Institut Universitaire de Technologie
Aix*Marseille Université

PHP

PROGRAMMER UN SITE WEB DYNAMIQUE

LES BASES DU LANGAGE

Les bases du langage

PRINCIPE DE LA CONSULTATION D'UNE PAGE WEB

Le protocole HTTP

Qu'est-ce que le protocole HTTP ?

Le rôle de HTTP (*HyperText Transfer Protocol*) est d'assurer le transport d'informations relatives aux échanges portés sur le contenu Web, il définit la communication entre un serveur et un client.

Que se passe-t-il lorsque je navigue sur internet ?

Quand je clique sur un lien hypertext, je transmet à mon navigateur une URL. Celui-ci sait, alors, quel serveur contacter et quel fichier demander.

C'est à ce moment que le protocole http entre en action : le transfert du fichier depuis le serveur jusque sur mon navigateur !

La communication entre serveur et client est réalisé par des requêtes http

Le protocole HTTP

Méthodes définies par le protocole HTTP :

Get : Requête la ressource située à l'URL spécifiée.

Head : Requête la ressource située à l'URL spécifiée (la réponse ne contient que l'entête, et pas le contenu de la ressource).

Post : Envoi de données à destination de la page situé à l'URL spécifiée

Put : Envoi des données à l'URL spécifiée

Delete : Suppression de la ressource située à l'URL spécifiée

Exemple de commande :

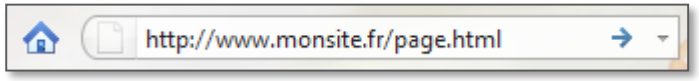
GET / HTTP/1.1
host: www.google.com

-> on peut par exemple tester cette commande via un telnet

telnet www.google.com 80
GET / HTTP/1.1
host: www.google.com

On si on essai avec iut.univ-amu.fr ?

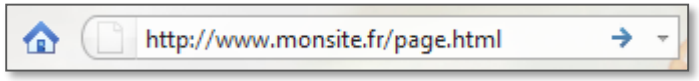
Consultation d'une page HTML



Navigateur

Serveur Web

Consultation d'une page HTML



❶ Demande d'une page HTML.

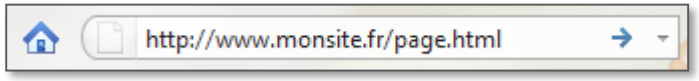
Exemple de requête http générée par le client (requête simpliste)
GET /page.html Http1.1
Host: www.monsite.fr
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)
Connection: Keep-Alive

Navigateur

http://www.monsite.fr/page.html

Serveur Web

Consultation d'une page HTML



❶ Demande d'une page HTML.

http://www.monsite.fr/page.html

HTML

Serveur Web

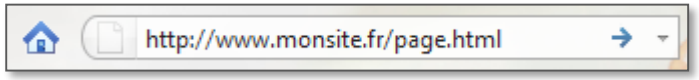
Navigateur

Le serveur répond à la requête http du client
HTTP/1.1 200 OK
Content-type: text/html

.. Envoi du fichier html demandé

❷ Le serveur reçoit la demande et transmet la page HTML.

Consultation d'une page HTML



❶ Demande d'une page HTML.

❸ Le client reçoit et interprète la page.

http://www.monsite.fr/page.html

HTML

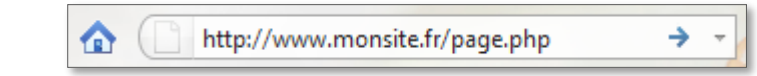
Serveur Web

Navigateur

❷ Le serveur reçoit la demande et transmet la page HTML.

Pour plus de détail sur le protocole HTTP 1.1
http://julien-pauli.developpez.com/tutoriels/web/http/
Signification des codes HTTP
http://fr.wikipedia.org/wiki/Liste_des_codes_HTTP

