Web Dynamique
PHP + MySQL

Les défauts du langage HTML (1/2)

Absence des structures algorithmiques (conditionnelles ou itératives)

Un langage de création d'interface sans aucune logique de programmation procédurale(notion de sous programmes, type de données, variables...).

Aucune communication avec la plate forme d'exécution (date système, le navigateur utilisé..)

Absence de possibilité d'interfaçage avec une base de données.

Les défauts du langage HTML (2/2)

Absence de gestionnaires d'évènements autre que l'événement clique (pas de gestionnaires d'évènements par exemple pour le chargement d'une page ou le survol au dessus d'un objet)

Absence de mécanismes de verrouillage de code source (pour pouvoir le visualiser , il suffit d'utiliser l'option affichage code source de votre navigateur)

La petite histoire du PHP

Il a été créé en 1994 par Rasmus Lerdorf pour les besoins des pages web personnelles (livre d'or, compteurs, etc.). A l'époque, PHP signifiait *Personnal Home Page*.

C'est un langage incrusté au HTML et interprété (PHP3) ou compilé (PHP4) côté serveur. Il dérive du C et du Perl dont il reprend la syntaxe. Il est extensible grâce à de nombreux modules et son code source est ouvert. Comme il supporte tous les standards du web et qu'il est gratuit, il s'est rapidement répandu sur la toile.

En 1997, PHP devient un projet collectif et son interpréteur est réécrit par Zeev Suraski et Andi Gutmans pour donner la version 3 qui s'appelle désormais *PHP : Hypertext Preprocessor* (acronyme récursif à l'exemple du système Open Source *Linux : Is Not UniX*).

Il existe par ailleurs des applications web prêtes à l'emploi (PHPNuke, PHP SPIP, PHPSlash…) permettant de monter facilement et gratuitement son portail. En juillet 2000 plus de 300.000 sites tournaient déjà sous PHP !

PHP (Personnal Home Page)

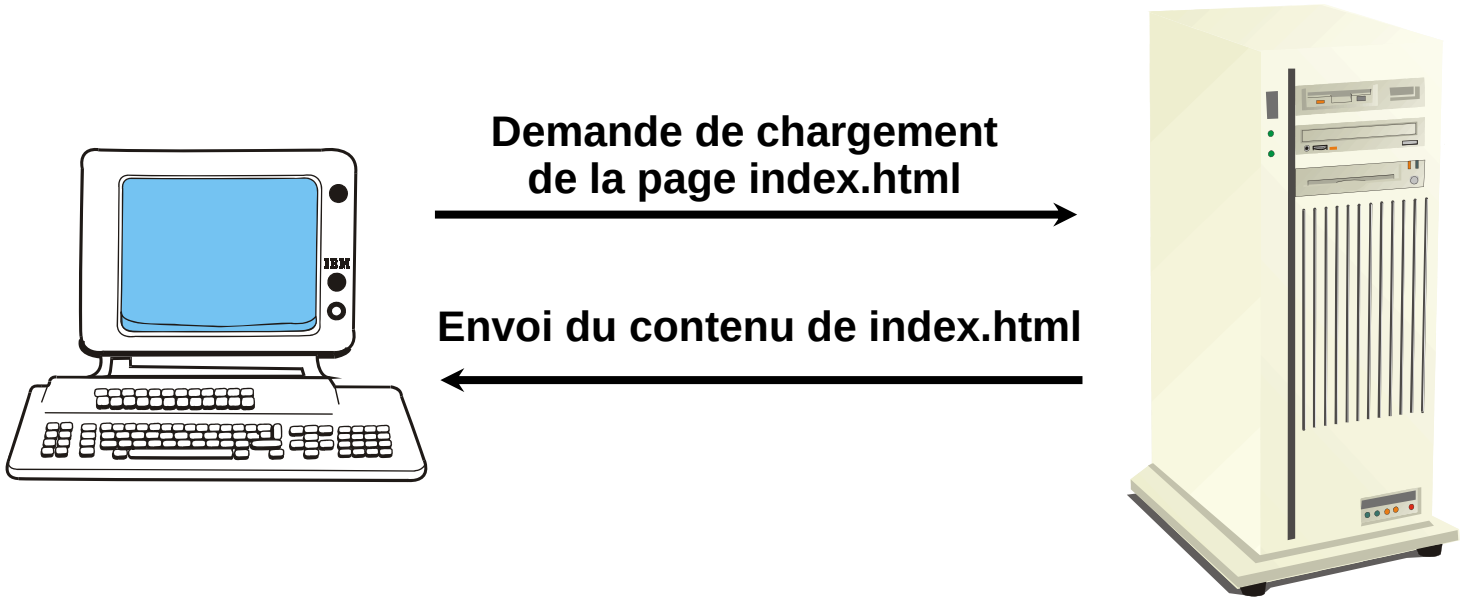
- Langage de script côté serveur
- Supporté par le Serveur Web Appache (le plus répandu)
- Conçu pour le web
- Proche du langage C
- portabilité du script : Multi plates-formes (Unix, Windows, …)
- connectivité avec un grand nombre de bases de données (en particulier MySql : Gratuit et Performant)
- Intégration directe dans le code HTML (sur la même page)
- Puissance
- Facilité de programmation
- Nombreuses sources d'informations et d'aide (tutoriel, froums, …)
-

