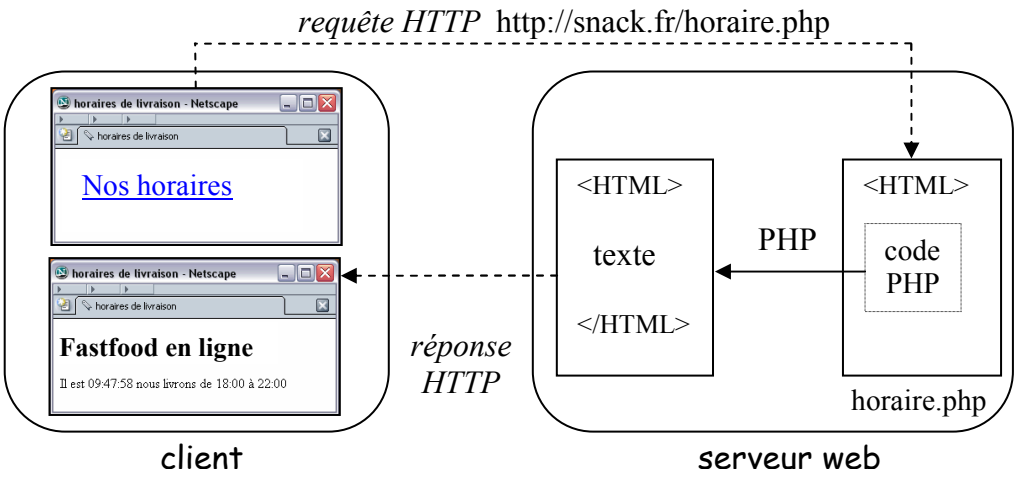
Chapitre 1 – Bases du langage PHP

1. Caractéristiques

- langage de scripts
- scripts exécutés par le serveur Web
- créé pour la génération de pages Web
- API pour un grand nombre de bases de données.
- Similitudes avec les autres langages
  - syntaxe proche du langage C,
  - gestion de la mémoire, manipulation des chaînes de caractères et syntaxe des variables scalaires proche de Perl
  - proche de Java pour les objets.
- gratuit et Open Source
- intégré dans le serveur Web Apache (exécution possible en CGI)
- porter un script PHP sur une plate-forme ne demande aucun effort supplémentaire de développement.
- Convention de codage PEAR (*PHP Extension and Application Repository)*

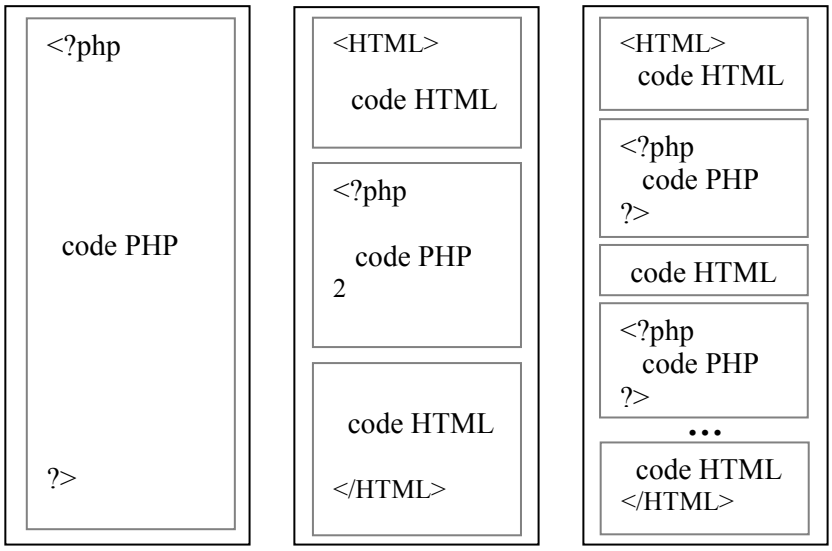
2. Principe



3. Historique et versions

- début des années 90 Rasmus Lerdorf développe un ensemble d'outils (scripts CGI écrits en Perl réécrits en C par la suite) afin de conserver une trace des accès à son *curriculum vitae* sur son site Web
- 1995 : 1ère version publique *Personal Home Page Tools* 1.0 (ensemble d'outils + Forms Interpreter (FI)
- 1997 PHP/FI version 2.0 (boucles, conditions, tableaux, …)
- 1998 PHP 3.0
  - nouvel analyseur grammatical (Andi Gutmans et Zeev Suraski)
  - API pour étendre le langage facilement en ajoutant des modules
  - scripts exécutés au cours de l'analyse
- PHP devient l'acronyme de *Professional Home Page*.
- 2000 PHP 4.0
  - sessions
  - moteur interne plus performant (Zend Engine) : l'analyse et l'exécution deviennent deux étapes séparées.
  - compatibilité ascendante assez bonne
  - acronyme récursif « PHP : Hypertext Preprocessor »
- PHP 5 : nouvelle version de Zend Engine, objet

4. Structure d'un script PHP



Trois autres balises spéciales existent (à éviter)

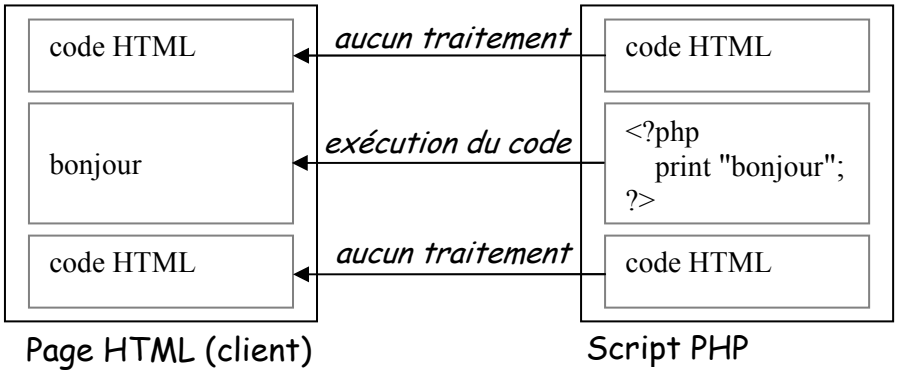
```
<? code ?>
<% code %>
<script language="php"> code </script>
```