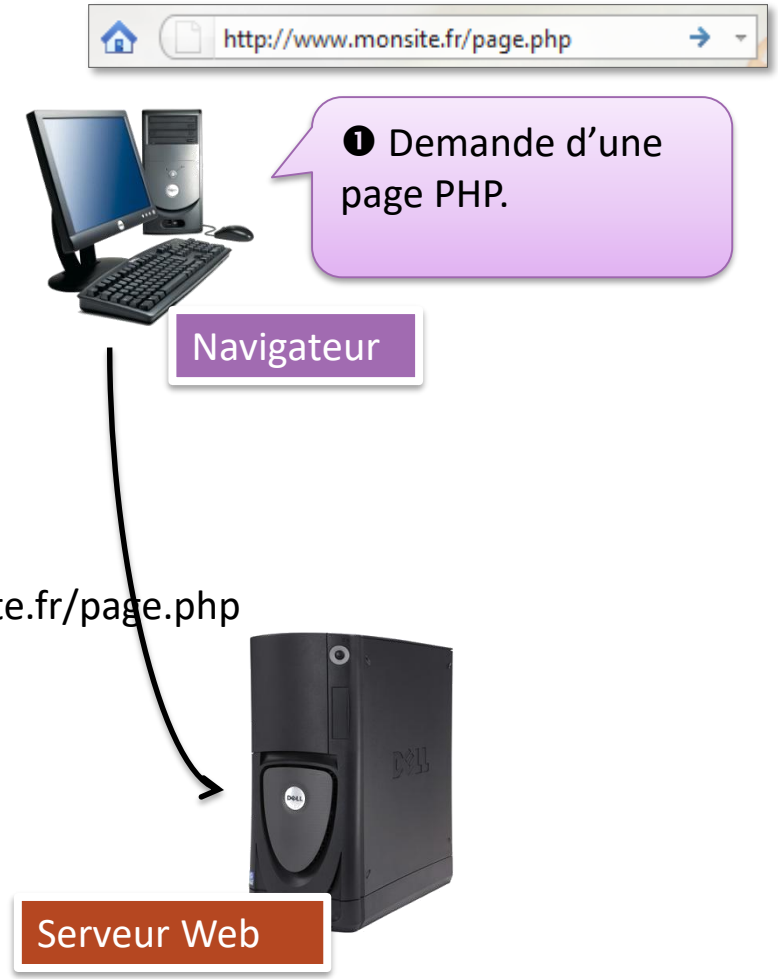
Consultation d'une page PHP



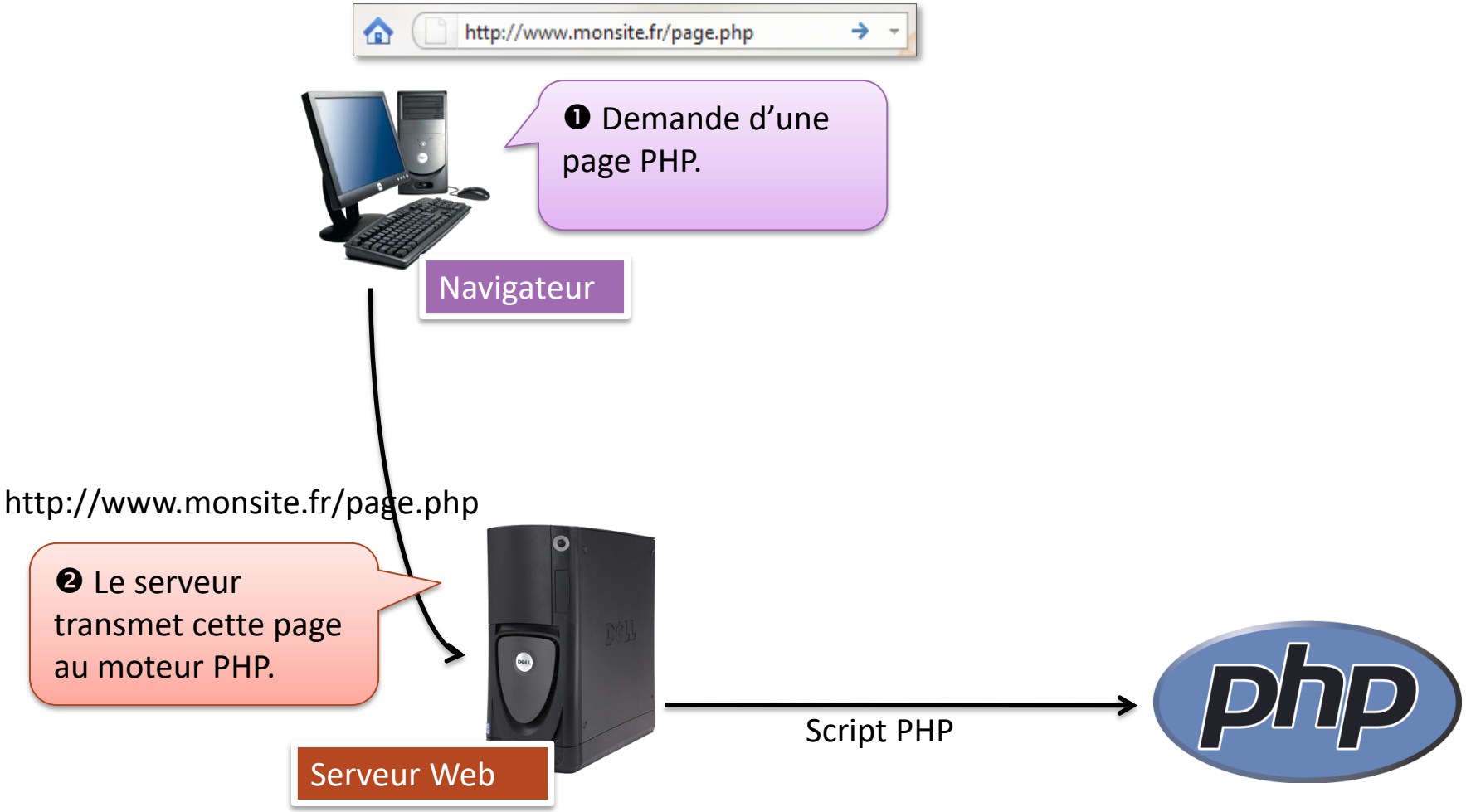
Navigateur

Serveur Web

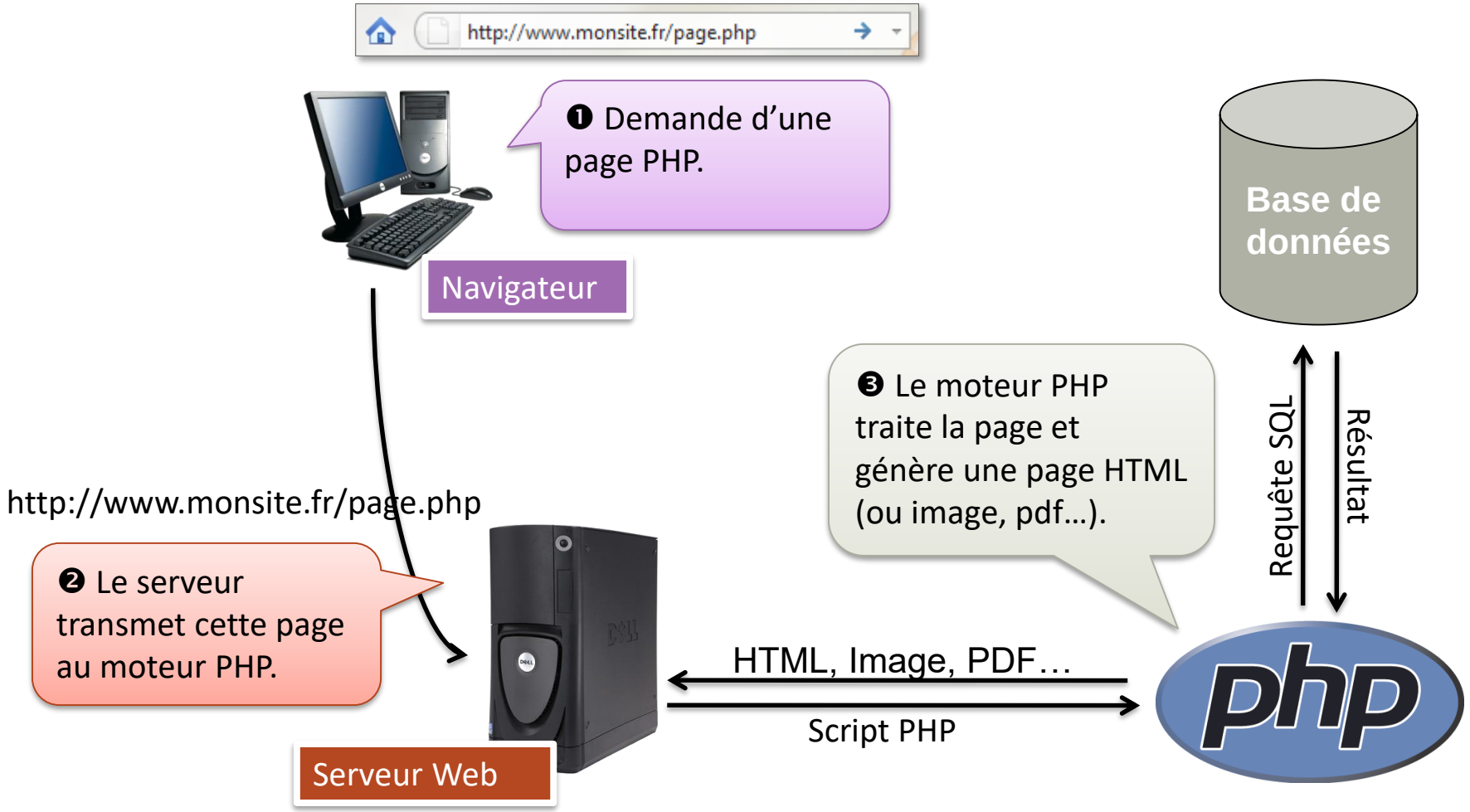
Consultation d'une page PHP