PHP (Complément de HTML)

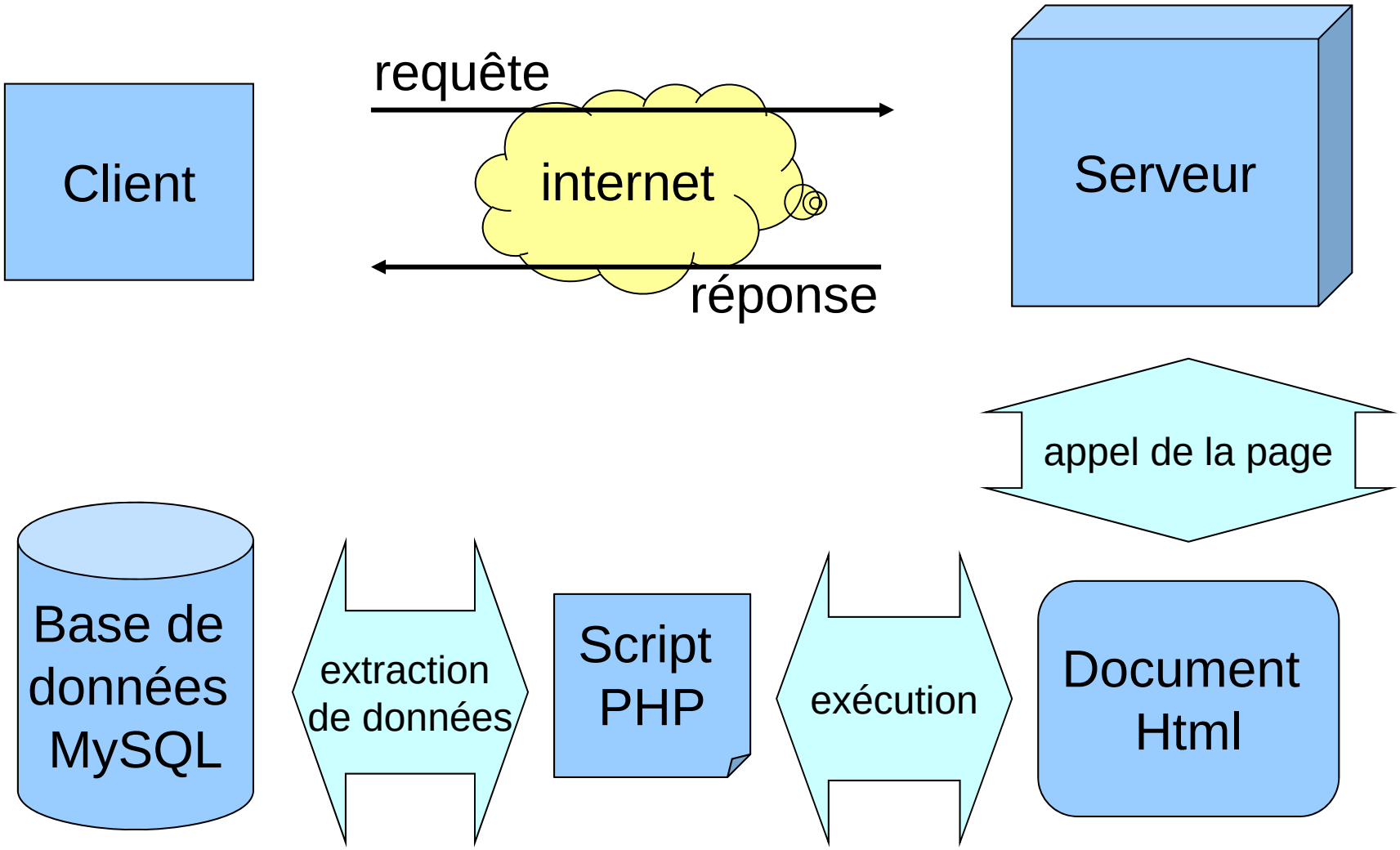
- PHP peut et doit être utilisé partout où il complète et simplifie la production de pages HTML.