Un script peut comporter

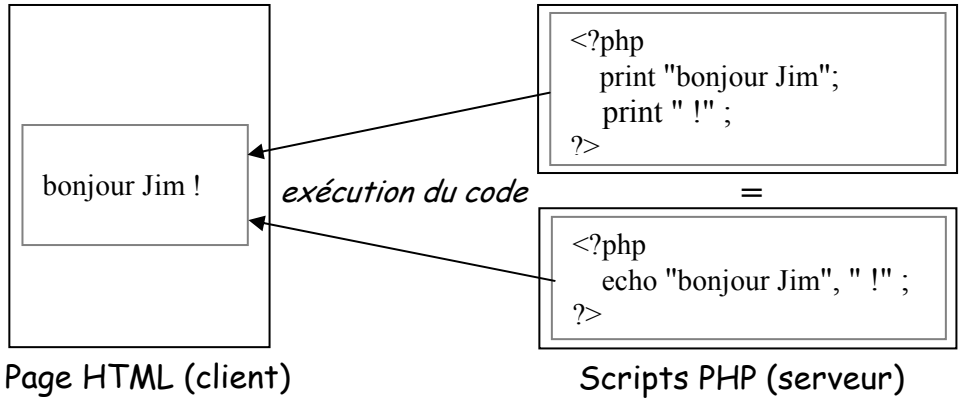
- une ou plusieurs instructions terminées par un ';' (optionnel pour la dernière instruction)
- des blocs : portions de code PHP entre accolades
- des commentaires : // ou /* */ (le symbole # peut être rencontré comme marque de commentaire sur une ligne il faut l'éviter).

5. Envoi de données vers la sortie standard

```
print arg;
```




```
echo arg_1, arg_2, …, arg_N ;
```



Un premier script PHP

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
  <TITLE>horaires de livraison</TITLE>
  <META http-equiv="Content-Type" content="text/html;
        charset=ISO-8859-1">
</HEAD>
<BODY>
<H1>Fastfood en ligne</H1>
<DIV>
<?php
   print "Il est "; // ecrit la chaine entre guillemets
   echo date("H:i:s"); /* ecrit l'heure courante
                       donnee par la fonction predefinie
                       date dans le format specifie */
   echo " nous livrons de <EM>18:00 &agrave; 22:00</EM>";
?>  </DIV></BODY></HTML>
```

Code HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
  <TITLE>horaires de livraison</TITLE>
  <META http-equiv="Content-Type" content="text/html;
        charset=ISO-8859-1">
</HEAD>
<BODY>
<H1>Fastfood en ligne</H1>
<DIV>
Il est 17:43:21 nous livrons de <EM>18:00 &agrave; 22:00</EM>
</DIV></BODY></HTML>
```

6. Variables

- toute variable commence par le caractère '$' suivi d'un identifiant.
- L'identifiant comporte un premier caractère qui est obligatoirement une lettre ou un caractère '_', suivi éventuellement d'autres caractères qui peuvent être des lettres, des chiffres ou des '_'.
- sensible à la casse (sauf pour les noms de fonctions)
- typage réalisé automatiquement parmi un des huit types PHP : *integer*, *float*, *boolean*, *string* (chaînes de caractères), *array* (tableaux), *object*, `resource` (depuis PHP 4 - représente une ressource externe) et *null* (depuis PHP 4 - représente l'absence de valeur)
- Typage momentané : une variable qui est de type integer peut avoir le type string plus loin dans le code
- Affectation par valeur : `$nom_variable = expr ;`
- Affectation d'une référence (depuis PHP 4) : `$var1 = &$var2;`

```
$x = 12;
$y = 18;
$x = &$y;
echo "<BR>x = ", $x, " et y = ", $y;
$y = 5;
echo "<BR>x = ", $x, " et y = ", $y;
$x = 7;
echo "<BR>x = ", $x, " et y = ", $y;
```

```
x = 18 et y = 18
x = 5 et y = 5
x = 7 et y = 7
```

7. Constantes

```
define("HEURE_FIN", "21h30");
echo "commandez avant ", HEURE_FIN;
```

`boolean defined("CONST")` retourne `TRUE` si la constante est déjà définie `FALSE` sinon.

Constantes prédéfinies

`TRUE` et `FALSE`

`__FILE__` donne le nom du fichier du script exécuté,

`__LINE__` fournit le numéro de la ligne du script sur laquelle elle apparaît,

`__FUNCTION__` donne le nom de la fonction dans laquelle la constante apparaît.

Certaines dépendent de la configuration.

`PHP_VERSION` version de PHP
`PHP_OS` système d'exploitation sur lequel le script PHP est exécuté.