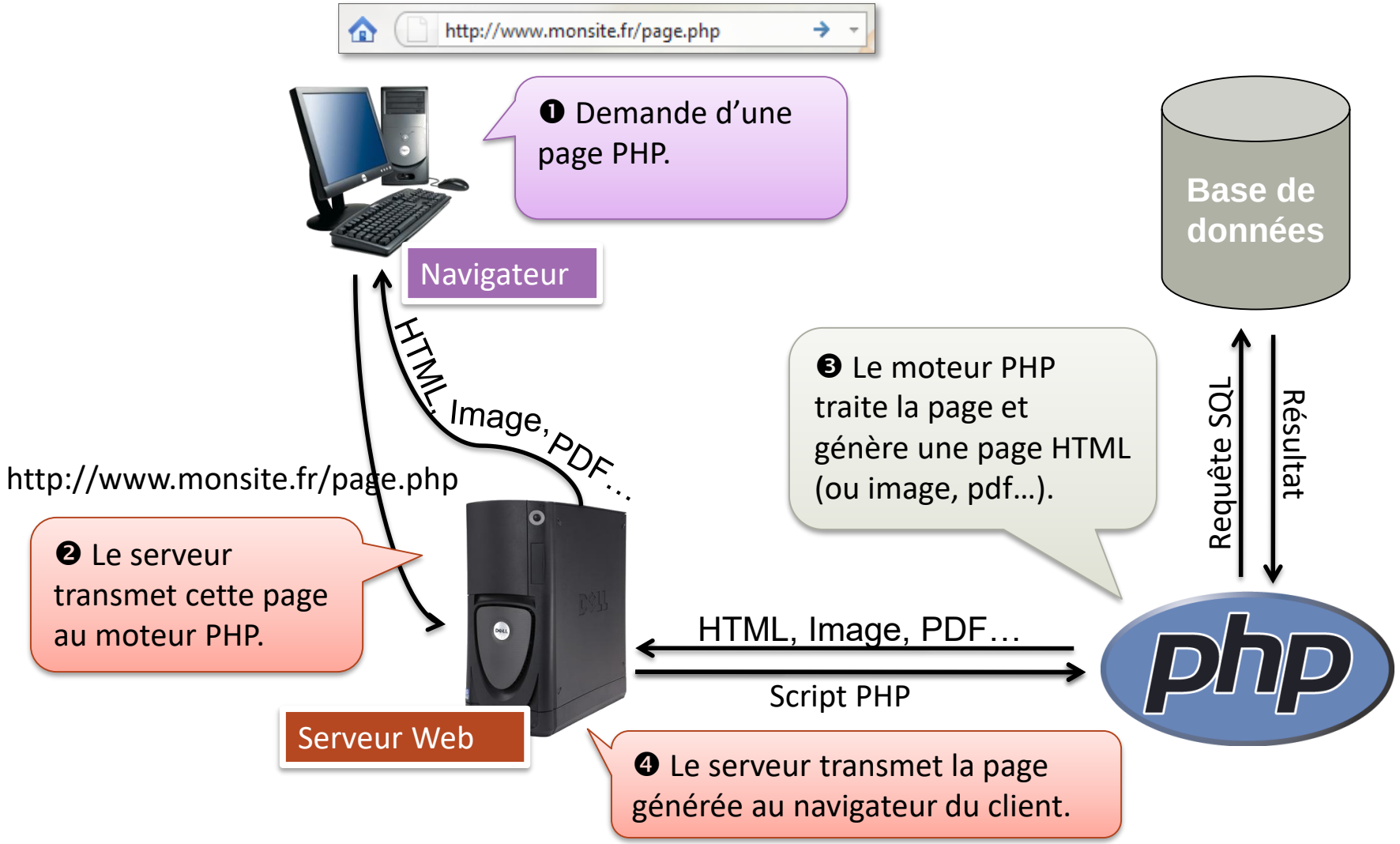
Consultation d'une page PHP



Consultation d'une page PHP



Consultation d'une page PHP

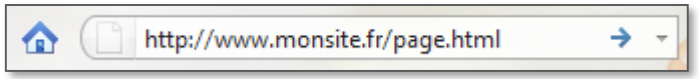


Qu'est-ce que cela change ?