  - En association avec des formulaires
  - En encapsulant dans des fonctions les parties bavardes de HTML
  - En gérant les transferts de fichier
  - En simplifiant la programmation de code HTML

Application Client-Serveur statique



Application Client-Serveur Dynamique



Partie 1 : PHP

Intégration d'un script dans une page

Les pages web sont au format html. Les pages web dynamiques générées avec PHP4 sont au format php. Le code source php est directement insérer dans le fichier html grâce au conteneur de la norme XML :

<?php … ?>

Exemple:

```
<html>
<body>
<?php
        echo ”Bonjour”;
?>
</body>
</html>
```

Autres syntaxes d'intégration :

- **<? … ?>**
- **<script language=‘’php”> … </script>**
- **<% … %>**

Exemple de script

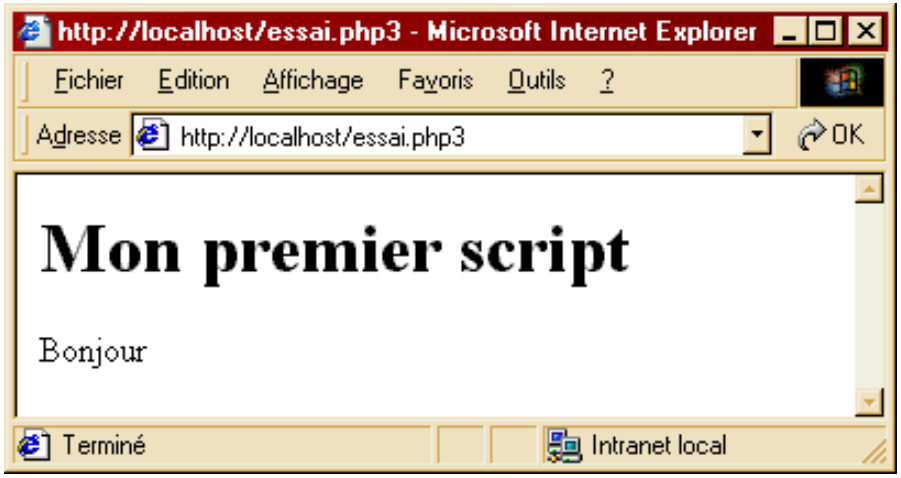
Exemple de script :

```
<html>
<body>
<h1>Mon premier script</h1>
<?php echo ‘’Bonjour\n”; ?>
</body>
</html>
```

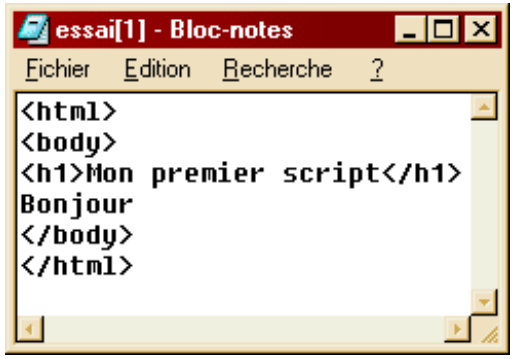
Autre écriture du même script :

```
<?php
echo ‘’<html>\n<body>\n”;
echo ‘’<h1>Mon premier script</h1>\n”;
echo ‘’Bonjour\n”;
echo ‘’</body>\n</html>\n”;
?>
```

Résultat affiché par le navigateur :



Code source de la page essai.ph3 résultant du script



Commentaires

Un script php se commente comme en C :

Exemple :

```
<?php

// commentaire de fin de ligne

/* commentaire
sur plusieurs
lignes */

# commentaire de fin de ligne comme en Shell
?>
```

Tout ce qui se trouve dans un commentaire est ignoré. Il est conseillé de commenter largement ses scripts.

Affichage

Les fonctions d'affichage :

echo() : écriture dans le navigateur

print() : écriture dans le navigateur

printf([$format, $arg1, $arg2]) : écriture formatée comme en C, i.e. la chaîne de caractère est constante et contient le format d'affichage des variables passées en argument

Exemples :

```
echo ''Bonjour $name'';
print(''Bonjour $name'');
printf(''Bonjour %s'', $name);
```

Variables, types et opérateurs (I)

Le typage des variables est implicite en php. Il n'est donc pas nécessaire de déclarer leur type au préalable ni même de les initialiser avant leur utilisation.

Les identificateurs de variable sont précédés du symbole « **$** » (dollars) et ne doivent pas contenir d'espace ou de caractères spéciaux.

Exemple : **$toto**.

Les variables peuvent être de type entier (**integer**), réel (**double**), chaîne de caractères (**string**), tableau (**array**), objet (**object**), booléen (**boolean**).

Il est possible de convertir une variable en un type primitif grâce au cast(1) (comme en C).

Exemple :

```
$str = ”12”;          // $str vaut la chaîne ‘’12”
$nbr = (int)$str;     // $nbr vaut le nombre 12
```

Variables, types et opérateurs (II)

Quelques fonctions :

empty($var) : renvoie vrai si la variable est vide

isset($var) : renvoie vrai si la variable existe

unset($var) : détruit une variable

gettype($var) : retourne le type de la variable

settype($var, ”type”) : convertit la variable en type **type** (cast)

is_long(), **is_double()**, **is_string()**, **is_array()**, **is_object()**, **is_bool()**,

…

Une variable peut avoir pour identificateur la valeur d’une autre variable.