Liste : `<pre> print_r(get_defined_constants()); ?> </pre>`

8. Opérateurs

- ✓ arithmétiques : `+ - * / % ++ --`
- ✓ relationnels : `== > < >= <= != === !==`
- ✓ de concaténation : `.`
- ✓ logiques : `! && || xor and or`
- ✓ bits à bits : `& | ^`
- ✓ de décalage : `>> << ~`
- ✓ d'affectation et d'affectation élargie : `= += -= *= /= %= .= &= |= ^= <<= >>=`
- ✓ de contrôle d'erreur : @ (bloque l'envoi d'un message d'erreur)
- ✓ d'exécution : ` commande ` (exécute la commande shell)
- ✓ conditionnel ternaire : `'? :'`
- ✓ d'ajout pour les tableaux : +
- ✓ de *cast* : ( )
- ✓ séquentiel : ,

PRIORITE DES OPERATEURS (ORDRE DECROISSANT)

Opérateur	Famille	Associativité
++ -- ! ~ @ (int) (float) (string) (array) (object) -	Inc./déc., logique, bit à bit, erreur, cast, arithmétique unaire	D
* / %	Arithmétique	G
+ - .	Arithmétique et concaténation	G
>> <<	Décalage	G
> >= < <=	Relationnel	Aucune
== === != !==	Relationnel	Aucune
&	Bit à bit	G
^	Bit à bit	G
|	Bit à bit	G
&&	Logique	G
||	Logique	G
? :	Conditionnel	G
= += -= *= /= %= .= &= |= ^= <<= >>=	Affectation	D
and	Logique	G
xor	Logique	G
or	Logique	G
,	Séquentiel	G

Pour un opérateur `op`, l'expression `exp1 op exp2 op exp3`, est l'équivalent de `exp1 op (exp2 op exp3)`, si op a une associativité à droite (D), et de `(exp1 op epx2) op exp3`, lorsque op a une associativité à gauche (G). L'associativité à droite de l'opérateur d'affectation '=' permet d'écrire, par exemple `$x = $y = 9`, la valeur `9` est affectée aux variables `$y` et `$x`.

9. Instructions conditionnelles

Une instruction ci-dessous représente une instruction unique ou un bloc (ensemble d'instructions entre accolades). Une expression exp peut être une entité (un nombre, une chaîne, une variable, une constante, un appel de fonction) ou une combinaison d'entités et d'opérateurs.

if…else	if…elseif…else	switch

`if (expr)`
`  instruction1`
`else`
`  instruction2`
 	
`if (expr)`
`  instruction`
	`if (exp1)`
	`  instruction1`
	`elseif (exp2)`
	`  instruction2`
	`…`
	`elseif (expN)`
	`  instructionN`
	`else`
	`  instructionN+1`
		`switch (exp){`
		`  case val1 : instructions`
		`  case val2 : instructions`
		`  ...`
		`  case valN : instructions`
		`  default  : instructions`
		`}`

Opérateur conditionnel ? :

L'opérateur conditionnel ternaire ' ? : ' évalue l'expression `exp2` si l'expression `exp1` est vraie, sinon c'est l'expression `exp3` qui est évaluée. Attention, ce n'est pas une instruction.

```
exp1 ? exp2 : exp3
```

```
$reste = $h_cou_min < $h_fin_min ?
    $h_fin_min – $h_cou_min : 0;
```

10. Instructions itératives

while	do…while	for	foreach

`while(exp)`
`  instruction`
	`do`
	`  instruction`
	`while(exp);`
		`for(exp1;exp2;exp3)`
		`  instruction`
			`foreach($tab as $el)`
			`  instruction`

11. Instructions d'interruption

Instruction break

Provoque la sortie des instructions `for`, `foreach`, `while`, `do…while`, et `switch`. L'instruction située immédiatement après l'instruction interrompue est exécutée.

Instruction continue

Provoque le passage à l'itération suivante si le test de continuation est encore vrai dans les instructions `for`, `foreach`, `while` et `do...while`.

12. Notation alternative pour les structures de contrôle

Les accolades ouvrante et fermante d'un bloc dans les instructions `if`, `while`, `for`, `foreach` et `switch` peuvent être remplacées respectivement par ':' et une marque de fin dépendante de l'instruction (`endif;` ou `endwhile;` ou `endfor;` ou `endforeach;` ou encore `endswitch;`).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
  <TITLE>horaires de livraison</TITLE>
  <META http-equiv="Content-Type" content="text/html;
        charset=ISO-8859-1">
</HEAD>
<BODY>
<H1>Fastfood en ligne</H1>
<DIV>
<?php
    $heure = date("H");
    if ($heure < 12): ?>
           Bonjour,
<?php
    endif;
    echo "il est ", date("H:i:s");
?>
</DIV>
</BODY>
</HTML>
```

Cette notation alternative peut être utile lorsque, par exemple, l'affichage d'un tableau en HTML est conditionné par la valeur de vérité de l'expression du `if`. Sans cette notation alternative, tout le code HTML du tableau et son contenu devrait être produit avec des instructions `echo` ou `print`.

13. Fonctions

Déclaration

```
function nom_fonction(liste_parametres){
    code PHP
    return expression;
}
```

Appel

```
$res = nom_fonction(liste_parametres) ;
nom_fonction(liste_parametres) ;
```

PHP 3, les déclarations de fonctions devaient être placées avant l'appel.

Paramètres

- ✓ passés par valeur par défaut
- ✓ peuvent être passés par adresse

```
function Convertir_en_min($heure, $min, &$h){
    $h = $heure * 60 + $min;
}
…
// appel
Convertir_en_min(21, 30, $h_fin_min);
```

- ✓ paramètres avec valeurs par défaut
  - o doivent figurer en fin de liste
  - o utiliser le symbole d'affectation suivi de la valeur à affecter (valeur constante)
    `function tva($prix_HT, $taux=19.6)`

Portée des variables globales et locales

- ➢ Les variables globales existent pendant toute la durée de l'exécution du script (statiques).
- ➢ visibles dans tout le script **sauf dans les fonctions** (utiliser le mot-clé `global`)
- ➢ une variable qui apparaît dans une fonction sans avoir été déclarée comme variable globale est considérée comme *locale* à la fonction.

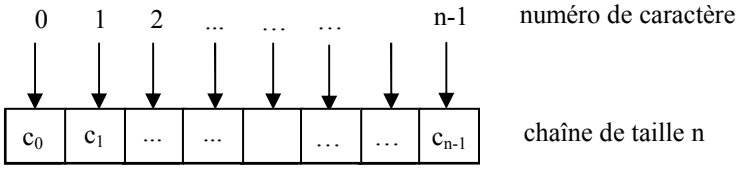
Mémoriser une variable locale