Consultation d'une page HTML

La page est retournée telle qu'elle est sur le serveur puis elle est interprétée coté client.

*On parle dans ce cas de **page statique**.*



Consultation d'une page PHP

La page est construite côté serveur en interprétant les éléments PHP avant d'être renvoyée au client qui pourra alors interpréter le contenu généré par le serveur.

*On parle dans ce cas de **page dynamique**.*

HTML vs PHP – Sites de petites annonces

Cas d'une page statique

Exemple pour un site de petites annonces

La fiche d'une annonce correspond à une page HTML.

L'ajout d'une petite annonce implique de créer une nouvelle page.

Avantages	Inconvénients
Facile à mettre en œuvre	Fastidieux s'il y a beaucoup de pages identiques
Nécessite peu de connaissances	Difficile à maintenir d'un point de vu contenu et cohérence visuelle.

HTML vs PHP – Sites de petites annonces

Cas d'une page dynamique

- ***Exemple pour un site de petites annonces***
- La fiche d'une petite annonce est construite à partir d'une entrée d'une base de données.
- L'ajout d'une annonce nécessite d'ajouter une entrée dans la base de donnée.

Avantages	Inconvénients
Cohérence visuelle aisé – toutes les pages sont construites sur le même modèle.	Nécessite des compétences en programmation.
Permet la mise en place d'un back-office pour maintenir le contenu.	

Les bases du langage

INTRODUCTION AU LANGAGE

PHP - Un peu d'histoire…

1994 : Bibliothèque en Perl écrite par Rasmus Lerdorf pour conserver une trace des consultations de son CV.

1994-1995 : PHP/FI (Personnal Home Page Form Interpreter)

- Réécriture complète en C.
- Ajout de fonctionnalités (ex. communication avec les BD).

1997-1998 : réécriture du cœur par Andi Gutmans et Zeev Surask

- Intégration d'une API modulaire
- PHP 3 (**P**HP : **H**ypertext **P**reprocessor).

1998-2002 : Nouvelle réécriture du moteur

- Développement du *Zend Engine*
- PHP4 : prise en charge des sessions HTTP

2002-2006 : PHP5 avec Zend Engine 2, PDO, SQLite, les objets, etc.

Intégration d'un script dans une page

Le code PHP peut-être directement intégré à différents endroits dans les fichiers HTML.

Pour insérer du code dans le HTML, il suffit de le placer entre les balises :

<?php et ?> (tags formels du langage)
<? et ?> (tags courts très employés par les webmasters débutants)
<?= et ?> (raccourci du tag <?php echo…)
<% et %> (tag issus du langage ASP - rarement utilisés , supprimé depuis php 7.0.0)
<script language="php"> et </script?>
(très rarement utilisés, supprimé depuis php 7.0.0)

Les page contenant du php doivent porter l'extension .php3 ou .php (version > 3)

Intégration d'un script dans une page
Pourquoi préférer <?php et ?>

Parce que les tags <?php et ?> assurent une portabilité totale sur tous les serveurs et toutes les versions de PHP. Ce sont les tags par défaut du langage PHP.

Les « *short-tags* » (<? et ?>) pourraient empêcher l'exécution de vos scripts pour les deux raisons suivantes :

1. **Le serveur qui héberge vos pages PHP désactive l'utilisation de ces balises par la directive *short_open_tags* du php.ini placée à la valeur *off*.**

Intégration d'un script dans une page
Pourquoi préférer <?php et ?>

2. **Il y a une confusion avec la balise d'ouverture d'un fichier XML.**

En effet un fichier XML débute par la syntaxe suivante :

Prologue d'un fichier XML

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

On remarque la présence du short tag <? au tout début du code et ?> à la fin. A la lecture du fichier l'interprêteur PHP tentera d'exécuter cette ligne (pensant que c'est du PHP) et renverra une erreur d'analyse similaire à celle ci-dessous :

Parse error: syntax error, unexpected T_STRING in /Applications/MAMP/htdocs/Tests-PHP/xml.php on line 1

Commentaires

Un script PHP se commente comme en JavaScript :

```
<?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.

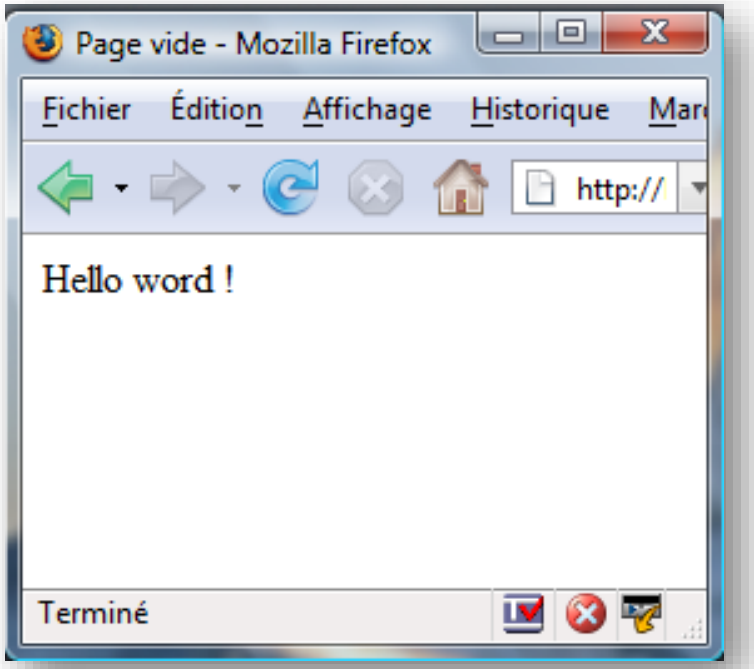
Il est conseillé de commenter largement ses scripts.

Mon premier script

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
  <title>Page vide</title>
  <meta http-equiv="content-type"
    content="text/html;charset=utf-8" />
  <meta http-equiv="Content-Style-Type" content="text/css" />
</head>

<body>

  <?php
  echo "Hello word !";
  ?>

</body>
</html>
```