Syntaxe : **${$var} = valeur;**

Exemple :

```
$toto = ‘’foobar’’;
${$toto} = 2002;
echo $foobar;                    // la variable $foobar vaut 2002
```

Variables, types et opérateurs (II)

Pour vérifier si une variable est vide, on utilise la fonction empty() qui renvoie true si elle contient une information et false sinon.

```
$variable = "a";

if(empty($variable))
echo "vide";
else echo "non vide";
```

La fonction gettype() permet de recupérer le type de données d'une variable. On peut assigner les types suivants :

```
if(gettype($variable) == "integer")
```

La fonction settype() permet de définir explicitement le type d'une variable.

```
$variable = 2.5;
settype($variable, "integer");
```

La variable $variable renverra maintenant 2 et non 2.5.

Variables, types et opérateurs (III)

La portée d'une variable est limitée au bloc dans lequel elle a été crée. Une variable locale à une fonction n'est pas connue dans le reste du programme. Tout comme une variable du programme n'est pas connue dans une fonction. Une variable créée dans un bloc n'est pas connue dans les autres blocs, mêmes supérieurs.

Si la variable est déclarée tout au début du script, en dehors et avant toute fonction, elle est toujours globale.

```
<? $ecole=« Atelier PHP" // est connue de toutes les fonctions function
    signer()
    {
        global $ecole; // $ecole est la variable globale précédente
        $titre= "Participant ";
        $titre = $titre . $ecole; // Participant Atelier PHP
    }

?>
```

Variables, types et opérateurs (III)

Opérateurs arithmétiques :

+ (addition), **-** (soustraction), ***** (multiplié), ***/*** (divisé), **%** (modulo), **++** (incrément), **--**(décrément). Ces deux derniers peuvent être pré ou post fixés

Opérateurs d'assignement :

= (affectation), ***=** (**$x*=$y** équivalent à **$x=$x*$y**), ***/=***, **+=**, **-=**, **%=**

Opérateurs logiques :

and, **&&** (et), **or**, **||** (ou), **xor** (ou exclusif), **!** (non)

Opérateurs de comparaison :

== (égalité), **<** (inférieur strict), **<=** (inférieur large), **>**, **>=**, **!=** (différence)

Variables, types et opérateurs (IV)

Signalons un opérateur très spécial qui équivaut à une structure conditionnelle complexe *if then else* à la différence qu'il renvoie un résultat de valeur pouvant ne pas être un booléen : l'opérateur ternaire.

Syntaxe :

(condition)?(expression1):(expression2);

Si la **condition** est vrai alors évalue et renvoie l'**expression1** sinon évalue et renvoie l'**expression2**.

Exemple :

```
$nbr = ($V1>10)?($V1):($V1%10);
```

Dans cet exemple, la variable **$nbr** prend **$V1** pour valeur si **$V1** est strictement supérieur à **10**, sinon vaut le reste de la division entière de **$V1** par **10**.

Constantes

L'utilisateur peut définir des constantes dont la valeur est fixée une fois pour toute. Les constantes ne portent pas le symbole **$** (dollars) en début d'identificateur et ne sont pas modifiables.

define(''var'',valeur) : définit la constante **var** (sans **$**) de valeur **valeur**

Exemple 1 :

```
define("auteur","Foobar");
echo auteur;                    // affiche 'Foobar'
```

Exemple 2 :

```
define(Mon_Annee,1980);
echo Mon_Annee;                 // affiche 1980
```

Contrairement aux variables, les identificateurs de constantes (et aussi ceux de fonction) ne sont pas sensibles à la casse.

Références

On peut à la manière des pointeurs en C faire référence à une variable grâce à l'opérateur **&** (ET commercial).

Exemple 1 :

```
$V1 = 100;          // la variable $V1 est initialisée à la valeur 100
$foobar = &$V1; // la variable $foobar fait référence à $V1
$V1++;              // on change la valeur de $V1
echo $foobar;       // qui est répercutée sur $foobar qui vaut alors 101
```

Exemple 2 :

```
function change($var) {
        $var++;     // la fonction incrémente en local l'argument
}
$nbr = 1;  // la variable $nbr est initialisée à 1
change(&$nbr);      // passage de la variable par référence
echo $nbr;          // sa valeur a donc été modifiée
```

Mathématiques

Il existe une multitude de fonctions mathématiques.

abs($x) : valeur absolue
ceil($x) : arrondi supérieur
floor($x) : arrondi inférieur
pow($x,$y) : x exposant y
round($x,$i) : arrondi de x à la ième décimale
max($a, $b, $c …) : retourne l'argument de valeur maximum
pi() : retourne la valeur de Pi

Et aussi : **cos**, **sin**, **tan**, **exp**, **log**, **min**, **pi**, **sqrt**…

Nombres aléatoires :
rand($x,$y) : valeur aléatoire entre x et y
srand() : initialisation du générateur aléatoire

Dates et heures (I)

Les fonctions de dates et heures sont incontournables sur Internet et sont indispensables pour la conversion en français des dates fournies par la base de données MySQL qui les code au format anglophone (YYYY-DD-MM hh:mm:ss).