```
<?php
function test(){
   static $x = 20;
   $x--;
   return $x;
}

for ($i = 0; $i < 5; $i++){
   echo test(), "<BR>";
}
?>
```

Affichage : 19, 18, 17, 16 et 15

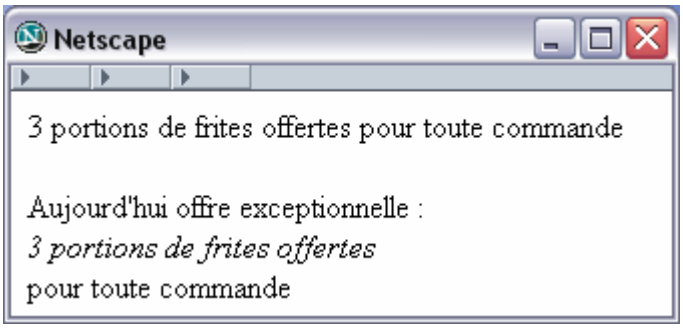
14. Chaînes de caractères



Marque de chaînes de caractères

- Guillemets
- Apostrophes
- Syntaxe here-doc <<<

```
<?php
$nb = 3;
$ch1 = "$nb portions de frites offertes";
$ch2 = 'pour toute commande';
$ch3 = <<<promo
<BR>Aujourd'hui offre exceptionnelle :<BR>
<EM>$nb portions de frites offertes</EM><BR>
pour toute commande
promo;
echo "$ch1 $ch2<BR>";
echo $ch3;
?>
```



Accès à un caractère de la chaîne

`$ch{8}`, permettra de lire ou d'affecter le 9ème caractère de la chaîne.

Caractère d'échappement

Le caractère d'échappement \ est utilisé pour représenter le saut de ligne \n, la tabulation \t, le guillemet \", le caractère '\' \\.

Ne pas confondre \n et
.

QUELQUES FONCTIONS PREDEFINIES POUR LES CHAINES DE CARACTERES

Prototype	Description
string strtolower(string ch)	Retourne ch converti en minuscules
string strtoupper(string ch)	Retourne ch converti en majuscules
string ucfirst(string ch)	Retourne ch avec le 1er caractère en majuscule
string ucwords(string ch)	Retourne ch avec le 1er caractère de chaque mot en majuscule.
string trim(string ch)	Retourne ch sans les espaces de début et fin
string rtrim(string ch)	Retourne ch sans les espaces de fin
string ltrim(string ch)	Retourne ch sans les espaces de début
string chop(string ch)	alias pour rtrim
int strcmp(string ch1, string ch2)	Retourne 0 si les deux chaînes sont égales, un nombre positif si ch1 est alphanumériquement plus grand que ch2, et un nombre négatif sinon.
int strcasecmp(string ch1, string ch2)	Même comportement que strcmp mais sans tenir compte de la casse.
int strncasecmp(string ch1, string ch2, int n)	Même comportement que strcasecmp mais la comparaison est effectuée sur les n premiers caractères
int strpos(string ch, string sc)	Retourne la position de la 1ère occurrence de la sous chaîne sc dans ch, ou FALSE si sc n'est pas dans ch
int strrpos(string ch, string sc)	Retourne la position de la dernière occurrence de sc dans ch, ou FALSE si sc n'est pas dans ch
int strspn(string ch, string enscar)	Retourne la longueur de la plus grande sous chaîne initiale de ch dont les caractères sont dans enscar
int strcspn(string ch, string enscar)	Retourne la longueur de la plus grande sous chaîne initiale de ch qui ne comporte aucun caractère de enscar
string strstr(string ch, string sc) ou son alias strchr	Retourne la portion de ch à partir de la 1ère occurrence de sc jusqu'à la fin
string strrchr(string ch, string car)	Retourne la sous chaîne qui commence à la dernière occurrence du caractère car dans ch
string stristr(string ch, string sc)	strstr sans tenir compte de la casse
string substr(string ch, int deb, int n)	Renvoie la sous chaîne de taille n qui commence à l'indice deb dans ch (n est optionnel, s'il n'est pas précisé retourne la sous chaîne de l'indice deb jusqu'à la fin de la chaîne)
int strlen(string ch)	Retourne le nombre de caractères de ch
string substr_replace(string ch, string rep, int deb, int n)	Remplace, à partir de l'indice deb, la sous-chaîne de n caractères par rep dans la chaîne ch, et retourne la chaîne modifiée
string strtr(string ch, string rech, string rep)	Remplace dans ch chaque caractère de rech par le caractère correspondant de rep
string str_repeat (string ch, int n)	Retourne la chaîne ch répétée n fois
string strrev(string ch)	Retourne la chaîne ch inversée
array explode(string sep, string ch, int n)	Sépare ch selon le séparateur sep et place chaque sous chaîne dans une case de tableau, dans la limite de n cases si n est précisé (la dernière case contient alors la fin de la chaîne)

15. Tableaux

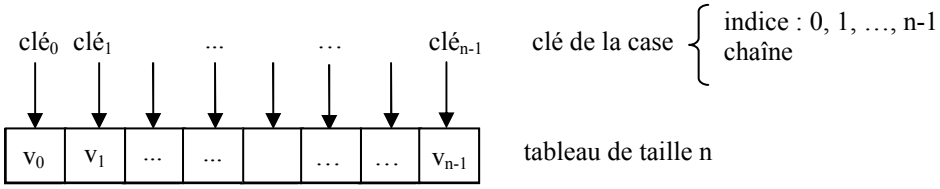


Tableau avec accès par indice

Initialisation et ajout de cases