Fichier PHP sur le serveur

Mon premier script



Résultat affiché sur le navigateur du poste client

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
        <title>Page vide</title>
        <meta http-equiv="content-type"
                content="text/html;charset=utf-8" />
        <meta http-equiv="Content-Style-Type" content="text/css" />
</head>

<body>

  Hello word !
</body>
</html>

```

Code source généré

Les bases du langage

VARIABLES & CONSTANTES

Déclaration

Pas de déclaration : une variable est implicitement déclarée dès qu'elle est utilisée.

Le type de la variable se détermine en fonction du contenu de la variable et peut évoluer au fur et à mesure des utilisations.

Exemple :

```
$prix = 100;
$nom = "Franck";
```

Identifiants

Syntaxe :

- sensibles à la casse
- commencent toujours par $
- continuent par une lettre ou par _ mais pas un chiffre
- se finissent par un nombre quelconque de lettres, chiffres et _

Convention de nommage :

- **les caractères accentués sont possibles, mais fortement déconseillés**
- la première lettre d'une variable est une minuscule
- chaque changement de mot est précédé du caractère _ et la première lettre du mot est en minuscule

Porté

Variables locales :

visibles que dans le contexte (block) où elles ont été créées.

Par défaut les variables de PHP sont des variables locales.

Pour utiliser une variable dans un autre contexte, il faut la passer en paramètre, ou utiliser une variable globale. $_GLOBALS …

Porté

Variables globales :

Utilisation de la super-globale *$GLOBALS[]*

```php
<?php
function message($_msg1,$_msg2)
{
   echo $_msg1." ".$GLOBALS['sNom']." ".$_msg2." ".$GLOBALS['sPrenom'];
}

$sNom="Cangimrac";
$sPrenom="Laurent";
message("Mon nom est","Mon prénom est")
?>
```

Mon nom est Cangimrac Mon prénom est Laurent

Utilisation de l'instruction *global*

```php
<?php
function message($_msg1,$_msg2)
{
   global $sNom,$sPrenom;
   echo $_msg1." ".$sNom." ".$_msg2." ".$sPrenom;
}

$sNom="Cangimrac";
$sPrenom="Laurent";
message("Mon nom est","Mon prénom est")
?>
```

Mon nom est Cangimrac Mon prénom est Laurent

Test d'existence ou de contenu / Destruction

La fonction *isset($var)* : Permet de tester si une variable *$var* existe.
La fonction *unset($var)* : Permet de détruire la variable *$var*.
La fonction *empty($var)* : renvoie vrai si la variable $Var n'existe pas ou si elle contient une chaine vide('') ou 0

Exemple :

```php
$vt = "exemple";
echo isset($vt);   //Renvoie TRUE
unset($vt);
echo isset($vt);   //Renvoie FALSE
```

Variables dynamiques

Un nom de variable peut lui-même être une variable.

```php
$antoine="18";
$joe="20";

$age = "antoine";
echo "L'age de ", $age, " est de ", $$age, " ans.";
```

// affiche : l'age de antoine est de 18 ans

Le nom de la variable peut être calculé.

```php
$prenom1="Antoine";
$prenom2="Joe";

echo ${"prenom"."2"};   //Affiche Joe
```

Variables dynamiques

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

Exemple 1 :

```
$iNbr = 100;          // la variable $var est initialisée à la valeur 100
$adr_iNbr= & $iNbr;   // la variable $adr_var fait référence à $var, elle contient l'adresse de $var
$iNbr ++;             // on change la valeur de $iNbr = 101
echo $adr_iNbr;       // affiche 101
```

Constantes

Identifiant :

Mêmes règles de nommage que pour les variables

Par convention, les identifiants des constantes sont en majuscules, si c'est un mot composé on utilise le _ pour séparer les mots

Déclaration :

define("NOM_CONSTANTE", "Valeur");

Exemple :

```
define("PI", 3.145192);
echo PI;
```

Les bases du langage

TYPES DE DONNÉES

Deux familles

Les variables PHP supportent plusieurs types de données

les types scalaires (types simples)

entier : int, integer
réel : real, float, double
chaîne : string
booléen : bool, boolean

les types composés

tableau : array
objet : object

Les nombres

Nombres entiers

Décimale : entier commençant par un nombre entre 1 et 9.
Octale : entier commençant par un 0 (zéro).
Hexadécimal : entier commençant par 0x (zéro x).

```
$nb = 10;
$nb_negatif = -35;
$nb_octal = 015;  //13 en base 10
$nb_hexa = 0x1A;  //26 en base 10
```

Nombres flottant

```
$nb = 3.14159;  //notation classique
$nb = 5e7;      //notation exponentielle
```

Les chaînes de caractères

On peut les définir :

soit à l'aide d'une paire de ' *(simple quote)*
soit à l'aide d'une paire de " *(double quote)*

Les deux formes ne sont pas équivalentes...

Définition entre simple quote : '.....**'**

peut contenir des " et des retours à la ligne
caractères spéciaux supportés : \' (') et \\ (\)
aucun autre caractère spécial n'est accepté
les variables ne sont pas substituées par leur valeur

Les chaînes de caractères

Définition entre double quote "....."

peut contenir des simples quotes et des caractères spéciaux :

\n → retour à la ligne

\r → retour chariot

\t → tabulation

\\ → caractère \

\$ → caractère $

\" → caractère "

les variables sont expansées *i.e* remplacées par leur valeur

```php
$prenom = "Ruby";
$age = "12";
echo "$prenom a $age ans.";   //Affiche : Ruby a 12 ans.
```

Les chaînes de caractères

Exemple d'utilisation

```php
<?php
    $sNom = 'Laurent';
    echo 'bonjour '.$sNom.'<br>';
    echo 'bonjour $sNom <br>';
    echo "bonjour $sNom <br>";
    echo '<p id="para">boujour '.$sNom.'</p>';
    echo "<p id=\"para\">$sNom </p>";
?>
```

bonjour Laurent
bonjour $sNom
bonjour Laurent

boujour Laurent

Laurent

Les chaînes de caractères

Syntaxe heredoc

le texte est délimité à l'ouverture par **<<< suivi d'un identifiant et à la fermeture** par une ligne avec l'identifiant et un point-virgule.