Quelques fonctions :

date(''$format'') : retourne une chaîne de caractères contenant la date et/ou l'heure locale au format spécifié

getdate() : retourne un tableau associatif contenant la date et l'heure

checkdate($month, $day, $year) : vérifie la validité d'une date

mktime ($hour, $minute, $second, $month,$day, $year) : retourne le timestamp UNIX correspondant aux arguments fournis c'est-à-dire le nombre de secondes entre le début de l'époque UNIX (1er Janvier 1970) et le temps spécifié

time() : retourne le timestamp UNIX de l'heure locale

Dates et heures (II)

Exemple 1 :

```
echo date(''Y-m-d H:i:s'');
/* affiche la date au format MySQL : '2002-03-31 22:30:29' */
```

Exemple 2 :

```
$aujourdhui = getdate();
$mois = $aujourdhui['mon'];
$jour = $aujourdhui['mday'];
$annee = $aujourdhui['year'];
echo ''$jour/$mois/$annee'';          // affiche '31/3/2002'
```

Dates et heures (III)

Les clés du tableau retourné par **getdate** :

seconds : secondes

minutes : minutes

hours : heures

mday : jour du mois

wday : jour de la semaine, numérique

mon : mois de l'année, numérique

year : année, numérique

yday : jour de l'année, numérique

weekday : jour de la semaine, textuel complet en anglais

month : mois, textuel complet en anglais

Dates et heures (IV)

Les formats pour date :

d Jour du mois sur deux chiffres [01..31] **j** Jour du mois sans les zéros initiaux

l Jour de la semaine textuel en version longue et en anglais

D Jour de la semaine textuel en trois lettres et en anglais

w Jour de la semaine numérique [0..6] (0: dimanche)

z Jour de l'année [0..365]

m Mois de l'année sur deux chiffres [01..12] **n** Mois sans les zéros initiaux

F Mois textuel en version longue et en anglais

M Mois textuel en trois lettres

Y Année sur 4 chiffres **y** Année sur 2 chiffres

h Heure au format 12h [01..12] **g** Heure au format 12h sans les zéros initiaux

H Heure au format 24h [00..23] **G** Heure au format 24h sans les zéros initiaux

i Minutes [00..59] **s** Secondes [00.59]

a am ou pm **A** AM ou PM

L Booléen pour savoir si l'année est bisextile (1) ou pas (0)

S Suffixe ordinal anglais d'un nombre (ex: *nd* pour 2)

t Nombre de jour dans le mois donné [28..31]

U Secondes depuis une époque **Z** Décalage horaire en secondes [-43200..43200]

Structures de contrôle (I)

Structures conditionnelles (même syntaxe qu'en langage C) :

```
if( ... ) {
        ...
} elseif(... ) {
        ...
} else {
        ...
}

switch( ... ) {
        case ... : { ... } break
        ...
        default : { ... }
}
```

Structures de contrôle (II)

Exemple 1:

```
$aujourdhui = getdate();
$mois = $aujourdhui['mon'];

If ($mois == 12 )
{
        echo "Decembre";
}
Else
{
        echo " Ce n\'est pas le dernier mois ";
}
```

Structures de contrôle (III)

Exemple 2:

Utilisation de switch

Structures de contrôle (IV)

Structures de boucle (même syntaxe qu'en langage C) :

```
for( ... ; ... ; ... ) {
        ...
}

while( ... ) {
        ...
}

do {
        ...
} while( ... );
```

Structures de contrôle (V)

Exemple 1:

```
$i=1;
$somme=0;          //facultative
While ($i<=10) {
        $somme = $somme +$i;
        $i++;
        Echo $somme;
}

for($i=1; $i<=10; $i++) {
        $somme = $somme +$i;
        Echo $somme;
}
```

Structures de contrôle (VI)

L'instruction **break** permet de quitter prématurément une boucle.

Exemple :

```
while($nbr = $tab[$i++]) {
        echo $nbr."<br />";
        if($nbr == $stop)
                break;
}
```

L'instruction **continue** permet d'éluder les instructions suivantes de l'itération courante de la boucle pour passer à la suivante.

Exemple :

```
for($i=1; $i<=10; $i++) {
        if($tab[$i] == $val)
                continue;
        echo $tab[$i];
}
```

Arrêt prématuré

Pour stopper prématurément un script, il existe deux fonctions.

die arrête un script et affiche un message d'erreur dans le navigateur.

Exemple :

```
if(mysql_query($requette) == false)
        die("Erreur de base de données à la requête : <br />$requet");
```

exit l'arrête aussi mais sans afficher de message d'erreur.

Exemple :

```
function foobar() {
        exit();
}
```

Ces fonctions stoppent tout le script, pas seulement le bloc en cours.

Chaînes de caractères (I)

Une variable chaîne de caractères n'est pas limitée en nombre de caractères.