```
$tab = array("b","o","n");
$tab[0] = "b"; $tab[1] = "o"; $tab[2] = "n";
$tab[] = "b"; $tab[] = "o"; $tab[] = "n";
```

Parcours

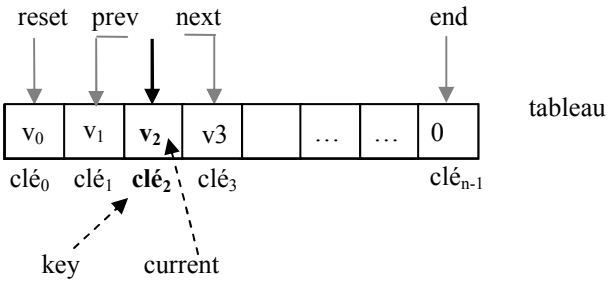
```
foreach ($tab as $i){
    echo "$i-";
```

Tableau associatif

Initialisation

```
$nomtab["cle"] = valeur;
$nomtab = array("cle1"=>valeur1,"cle2"=>valeur2,...);
```

Parcours




```
for (reset($nomtab) ; $cle = key($nomtab) ; next($nomtab)){
  …
}

reset($nomtab);
while (list($cle, $valeur) = each($nomtab)){
  …
}

foreach ($nomtab as $cle=>$val){
   …
}
```

```php
<?php

//initialisation 1
$tarif["pizza napolitaine"] = 6;
$tarif["pizza royale"] = 8;
$tarif["pizza aux fruits de mer"] = 8.50;
$tarif["moules frites"] = 7.30;
$tarif["poulet frites"] = 6;

//initialisation 2
$stock = array("poulets frites"=>50, "pizzas"=>22, "moules
frites"=>8);

//parcourir le tableau avec for
echo "<BR>parcours avec for : ";
for (reset($tarif) ; $cle = key($tarif) ; next($tarif)){
  $val = pos($tarif);
  echo "<BR>$cle = $val euros";
}

//parcourir le tableau avec while
echo "<BR>premier parcours avec while : ";
reset($stock);
while (list($cle, $valeur) = each($stock)){
  echo "<BR>nombre de $cle = $valeur portions";
}

echo "<BR>deuxieme parcours avec while : ";
while (list($cle, $valeur) = each($stock)){
  echo "<BR>nombre de $cle = $valeur portions";
}

// parcourir le tableau avec foreach
echo "<BR>parcours avec foreach : ";
foreach ($stock as $cle=>$val){
        echo "<BR>$cle = $val portions";
}
?>
```