```
50   $var = "titi";
51   $texte =<<<identifianttexte
52     Ici je peux écrire du
53     texte dur plusieurs lignes
54     avec la syntaxe heredoc.<br>
55     Je peux même faire référence à des varaibles directement
56     sans utiliser de concaténation.<br>
57     Exemple : la variable \$var vaut "$var".
58 identifianttexte;
59
60   echo $texte;
```

Depuis php 5.3.0 l'identifiant peut être entre double quote.

Le délimiteur de fin ne doit pas contenir d'espace, n'y avant, n'y après.

Les chaînes de caractères

Syntaxe heredoc

le texte est délimité à l'ouverture par **<<< suivi d'un identifiant et à la fermeture** par une ligne avec l'identifiant et un point-virgule.

```
50   $var = "titi";
51   $texte =<<<identifianttexte
52     Ici je peux écrire du
53     texte dur plusieurs lignes
54     avec la syntaxe heredoc.<br>
55     Je peux même faire référence à des varaibles directement
56     sans utiliser de concaténation.<br>
57     Exemple : la variable \$var vaut "$var".
58 identifianttexte;
59
60   echo $texte;
```

Ici je peux écrire du texte dur plusieurs lignes avec la syntaxe heredoc.
Je peux même faire référence à des varaibles directement sans utiliser de concaténation.
Exemple : la variable $var vaut "titi".

Les variables sont expansées

Les chaînes de caractères

Syntaxe Nowdoc (depuis php 5.3.0)
Nowdoc est spécifié de manière similaire à Heredoc, mais ***aucune analyse n'est effectuée sur le texte***.

```
63   $var = "titi";
64   $texte =<<<'identifianttexte'
65   Ici je peux écrire du
66   texte dur plusieurs lignes
67   avec la syntaxe Nowdoc.<br>
68   Je peux même faire référence à des varaibles directement
69   sans utiliser de concaténation.<br>
70   Exemple : la variable $var vaut "$var".
71 identifianttexte;
72
73   echo $texte;
```

Les chaînes de caractères

Syntaxe Nowdoc (depuis php 5.3.0)
Nowdoc est spécifié de manière similaire à Heredoc, mais ***aucune analyse n'est effectuée sur le texte***.

```
63   $var = "titi";
64   $texte =<<<'identifianttexte'
65   Ici je peux écrire du
66   texte dur plusieurs lignes
67   avec la syntaxe Nowdoc.<br>
68   Je peux même faire référence à des varaibles directement
69   sans utiliser de concaténation.<br>
70   Exemple : la variable $var vaut "$var".
71 identifianttexte;
72
73   echo $texte;
```

Ici je peux écrire du texte dur plusieurs lignes avec la syntaxe Nowdoc.
Je peux même faire référence à des varaibles directement sans utiliser de concaténation.
Exemple : la variable $var vaut "$var".

Les variables ne seront pas expansées

Connaître le type

Connaître le type d'une variable : *gettype (…);*

```php
echo gettype("chaîne de caractère");  //Affiche : string

$var = 10;
echo gettype($var);    //Affiche : integer

$var=$var==10;
echo gettype($var);    //Affiche : boolean
```

Tester le type (fonctions d'accès rapide) :

Types scalaires :

- is_string(…)
- is_double (…) et is_float(…)
- is_int (…) et is_integer(…)
- is_boolean(…)

Types composés : is_object(…) et is_array(…)

Le transtypage : règles de conversion

Chaîne de caractères ⇨ Nombres

```
echo "3" + 1;    //Affiche 4
echo 1 + "3 petits cochons";    //Affiche 4
echo 1 + "-1.3e3";   //Affiche -1299
echo 1 + "marcel";   //Affiche 1
```

Tous types ⇨ Booléen

false :

- Constante FALSE et NULL ;
- Chaîne vide ou contenant juste un zéro;
- Nombre zéro ;
- Tableau vide ;
- Objet avec aucun champ défini

true : les autres données.

Le transtypage : règles de conversion

Tous types ⇨ Chaîne de caractères

Depuis booléen :

true → "1"

false → "" et ~~non pas le caractère 0~~

Depuis un entier : représentation classique en base dix.

Depuis Tableaux et objets : affichage du type comme valeur *ie* Array ou Object.

Le transtypage : forcer une conversion

En le spécifiant :

```
$var = (integer) "12";   //$var contient un nombre
$var = (string) 12;      //$var contient une chaîne
```

En utilisant la fonction *settype (…, …)* :

```
$var = 12;
settype($var, 'integer');   //$var contient un nombre
settype($var, 'string');    //$var contient une chaîne
```

En utilisant les fonctions dédiées :

Pour les nombres : intval(…), floatval(…), doubleval(…)

Pour les chaîne de caractères : strval(…)

```
$var = intval("12");     //$var contient le nombre 12
$var = strval(12);       //$var contient la chaîne "12"
```

Les bases du langage

LES OPÉRATEURS

Opérateurs d'affectation

Affectation par copie :

```php
$a = 1;
$b = $a;
$a = $a + 1;
echo $a;   //Affiche 2, car on a ajouté 1
echo $b;   //Affiche 1, sa valeur n'a pas été modifiée
```

L'opérateur d'affectation renvoie la valeur affectée : ceci permet des affectations en chaîne.