Elle est toujours délimitée par des simples quotes ou des doubles quotes.

Exemples :

```
$nom = "OUTANOUTE";
$prenom = 'M'hamed';
```

Les doubles quotes permettent l'évaluation des variables et caractères spéciaux contenus dans la chaîne (comme en C ou en Shell) alors que les simples ne le permettent pas.

Exemples :

```
echo "Nom: $nom";          // affiche Nom: OUTANOUTE
echo 'Nom: $nom';          // affiche Nom: $nom
```

Quelques caractères spéciaux : **\n** (nouvelle ligne), **\r** (retour à la ligne), **\t** (tabulation horizontale), **** (antislash), **\$** (caractère dollars), **\"** (double quote).

Exemple : `echo "Hello Word !\n";`

Chaînes de caractères (II)

Opérateur de concaténation de chaînes : **.** (point)

Exemple 1 :

```
$foo = "Salut";
$bar = "Tout le monde";
echo $foo.$bar;
```

Exemple 2 :

```
$name = "PHP";
$whoiam = $name." et MYSQL";
```

Exemple 3 :

```
$out = 'Debut ';
$out .= " a la fin...";
```

Chaînes de caractères (III)

Affichage d'une chaîne avec **echo**:

Exemples:

```
echo 'Bonjours.';
echo " Salut ${name}\n";
echo "Nom : ", $name;
echo("Bonjour");
```

Quelques fonctions:

strlen($str) : retourne le nombre de caractères d'une chaîne

strtolower($str) : conversion en minuscules

strtoupper($str) : conversion en majuscules

trim($str) : suppression des espaces de début et de fin de chaîne

substr($str,$i,$j) : retourne une sous chaîne (entre les positions i et j)

strnatcmp($str1,$str2) : comparaison de 2 chaînes

addslashes($str) : déspécialise les caractères spéciaux (‘, ‘’, \)

ord($char) : retourne la valeur ASCII du caractère

Chaînes de caractères (IV)

On peut délimiter les chaînes de caractères avec la syntaxe *Here-doc.*

Exemple :

```
$essai = <<<EOD
Ma chaîne ‘’essai”
sur plusieurs lignes.
EOD;
echo $essai;
```

La valeur de la variable **$essai** est délimitée par un identifiant que vous nommez librement. La première apparition de cet identifiant doit être précédée de 3 signes inférieurs **<**. Sa deuxième apparition doit se faire en premier sur une nouvelle ligne. Cette valeur chaîne se comporte comme si elle était délimitée par des doubles quotes ‘’ ‘’ dans le sens où les variables seront évaluées. Par contre il est inutile d’échapper les guillemets, c’est-à-dire que la déspécialisation des doubles quotes devient inutile.

Tableaux (I)

Une variable tableau est de type **array**. Un tableau accepte des éléments de tout type. Les éléments d'un tableau peuvent être de types différents et sont séparés d'une virgule.

Un tableau peut être initialisé avec la syntaxe **array**.

Exemple :

```
$tab_colors = array('rouge, jaune, 'bleu', 'blanc);
$tab = array('foobar', 2002, 20.5, $name);
```

Mais il peut aussi être initialisé au fur et à mesure.

Exemples :

```
$prenoms[ ] = "Mohamed";          $villes[0] = "Rabat";
$prenoms[ ] = "Hamid";            $villes[1] = "Alger";
$prenoms[ ] = "Fatima";           $villes[2] = "Tunis";
```

L'appel d'un élément du tableau se fait à partir de son indice (dont l'origine est zéro comme en C**)**.

Exemple : `echo $tab[10];` // pour accéder au 11ème élément

Tableaux (II)

Parcours d'un tableau.

```
$tab = array("Hassan", "Jamal", "Mounir");
```

Exemple 1 :

```
$i=0;
while($i < count($tab)) {          // count() retourne le nombre d'éléments
  echo $tab[$i].'\n';
  $i++;
}
```

Exemple 2 :

```
foreach($tab as $elem) {
  echo $elem."<BR>";
}
```

La variable **$elem** prend pour valeurs successives tous les éléments du tableau **$tab**.

Tableaux (III)

Quelques fonctions:

count($tab), **sizeof** : retournent le nombre d'éléments du tableau

in_array($var,$tab) : dit si la valeur de **$var** existe dans le tableau **$tab**

list($var1,$var2…) : transforme une liste de variables en tableau

range($i,$j) : retourne un tableau contenant un intervalle de valeurs

shuffle($tab) : mélange les éléments d'un tableau

sort($tab) : trie alphanumérique les éléments du tableau

rsort($tab) : trie alphanumérique inverse les éléments du tableau

implode($str,$tab), join : retournent une chaîne de caractères contenant les éléments du tableau **$tab** joints par la chaîne de jointure **$str**

explode($delim,$str) : retourne un tableau dont les éléments résultent du hachage de la chaîne **$str** par le délimiteur **$delim**

array_merge($tab1,$tab2,$tab3…) : concatène les tableaux passés en arguments

array_rand($tab) : retourne un élément du tableau au hasard

Tableaux (IV)

Il est possible d'effectuer des opérations complexes sur les tableaux en créant par exemple sa propre fonction de comparaison des éléments et en la passant en paramètre à une fonction de tri de php.

usort($tab, ''fonction'');

Trie les éléments grâce à la fonction **fonction** définie par l'utilisateur qui doit prendre 2 arguments et retourner un entier, qui sera inférieur, égal ou supérieur à zéro suivant que le premier argument est considéré comme plus petit, égal ou plus grand que le second argument. Si les deux arguments sont égaux, leur ordre est indéfini.