```
parcours avec for :
pizza napolitaine = 6 euros
pizza royale = 8 euros
pizza aux fruits de mer = 8.5 euros
moules frites = 7.3 euros
poulet frites = 6 euros
premier parcours avec while :
nombre de poulets frites = 50 portions
nombre de pizzas = 22 portions
nombre de moules frites = 8 portions
deuxieme parcours avec while :
parcours avec foreach :
poulets frites = 50 portions
pizzas = 22 portions
moules frites = 8 portions
```

PHP propose de nombreuses fonctions prédéfinies pour les tableaux, en voici quelques unes :

Prototype	Description
void sort (array t)	Trie t par ordre croissant
void rsort (array t)	Trie t par ordre décroissant
void asort (array t)	Trie le tableau associatif t par ordre croissant de valeurs
void arsort (array t)	Trie le tableau associatif t par ordre décroissant de valeurs
void ksort (array t)	Trie le tableau associatif t par ordre croissant de clés
void krsort (array t)	Trie le tableau associatif t par ordre décroissant de clés
void natsort(array t)	Trie t par ordre naturel
void natcasesort(array t)	Idem sans tenir compte de la casse
Int array_push (array t, liste_arg)	Empile les éléments de la liste (qui peuvent être de types différents) à la fin du tableau et retourne le nombre d'éléments du tableau
type_element array_pop (array t)	Dépile et retourne le dernier élément de t, le type retourné est celui de l'élément
type_element array_shift(array t)	Supprime et retourne la première valeur de t, le type retourné est celui de l'élément
int array_unshift (array t, liste_arg)	Ajoute les éléments de la liste (qui peuvent être de types différents) au début du tableau et retourne le nombre d'éléments du tableau
array array_splice(array t, int deb, int n, array remp)	Retourne n éléments de t à partir de l'indice deb, et les remplace par les éléments de remp. Si remp n'est pas donné, les éléments sont supprimés.
array array_keys(array t)	Retourne les clés de t
array array_values(array t)	Retourne les valeurs de t
boolean array_key_exists(array t)	Retourne TRUE si la clé existe
boolean in_array(type_element rech, array t)	Retourne TRUE si rech (dont le type n'est pas prédéfini) est dans t
type_cle array_search(type_element rech, array t)	Retourne la clé (un entier si c'est un tableau a accès par indice) de l'élément rech dans t, ou FALSE si rech n'est pas dans t.
type_element min(array t)	Retourne la valeur minimale de t. Le type retourné est celui de l'élément minimal
type_element max(array t)	Retourne la valeur maximale de t. Le type retourné est celui de l'élément maximal
array array_reverse(array t)	Retourne le tableau t dans l'ordre inverse
void shuffle(array t)	Mélange aléatoirement les cases de t
array array_slice(array t, int deb, int n)	Retourne n éléments de t à partir de l'indice deb. Si n n'est pas précisé, retourne les éléments jusqu'à la fin
array array_unique(array t)	Retourne le tableau t sans les doublons
array array_merge(array t1, array t2, ...)	Retourne un tableau qui est composé des cases des tableaux de la liste
array explode(string sep, string ch)	Transforme une chaîne ch en tableau en fonction du séparateur sep.
string implode(string sep, array t)	Opération inverse

Il faut noter que pour ajouter des éléments en fin de tableau, il n'est pas nécessaire d'utiliser array_push, en effet, $nomtab[] = val, ajoutera une case au tableau et lui associera la valeur val. De même $nomtab["nomcle"] = val permettra d'ajouter une case à un tableau associatif. Cette case sera accessible par la clé nomcle et contiendra la valeur val.

16 Expressions rationnelles

INDICATEURS D'OCCURRENCES

Symbole	Signification	Expression	Correspondance
*	Zéro occurrence ou plus	`a*`	'', 'a', 'aa', ...
+	Au moins une occurrence	`a+`	'a', 'aa', 'aaa', …
?	Zéro ou une occurrence	`a?`	'', 'a'
{n}	Exactement n occurrences	`a{3}`	'aaa'
{n,}	Au moins n occurrences	`a{2,}`	'aa', 'aaa', …
{n, m}	Au moins n occurrences et au plus m	`a{1,3}`	'a', 'aa', 'aaa'

ALTERNATIVE : |

CARACTERES SPECIAUX ET CLASSES DE CARACTERES

Symbole	Signification	Expression	Correspondance
.	Un caractère quelconque	`.u`	ou
[liste_car]	Tout caractère de la liste	`[ruBjno]+`	Bonjour
[^liste_car]	Tout sauf un caractère de la liste	`h[^s]+`	heure
-	intervalle	`[A-Za-z]*` `[0-9]`	Bonjour 9
\n	Nouvelle ligne	**« Bonjour, il est 9 heures »**	
\r	Retour chariot		
\t	Tabulation		

CLASSES DE CARACTERES PREDEFINIES DU STANDARD POSIX

Symbole	Signification	Expression	Correspon-dance
[:alpha:]	Lettres = [A-Za-z]	`[[:alpha:]]+`	Bonjour
[:lower:]	Lettres minuscules = [a-z]	`[[:lower:]]+`	onjour
[:upper:]	Lettres majuscules = [A-Z]	`[[:upper:]]+`	B
[:digit:]	Chiffres = [0-9]	`[[:digit:]]+`	9
[:alnum:]	= [A-Za-z0-9]	`[[:alnum:]]+`	Bonjour
[:xdigit:]	[0-9a-fA-F]	`[[:xdigit:]]`	B
[:punct:]	Ponctuation	`[[:punct:]]`	,
[:blank:]	Espace	`([A-Za-z]+[[:blank:]])+`	il est
[:space:]	Caractères d'espacement		

POSITIONS DE LA CORRESPONDANCE DANS LA CHAINE

Symbole	Signification	Expression	Correspondance
^	Correspondance en début de chaîne	`^i.*` `^B.{6}`	*aucune* Bonjour
$	Correspondance en fin de chaîne	`.{5}s$` `B.{6}$`	heures *aucune*
[:<:]	Le mot commence par	`[[:<:]]e.+`	est 9 heures
[:>:]	Le mot termine par	`.*t[[:>:]]`	Bonjour, il est

FONCTIONS POUR LES EXPRESSIONS RATIONNELLES

Prototype	Description
boolean ereg(string exp, string ch, array t)	Retourne `TRUE` s'il y a une correspondance de `exp` dans `ch`, `FALSE` sinon. La correspondance est stockée dans la case 0 de `t`, les cases suivantes contiennent, s'il y a lieu, les sous occurrences (`t` comporte toujours 10 cases).
boolean eregi(string exp, ch, array t)	Comme `ereg` sans tenir compte de la casse
string ereg_replace(string exp, string rep, string ch)	Retourne la chaîne `ch` dans laquelle les sous chaînes qui correspondent à l'expression `exp` ont été remplacées par la chaîne `rep`. Si des sous occurrences sont définies, elles peuvent être représentées dans `rep` en utilisant `\\m`, `m` étant le numéro de la sous occurrence dans `exp`.
boolean eregi_replace(string exp, string rep, string ch)	Comme `ereg_replace` sans tenir compte de la casse
array split(string exp, string ch, int n)	Retourne un tableau à partir de la chaîne `ch` en utilisant l'expression `exp` comme séparateur. Si une limite `n` est donnée et qu'elle est atteinte en séparant la chaîne en cases, alors la dernière case contiendra la fin de la chaîne
array spliti(string exp, string ch, int n)	Comme `split` sans tenir compte de la casse
string sql_regcase(string exp)	Convertit une expression rationnelle en une expression rationnelle non sensible à la casse

```php
<?php
$info_heure = "Il est ".date("H:i:s");
echo "$info_heure<BR>";

$exp = "([0-9]{2}):([0-9]{2}):([0-9]{2})";
if (ereg($exp, $info_heure, $tabocc)){
    echo "$tabocc[0]<BR>";
}
else{
    echo $info_heures,"<BR>";
}
$info_heure = ereg_replace($exp, "\\1 h \\2 mn et \\3 s", $info_heure);
echo $info_heure;
?>
```

```
Il est 11:49:58
11:49:58
Il est 11 h 49 mn et 58 s
```

17 Date et temps

`timestamp` = entier représentant le nombre de secondes écoulées depuis le 1er janvier 1970.

- int `time()` retourne le `timestamp` courant.
- int `mktime(int h, int mn, int s, int jour, int mois, int année)`.