```php
$a = $b = 3;
$a = ($b = 3);

$b = 3;
echo ($a=$b);
```

Opérateurs d'affectation

Affectation par référence

```php
$a = 1;
$b = & $a;
$a = $a + 1;
echo $a;  //Affiche 2, car on a ajouté 1
echo $b;  //Affiche 2, sa valeur a été modifiée
```

Effacer une référence avec la fonction *unset(…)* :

```php
$a = 1;
$b = & $a;
$a = 2;
echo 'Valeur de $b : ', $b, '<br>'; //Affiche 2

unset($a);
echo 'Valeur de $a : ', $a, '<br>'; //N'affiche plus rien
echo 'Valeur de $b : ', $b, '<br>'; //Affiche 2
```

Opérateurs de bases

Opérateurs arithmétiques

Addition : +
Soustraction : -
Multiplication : *
Division : /
Modulo : %

Opérateur de concaténation de chaînes : . (le point)

Opérateurs combinés : +=, -=, *=, /=, %=, .=, &=, |=, ^=

Opérateurs d'incrémentation : ++, --

Opérateurs de bases

Opérateurs logiques :

ET logique : && , and
OU logique : || , or
NON logique : !
OU exclusif logique : xor

Opérateurs de comparaison :

Egalité : ==, !=
Identité : ===, !==
Relation d'ordre : <, <=, >, >=

Opérateurs de bases

Quelques exemples

```php
$i = 1;
echo $i++;   // Affiche 1
echo $i;     // Affiche 2

$t = array(0 => "Zéro", 1 => "Un");
$i = 0;
echo $t[$i];     // Zéro car $i vaut 0

$s1 = "Hello ";
$s2 = "world";
echo $s1 . $s2;    // Hello world
$s1 .= $s2;        // ou $s1 = $s1 . $s2;
echo  $s1;         // Hello world
```

Priorités entre opérateurs

Priorité	Opérateurs
1	( ) [ ]
2	-- ++ !
3	* / %
4	+ -
5	< <= >= >
6	== != === !==
7	&
8	|
9	&&
10	||
11	Affectation, opérateurs combinés (= += -= etc.)
12	AND
13	OR
14	XOR

Les bases du langage

LES STRUCTURES DE CONTRÔLE

Structures conditionnelles

Quatre structures possibles pour la conditionnelle :

L'instruction *if* :

```
if (condition)
{
  //instructions ;
}
```

L'instruction *else* :

```
if (condition)
{
  //instructions si la condition est vérifiée;
}else {
  //instructions si la condition n'est pas vérifiée;
}
```

Structures conditionnelles

…

L'instruction *elseif* :

```
if (condition1)
  {
  //instructions si la condition1 est vérifiée;
  }elseif (condition2) {
      //instructions si la condition2 est vérifiée;
    }elseif (condition3) {
      //instructions si la condition3 est vérifiée;
      }
else {
  //instructions si les conditions ne sont pas vérifiée;
}
```

⇨ Permet d'enchaîner une série d'instructions *if* sans avoir à les imbriquer.

Structures conditionnelles

…..

Opérateur ternaire comme en C :

```
(condition)?expression si VRAI:expression si FAUX;
```

```
$var4=2;

echo  ($var4>3) ? 'plus grand<br />' : 'plus petit<br />';
```

Remarques:
la condition doit être entre des parenthèses

Le branchement conditionnelles

L'instruction *switch* permet de faire plusieurs tests sur la valeur d'une variable.

```php
$nb = mt_rand(0,3);

switch ($nb)
{
  case 3:
    echo "$nb est superieur à 2 <br>";
    break;
  case 2:
    echo "$nb est superieur à 1 <br>";
    break;
  case 1:
    echo "$nb est superieur à 0 <br>";
    break;
  default:
    echo "$nb est 0 <br>";
}
```

Les boucles

La boucle *tant que* :

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

```
$i = 1;
while ($i <= 10) {
  echo "$i ";
  $i++;
}
```

La boucle *répéter tant que* :

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

```
$i = 1;
do
{
  echo "$i ";
  $i++;
} while ($i <= 10);
```

Les boucles

La boucle *Pour* :

Syntaxe :

```
for (expression1 ; condition ; expression2) {
   //Code à exécuter
}
```

Exemples :

```
for ($i = 2; $i <= 10 ; $i++) {
   echo "$i ";
}
```

```
for ($i = 2, $k = 12; $i <= 10 ; $i++, $k--) {
   echo "$i , $k <br> ";
}
```

Les boucles

L'instruction *break* :

Permet de sortir d'une structure conditionnelle telle que *for*, *while*, *foreach* ou *switch*.

L'instruction *continue* :

Dans une boucle, permet d'éluder les instructions de l'itération courante et de passer directement à la suivante.

Les bases du langage

LES TABLEAUX

Les tableaux

En PHP, les tableaux :

- sont dynamiques *ie* leur taille peut évoluer au cours de l'exécution du script.
- peuvent contenir des éléments de types différents *ie* scalaires ou composés.

Il existe deux types de tableaux :

- Les tableaux indicés : C'est une liste d'éléments repérés par une position unique *i.e.* son indice. En PHP, cet indice commence à zéro.
- Les tableaux associatifs : C'est une liste d'éléments repérés par un identifiant arbitraire unique appelé la clé.

Les tableaux indicés : déclaration

Un nouvel élément peut être défini en spécifiant un indice. Dans le cas contraire, c'est l'entier immédiatement supérieur au plus grand indice du tableau qui est utilisé.

```php
$tab[0] = "Marcel";
$tab[1] = 18;
$tab[2] = "3 rue raoul";
$tab[3] = 13200;
$tab[4] = "Arles";
```

```php
$tab[] = "Marcel";
$tab[] = 18;
$tab[] = "3 rue raoul";
$tab[] = 13200;
$tab[] = "Arles";
```

On peut aussi utiliser l'instruction *array(…)* pour insérer plusieurs éléments.

