.

La syntaxe de cette expression est la suivante:

```
if (condition réalisée) {

   liste d'instructions

}

elseif (autre condition réalisée) {

       autre série d'instructions

}

 ...

else (dernière condition réalisée) {

     série d'instructions

}
```

Une façon plus courte de faire un test (opérateur ternaire)

Il est possible de faire un test avec une structure beaucoup moins lourde grâce à la structure suivante, appelée opérateur ternaire:

```
(condition) ? instruction si vrai : instruction si faux
```

Remarques:

- la condition doit être entre des parenthèses
- Lorsque la condition est vraie, l'instruction de gauche est exécutée
- Lorsque la condition est fausse, l'instruction de droite est exécutée

L'instruction switch

L'instruction *switch* permet de faire plusieurs tests de valeurs sur le contenu d'une même variable. Ce branchement conditionnel simplifie beaucoup le test de plusieurs valeurs d'une variable, car cette opération aurait été compliquée (mais possible) avec des *if* imbriqués. Sa syntaxe est la suivante:

```
switch (Variable) {
case Valeur1:
```

```
        Liste d'instructions
        break;
case Valeur2:
        Liste d'instructions
        break;
case Valeurs...:

        Liste d'instructions
        break;
default:
        Liste d'instructions
        break;

}
```

Les parenthèses qui suivent le mot clé *switch* indiquent une expression dont la valeur est testée successivement par chacun des *case*. Lorsque l'expression testée est égale à une des valeurs suivant un *case*, la liste d'instructions qui suit celui-ci est exécutée. Le mot clé *break* indique la sortie de la structure conditionnelle. Le mot clé *default* précède la liste d'instructions qui sera exécutée si l'expression n'est jamais égale à une des valeurs.

N'oubliez pas d'insérer des instructions *break* entre chaque test, ce genre d'oubli est difficile à détecter car aucune erreur n'est signalée...

Les boucles

Les boucles sont des structures qui permettent d'exécuter plusieurs fois la même série d'instructions jusqu'à ce qu'une condition ne soit plus réalisée. On appelle parfois ces structures *instructions répétitives* ou bien *itérations*. La façon la plus commune de faire une boucle, est de créer un compteur (une variable qui s'incrémente, c'est-à-dire qui augmente de 1 à chaque tour de boucle) et de faire arrêter la boucle lorsque le compteur dépasse une certaine valeur.

La boucle for

L'instruction *for* permet d'exécuter plusieurs fois la même série d'instructions: c'est une boucle!

Dans sa syntaxe, il suffit de préciser le nom de la variable qui sert de compteur (et éventuellement sa valeur de départ, la condition sur la variable pour laquelle la boucle s'arrête (basiquement une condition qui teste si la valeur du compteur dépasse une limite) et enfin une instruction qui incrémente (ou décrémente) le compteur.

La syntaxe de cette expression est la suivante:

```
for (compteur; condition; modification du compteur) {

    liste d'instructions

}
```

Par exemple:

```
for ($i=1; $i<6; $i++) {

    echo "$i<br>";
```

```
}
```

Cette boucle affiche 5 fois la valeur de *$i*, c'est-à-dire 1,2,3,4,5 . Elle commence à $i=1, vérifie que $i est bien inférieur à 6, etc... jusqu'à atteindre la valeur $i=6, pour laquelle la condition ne sera plus réalisée, la boucle s'interrompra et le programme continuera son cours.

L'instruction while

L'instruction *while* représente un autre moyen d'exécuter plusieurs fois la même série d'instructions.

La syntaxe de cette expression est la suivante:

```
while (condition réalisée) {
      liste d'instructions
}
```

Cette instruction exécute la liste d'instructions **tant que** (*while* est un mot anglais qui signifie *tant que*) la condition est réalisée.

La condition de sortie pouvant être n'importe quelle structure conditionnelle, les risques de boucle infinie (boucle dont la condition est toujours vraie) sont grands, c'est-à-dire qu'elle risque de provoquer un plantage du navigateur!

Saut inconditionnel

Il peut être nécessaire de faire sauter à la boucle une ou plusieurs valeurs sans pour autant mettre fin à celle-ci.

La syntaxe de cette expression est "*continue;*" (cette instruction se place dans une boucle!), on l'associe généralement à une structure conditionnelle, sinon les lignes situées entre cette instruction et la fin de la boucle seraient obsolètes.

Exemple: Imaginons que l'on veuille imprimer pour $x allant de 1 à 10 la valeur de 1/($x-7) ... il est évident que pour $x=7 il y aura une erreur. Heureusement, grâce à l'instruction *continue* il est possible de traiter cette valeur à part puis de continuer la boucle!

```
$x=1;
while ($x<=10) {
      if ($x == 7) {
         echo "Division par zéro!";
         continue;
      }
$a = 1/($x-7);
echo "$a<br>";
$x++;
```

```
}
```

Il y avait une erreur dans ce programme... peut-être ne l'avez-vous pas vue: Lorsque $x est égal à 7, le compteur ne s'incrémente plus, il reste constamment à la valeur 7, il aurait fallu écrire:

```
$x=1;
while ($x<=10) {
      if ($x == 7) {
         echo "division par 0";
         $x++;
         continue;
      }
      $a = 1/($x-7);
      echo "$a<br>";
     $x++;
}
```

Arrêt inconditionnel

A l'inverse, il peut être voulu d'arrêter prématurément la boucle, pour une autre condition que celle précisé dans l'en-tête de la boucle. L'instruction *break* permet d'arrêter une boucle (*for* ou bien *while*). Il s'agit, tout comme *continue*, de l'associer à une structure conditionnelle, sans laquelle la boucle ne ferait jamais plus d'un tour!

Dans l'exemple de tout à l'heure, par exemple si l'on ne savait pas à quel moment le dénominateur ($x-7) s'annule (bon...OK...pour des équations plus compliquées par exemple) il serait possible de faire arrêter la boucle en cas d'annulation du dénominateur, pour éviter une division par zéro!

```
for ($x=1; $x<=10; $x++) {
    $a = $x-7;
    if ($a == 0) {
       echo "division par 0";
       break;
    }
    echo "1/$a<br>";
}
```