```
echo mktime(14, 38, 20, 6, 2, 2003); // 2 juin 2003 à 14h38mn20s
```

```
1054557500
```

- `getdate` retourne un tableau associatif contenant les informations de date et temps définies par le `timestamp` passé en argument. Si celui-ci n'est pas précisé, c'est le `timestamp` de l'instant où le script est exécuté qui est utilisé.

CLES POUR GETDATE

Clé	Valeur
hours	Heure au format 24
minutes	Minutes
seconds	Secondes
year	Année sur 4 chiffres
mon	Mois entre 1 et 12
month	Nom du mois
mday	Jour du mois entre 1 et 31
yday	Jour de l'année
wday	Jour entre 0 (dimanche) et 6 (samedi)
weekday	Nom du jour
0	Timestamp

```
<?php
$d = getdate();
echo "Il est ", $d["hours"]."h ",$d["minutes"], "mn, ";
echo "nous sommes en ", $d["year"];
reset($d);
while (list($cle, $val) = each($d)){
        echo "<BR>$cle = $val";
}
?>
```

```
Il est 14h 31mn, nous sommes en 2003
seconds = 12
minutes = 31
hours = 14
mday = 24
wday = 5
mon = 10
year = 2003
yday = 296
weekday = Friday
month = October
0 = 1066998672
```

- `boolean checkdate`(mois, jour, année)

```
<?php
if (checkdate(2, 29, 2004)){
    echo "Le mois de f&eacute;vrier 2004 a 29 jours";
}
else{
    echo "Le mois de f&eacute;vrier 2004 a 28 jours";
}
?>
```

```
Le mois de février 2004 a 29 jours
```

- `date` prend en argument un format de date et éventuellement un `timestamp` (sinon, c'est le `timestamp` courant qui est utilisé) et retourne une chaîne de caractères représentant le `timestamp` selon le format précisé. .

CARACTERES POUR LE FORMAT DES DATES

Caractère	Signification	Caractère	Signification
H	heure de 00 à 23	h	heure de 01 à 12
G	heure de 0 à 23	g	heure de 1 à 12
a	ajoute am ou pm	A	ajoute AM ou PM
i	minutes 00 à 59	s	secondes 00 à 59
Y	année (4 chiffres)	y	année (2 chiffres)
z	numéro du jour de l'année	w	0 (dimanche) à 6 (samedi)
m	mois de 01 à 12	n	mois de 1 à 12
d	jour de 01 à 31	j	jour de 1 à 31
M	nom du mois (3 caractères)	F	nom du mois
D	jour (3 caractères)	l	le nom du jour

```
<?php
$t = time();
echo date("H\h i\m\\n s\s", $t), "<BR>";
echo date("hA i s", $t), "<BR>";
echo date("d m Y", $t), "<BR>";
?>
```

```
14h 38mn 20s
02PM 38 20
02 06 2003
```

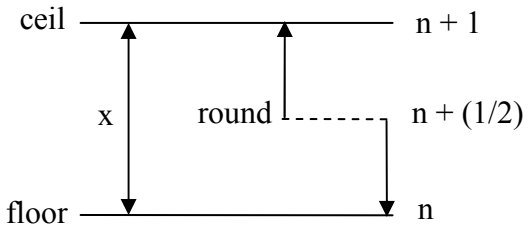
18 Mathématiques

Trigonométrie

sinus, cosinus, tangente, arc sinus, arc cosinus, arc tangente, sinus et arc sinus hyperbolique, cosinus et arc cosinus hyperbolique et tangente et arc tangente hyperbolique (`sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `sinh`, `asinh`, `cosh`, `acosh`, `tanh`, `atanh`). Ces fonctions prennent toutes en argument un nombre réel et retournent un nombre réel. La fonction `atan2` (arc tangente de deux variables) prend deux arguments de type réel et retourne un résultat réel. La valeur pi peut être obtenue en utilisant la constante `M_PI` ou la fonction `pi()`.

Arrondis

Un nombre réel x peut être arrondi au plus petit entier n+1 qui lui est supérieur (`ceil`), à sa partie entière n (`floor`) ou en fonction de la moitié de l'intervalle (`round`).



Extrema

La fonction `min` et la fonction `max` retournent respectivement le minimum et le maximum de la liste d'arguments, ces arguments peuvent être des entiers ou des réels. La valeur retournée est du type de l'argument minimum ou maximum.

Changements de bases et conversions

```
echo "decbin = ", decbin(255), " et ";
echo "bindec = ", bindec("11111111"), "<BR>";
echo "dechex = ", dechex(255), " et ";
echo "hexdec = ", hexdec("FF"), "<BR>";
echo "decoct = ", decoct(255), " et ";
echo "octdec = ", octdec("377"), "<BR>";

echo "basec = ", base_convert('11111111', 2, 10), "<BR>";
echo "basec = ", base_convert('255', 10, 16), "<BR>";
echo "basec = ", base_convert('377', 8, 16), "<BR>";
```

```
decbin = 11111111 et bindec = 255
dechex = ff et hexdec = 255
decoct = 377 et octdec = 255
basec = 255
basec = ff
basec = ff
```

Nombres aléatoires

`rand` et `mt_rand` permettent d'obtenir des nombres pseudo-aléatoires (la seconde est beaucoup plus rapide que la première). Ces fonctions retournent un entier compris entre les bornes entières minimale et maximale données en argument (bornes incluses). Il est possible de connaître le plus grand nombre aléatoire qui peut être généré en utilisant la fonction `getrandmax` pour `rand` ou `mt_getrandmax` pour `mt_rand`. Le générateur de nombre pseudo-aléatoires peut être initialisé en utilisant la fonction `srand` pour les nombres qui seront générés avec `rand` ou `mt_srand` pour ceux produits par `mt_rand`. Il faut noter cependant que depuis PHP 4.2, cette initialisation est effectuée automatiquement. Ces deux fonctions ne retournent rien, elles initialisent le générateur en utilisant l'entier qui est passé en paramètre. Pour obtenir un nombre aléatoire compris entre 0 et 1, il faut utiliser la fonction `lcg_value` qui retourne un réel.