Attention, les variables tableaux ne sont pas évaluées lorsqu'elles sont au milieu d'une chaîne de caractères délimitée par des doubles quottes.

Exemple :

```
echo "$tab[3]";          // syntaxe invalide
echo $tab[3];            // syntaxe valide
```

Tableaux associatifs (I)

Un tableau associatif est appelé aussi *dictionnaire* ou *hashtable*. On associe à chacun de ses éléments une clé dont la valeur est de type chaîne de caractères.

L'initialisation d'un tableau associatif est similaire à celle d'un tableau normal.

Exemple 1 :

```
$personne = array("Nom" => "César", "Prénom" => "Jules");
```

Exemple 2 :

```
$personne["Nom"] = "César";
$personne["Prénom"] = "Jules";
```

Ici à la clé **''Nom''** est associée la valeur **''César''**.

Tableaux associatifs (II)

Parcours d'un tableau associatif.

```
$personne = array("Nom" => "César", "Prénom" => "Jules");
```

Exemple 1 :

```
foreach($personne as $elem) {
  echo $elem;
}
```

Ici on accède directement aux éléments du tableau sans passer par les clés.

Exemple 2 :

```
foreach($personne as $key => $elem) {
  echo "$key : $elem";
}
```

Ici on accède simultanément aux clés et aux éléments.

Tableaux associatifs (III)

Quelques fonctions :

array_count_values($tab) : retourne un tableau contenant les valeurs du tableau **$tab** comme clés et leurs fréquence comme valeur (utile pour évaluer les redondances)

array_keys($tab) : retourne un tableau contenant les clés du tableau associatif **$tab**

array_values($tab) : retourne un tableau contenant les valeurs du tableau associatif **$tab**

array_search($val,$tab) : retourne la clé associée à la valeur **$val**

L'élément d'un tableau peut être un autre tableau.

Les tableaux associatifs permettent de préserver une structure de données.

Tableaux associatifs (IV)

Quelques fonctions alternatives pour le parcours de tableaux (normaux ou associatifs) :

reset($tab) : place le pointeur sur le premier élément

current($tab) : retourne la valeur de l'élément courant

next($tab) : place le pointeur sur l'élément suivant

prev($tab) : place le pointeur sur l'élément précédant

each($tab) : retourne la paire clé/valeur courante et avance le pointeur

Exemple :

```
$colors = array("red", "green", "blue");
$nbr = count($colors);
reset($colors);
for($i=1; $i<=$nbr; $i++) {
        echo current($colors)."<br />";
        next($colors);
}
```

Tableaux à deux dimensions (I)

Les éléments d'un tableau peuvent être à leur tour des tableaux.

Ces tableaux dits multidimensionnels peuvent être comparés à des matrices.

Exemple :

```
$produit = array ( array ( "Ord", "Ordinateur",6700),

        array ( "Imp", "Imprimante", 1600),
        array ( ""Vid", "Video", 2200) );
```

Pour afficher le contenu de ce tableau :

```
For ($i=0;$i<3;$i++)
{
    for($j=0;$j>3;$j++)
    {
        echo "|".$produit[$i][$j]."|";
    }
    echo "<br>";
}
```

Fonctions (I)

Comme tout langage de programmation, php permet l'écriture de fonctions.

Les fonctions regroupent un ensemble d'instructions réutilisables, elles peuvent (mais pas obligatoire) accepter des valeurs appelées arguments ou paramètres.

S'il y'en a plusieurs, les paramètres sont séparés par des virgules.

Une fonction réalise une succession d'instructions et peut renvoyer une et une seule valeur à préciser à l'aide du Return.

Exemple :

```
function mafonction($V) {
    $V += 15;
    echo "Salut !";
    return ($V+10);
}
$nbr = MaFonction(15.1);
/* retourne 15.1+15+10=40.1, les majuscules n'ont pas d'importance */
```

Fonctions (II)

Les identificateurs de fonctions sont insensibles à la casse.

On peut modifier la portée des variables locales à une fonction.

global permet de travailler sur une variable de portée globale au programme.
Le tableau associatif **$GLOBALS** permet d'accéder aux variables globales du script (**$GLOBALS[''var'']** accède à la variable **$var**).