```
<?php
echo "max = ", getrandmax(), "<BR>";
echo "max = ", mt_getrandmax(), "<BR>";
for ($i = 0; $i < 10; $i++){
        echo mt_rand(0, 9), " ";
}
echo "<BR>";
for ($i = 0; $i < 10; $i++){
        echo mt_rand(0, 9), " ";
}
echo "<BR>";
echo lcg_value(), " ";  ?>
```

```
max = 2147483647
max = 2147483647
5 3 9 7 4 5 0 2 2 7
5 1 7 2 7 7 6 6 7 5
0.97800633881491
```

Logarithmes et exponentielle

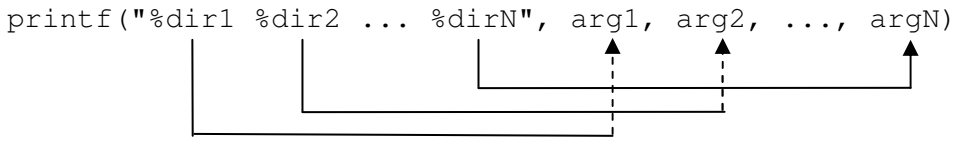
L'exponentielle, le logarithme et le logarithme base 10 sont obtenus respectivement en utilisant les fonctions `exp`, `log` et `log10` qui retournent un réel et prennent comme argument un réel. Le logarithme peut être défini depuis la version 4.3 pour une base `b` donnée en deuxième argument.

Autres fonctions

Prototype	Description
int abs(int x) ou float abs(float x)	Valeur absolue de `x`
float sqrt(float x)	Racine carrée de `x`
float pow(float x, float y) ou int pow(int x, int y)	Elève `x` à la puissance `y`, si c'est impossible retourne `FALSE`
boolean is_finite(float x)	Retourne `TRUE` si `x` est un nombre fini compris dans les bornes d'un réel.
boolean is_infinite(float x)	Retourne `TRUE` si `x` est infini ou ne peut pas être représenté par un réel.
boolean is_nan(float x)	Retourne `TRUE` si `x` n'est pas un nombre

19 Formatage des données

printf



En l'absence d'argument après la chaîne de format, printf réalise un affichage du contenu de la chaîne, les deux instructions suivantes sont donc équivalentes :

```
printf("bonjour") ;  echo "bonjour" ;
```

Une directive de formatage permet de définir : le caractère de *complétion*, l'*alignement*, la *taille*, la *précision* des nombres réels, et le *type*. Cet ordre doit être respecté, mais il n'est pas nécessaire de préciser chacune des options de formatage, en effet, chaque option a une valeur par défaut.

La directive de type permet d'afficher un argument selon le type précisé, ainsi un nombre entier décimal pourra être affiché en octal.

SYMBOLES DE TYPES POUR FORMATER DES DONNEES

Symbole	Signification
b	Entier sous forme binaire
d	Entier décimal
f	Réel
o	Entier sous forme octale
s	Chaîne de caractères
u	Entier non signé
x	Entier sous forme hexadécimale
X	Entier sous forme hexadécimale en majuscules

Le caractère de complétion est utilisé pour compléter le résultat affiché lorsqu'il n'atteint pas la taille demandée. Par exemple, si nous souhaitons afficher une variable `$nb1` sur 3 caractères et que sa valeur est 8, les deux caractères manquants pourront être complétés par le caractère précisé. Les caractères de complétion possibles sont le caractère zéro ou tout autre caractère préfixé par une apostrophe. Par défaut, c'est l'espace qui est utilisé.

La directive d'alignement '-' permet d'aligner le résultat à droite, par défaut c'est un alignement à gauche qui est réalisé. Cet alignement est lui aussi utilisé lorsque le résultat n'atteint pas la taille demandée.

La directive de taille permet de préciser le nombre de caractères minimal à afficher.

La directive de précision s'applique uniquement aux nombres réels, elle permet de préciser le nombre de caractères exact à afficher après la virgule. Elle aura pour effet de tronquer les caractères après le chiffre précisé.

sprintf

`sprintf` retourne une chaîne formatée selon un format spécifié par une chaîne de formatage. Par exemple, `$ch = sprintf("%03d", 1)` placera la chaîne « 003 » dans la variable `$ch`.

20 Types et conversions

Vérifier si une variable est d'un type donné en utilisant les fonctions préfixées par is_ :

- is_int, is_integer, is_long, retournent TRUE si la variable est de type entier,
- is_float, is_double, is_real retournent TRUE lorsque l'argument est un réel,
- is_string retourne TRUE lorsque la variable est une chaîne de caractères,
- is_numeric retourne TRUE lorsque la variable contient une valeur numérique (entier, réel ou chaîne de caractères comportant uniquement des chiffres),
- is_bool retourne TRUE lorsque la variable est booléenne,
- is_null retournera TRUE si la variable ne contient pas de valeur ou a été affectée avec la constante NULL,
- is_resource retourne TRUE si la variable est une ressource,
- is_array retourne TRUE lorsque la variable est un tableau,
- is_scalar retourne TRUE si la variable est une valeur scalaire (la fonction retournera notamment FALSE pour les variables de types array car un tableau est une collection de données).

La fonction `gettype` retourne une chaîne de caractères contenant le type de la variable passée en argument.

Convertir des données en entier, réel, et chaîne de caractères : fonctions `intval`, `doubleval` ou `floatval` et `strval`. Ces fonctions convertissent l'argument dans le type souhaité.

L'opérateur de `cast` (…) permet également d'effectuer des conversions. L'expression auquel il est appliqué est convertie dans le type précisé entre parenthèses. Dans l'exemple ci-dessous, `(int)` devant `$nb_stock` ne change pas le type de `$nb_stock`, l'expression `$nb_stock/2` est convertie en entier, le résultat de la division est donc la partie entière de 12.5.

```
<?php
$nb_stock = 25;
$a = doubleval($nb_stock);
echo gettype($a),"<BR>";
echo (int)($nb_stock/2); // cast
?>
```

```
double
12
```