Exemple :

```
function change() {
        global $var;              // définit $var comme globale
        $GLOBALS["V"] ++; // incrémente la variable globale $V
        $var++;                   // incrémente la variable globale $var
}                                 // cela sera répercuté dans le reste du programme
```

static permet de conserver la valeur d'une variable locale à une fonction.

Exemple :

```
function change() {
        static $var;      // définit $var comme statique
        $var++;           // sa valeur sera conservée jusqu'au prochain appel
}
```

Fonctions (III)

On peut donner une valeur par défaut aux arguments lors de la déclaration de la fonction.
Ainsi, si un argument est « oublié » lors de l'appel de la fonction, cette dernière lui donnera automatiquement une valeur par défaut décidée à l'avance par le programmeur.

Exemple :

```
function Set_Color($color="black") {
        global $car;
        $car["color"] = $color;
}
```

Forcer le passage de paramètre par référence (équivalent à user de **global**):

Exemple :

```
function change(&$var) { // force le passage systématique par référence
        $var += 100;       // incrémentation de +100
}
$V = 12;       // $V vaut 12
change($V); // passage par valeur mais la fonction la prend en  référence
echo $V;      // $V vaut 112
```

Fonctions (IV)

Une fonction peut être définie après son appel (en PHP4 du fait de la compilation avant exécution contrairement au PHP3 qui est interprété).

Exemple :

```
function une() {        // définition de la fonction une
        echo "Binevenu…";
}
une();                  // appel de la fonction une définie plus haut
deux();                 // appel de la fonction deux pas encore définie
function deux() {       // définition de la fonction deux
        echo "PHP!<br>";
}
```

Cet exemple affichera : **Bienvenu…PHP!**.

Fonctions (V)

Grâce à une petite astuce, il est possible de faire retourner plusieurs valeurs d'une fonction.
En retournant un tableau ayant pour éléments les variables à retourner.
Dans l'appel de la fonction, il faudra alors utiliser la procédure **list()** qui prend en paramètre les variables à qui ont doit affecter les valeurs retournées.
On affecte à **list()** le retour de la fonction.

Exemple :

```
function trigo($nbr) {
     return array(sin($nbr), cos($nbr), tan($nbr));   // retour d'un tableau
}
$r = 12;
list($a, $b, $c) = trigo($r);          /* affectation aux variables $a,$b et $c
                        des éléments du tableau retourné par la fonction trigo */
echo "sin($r)=$a, cos($r)=$b, tan($r)=$c";      // affichage des variables
```

Cet exemple affichera ceci :

sin(12)=-0,5365729180, cos(12)=0,8438539587, tan(12)=-0,6358599286

Classes (I)

PHP gère la programmation orientée objet à l'aide de classes.

Exemple :

```
class Voiture {                          // déclaration de la classe
        var $couleur;                    // déclaration d'un attribut
        var $belle = TRUE;               // initialisation d'un attribut
        function voiture() {             // constructeur
                $this->couleur = "noire";
        }   // le mot clé $this faisant référence à l'objet est obligatoire
        function Set_Couleur($couleur) {
                $this->couleur = $couleur;
        }
}
$mavoiture = new Voiture();              // création d'une instance
$mavoiture->Set_Couleur("blanche");      // appel d'une méthode
$coul = $mavoiture->couleur;             // appel d'un attribut
```

Classes (II)

Le système de classes de php est très succint, il ne gère que l'héritage simple.

Exemple :

```
class Voituredeluxe extends Voiture {   // déclaration de la sous classe
        var $couleur;
        function voituredeluxe() {        // constructeur
                $this->Voiture();
        }
        function Set_Couleur($couleur) {
                $this->couleur = $couleur;
        }
        function Get_Couleur() {
                return $this->couleur;
        }
}
```

La nouvelle classe **Voituredeluxe** hérite de tous les attributs et méthodes de la classe parente **Voiture** dont elle est une extension (**extends**).
Il est possible de surcharger les méthodes, d'en déclarer de nouvelles, etc.

Classes (III)

Quelques fonctions :

get_declared_classes() : retourne un tableau listant toutes les classes définies

class_exists($str) : vérifie qu'une classe dont le nom est passé en argument a été définie

get_class($obj), **get_parent_class** : retournent le nom de la classe de l'objet **$obj**

get_class_methods($str) : retourne les noms des méthodes de la classe **$str** dans un tableau

get_class_vars($str) : retourne les valeurs par défaut des attributs de la classe **$str** dans un tableau associatif

get_object_vars($obj) : retourne un tableau associatif des attributs de l'objet **$obj** les clés sont les noms des attributs et les valeurs, celles des attributs si elles existent

is_subclass_of($obj,$str) : détermine si l'objet **$obj** est une instanciation d'une sous-classe de **$str**, retourne VRAI ou FAUX

method_exists($obj,$str) : vérifie que la méthode **$str** existe pour une classe dont **$obj** est une instance, retourne VRAI ou FAUX