```php
$tableau = array(1, 2, 3, 4, 5, 6, 7);
$mix = array("Marcel",18,"3 rue raoul",13200,"Arles");
```

Les tableaux indicés : parcours

Parcours du tableau :

```php
$tab = array("Marcel",18,"3 rue raoul",13200,"Arles");
```

En utilisant la fonction *count(…)* qui retourne le nombre d ’éléments du tableau :

```php
$i=0;
while($i < count($tab))
{
  echo $tab[$i].'<br>';
  $i++;
}
```

Marcel
18
3 rue raoul
13200
Arles

En utilisant l’instruction *foreach(…)* :

```php
foreach($tab as $element)
{
  echo $element.'<br>';
}
```

Les tableaux associatifs : déclaration

Dans un tableau associatif, on crée une association entre une clé et un élément.

En PHP, ces tableaux supportent les clés de type chaîne de caractères.

```
$tab['nom'] = "Deuf";
$tab['prenom'] = "John";
$tab['age'] = 12;
```

On peut aussi utiliser l'instruction array(…) en donnant la clé et l'élément, séparés par "=>"

```
$tab = array("nom" => "Deuf", "prenom" => "John","age" => 12);
```

Les tableaux associatifs : parcours

Parcours du tableau :

```php
$tab = array("nom" => "Deuf", "prenom" => "John","age" => 12);
```

En utilisant l'instruction *foreach(…)* :

```php
foreach($tab as $element)
{
   echo $element.'<br>';
}
```

```
Deuf
John
12
```

En utilisant l'instruction *foreach(…)* :

```php
foreach($tab as $key => $value)
{
   echo $key." : ".$value.'<br>';
}
```

```
nom : Deuf
prenom : John
age : 12
```

Les tableaux associatifs : parcours

Parcours du tableau :

```php
$tab = array("nom" => "Deuf", "prenom" => "John","age" => 12);
```

En utilisant list(…) et each(…) :

```php
while( list($key,$value) = each($tab) )
{
  echo $key." : ".$value.'<br>';
}
```

list(…) : Permet d'affecter les éléments du tableau dans des variables distinctes. *list()* n'est pas une véritable fonction, mais un élément de langage.

each(…) : retourne la combinaison clé-valeur courante du tableau passé en paramètre, puis se positionne sur l'élément suivant. Lorsque la fin du tableau est atteinte, elle retourne la valeur false.

Les tableaux multidimensionnels

Les tableaux indicés et associatifs se généralisent aux tableaux multidimensionnels, pour lesquels l'indice, ou la clé, est constituée de plusieurs valeurs.

Un tableau à deux dimensions peut être vu comme un tableau de tableaux *ie* chaque case du tableau est un tableau.

```php
$matrice = array(                    // Definit la matrice :
               array(1, 3, 4),       //   1 3 4
               array(4, 2, 6),       //   4 2 6
               array(8, 0, 3)        //   8 0 3
           );
```

Les tableaux multidimensionnels

Construction :

Parcours :

```php
<?php

$tabDep[13]=array("marseille","arles", "aix", "salon");
$tabDep[77]=array("Melun","Montereau", "Fontainebleau");
$tabDep[10]=array("Troyes","Romilly-Sur-Seine","La Chapelle-Saint-Luc");



foreach ($tabDep as $dep => $tabVille)
  {
   echo '<p><b>'.$dep.'</b><br />';
   foreach( $tabVille as $ville)
      echo '<i>'.$ville.'</i><br />';
   echo '</p>';
  }
?>
```

13
marseille
arles
aix
salon

77
Melun
Montereau
Fontainebleau

10
Troyes
Romilly-Sur-Seine
La Chapelle-Saint-Luc

```
Array
(
    [13] => Array
        (
            [0] => marseille
            [1] => arles
            [2] => aix
            [3] => salon
        )

    [77] => Array
        (
            [0] => Melun
            [1] => Montereau
            [2] => Fontainebleau
        )

    [10] => Array
        (
            [0] => Troyes
            [1] => Romilly-Sur-Seine
            [2] => La Chapelle-Saint-Luc
        )

)
```

Les bases du langage

LES FONCTIONS UTILISATEUR

Déclaration de fonctions

La déclaration d'une fonction peut se faire n'importe où dans le code avec le mot-clé *function* :

```
function nomDeLaFonction($arg1, $arg2, ...) {
  //Liste d'instructions ;
}
```

Une fonction peut avoir aucun ou plusieurs paramètres (arguments) qui peuvent être de type :

Scalaire : integer, double, string…
Ou
composé : tableaux ou objets

On utilisera la convention de nommage lowerCamelCase pour les noms fonctions, *ie* le premier mot en minuscule (lower) pour les autres la premier lettre en majuscule et le reste en minuscule (CamelCase) :

```
afficherMonMessage();
```

Déclaration de fonctions

Valeur par défaut :

il est possible de donner une valeur par défaut aux arguments de la fonction :

```php
function nomDeLaFonction($arg1 = 'valeur_par_defaut') {
  //Liste d'instructions ;
}
```

Exemple :

```php
function sayHello($nom = 'John') {
  echo "Hello $nom !";
}
```

Appel de la fonction :

```
sayHello('laurent') ; // affiche laurent
sayHello() ; // affiche John
```

Déclaration de fonctions

Valeur de retour :

On utilise l'instruction *return* pour faire retourner une valeur à la fonction.

```php
function afficheMessage ($msg) {
  if (empty($msg))
  {
    //Si le message est vide, on retourne faux
    return FALSE;
  }
  else {
    //Sinon, on affiche le texte
    echo $msg;
    //Et on retourne vrai
    return TRUE;
  }
}
```

Appel de fonctions

Pour utiliser une fonction, il suffit d'y faire appel en lui passant les paramètres nécessaires.

Exemple avec les fonctions précédentes :

```php
//Déclaration du nom
$nom = "Brett";
//Appel de la fonction qui dit bonjour
//avec comme paramètre $nom
sayHello($nom);

//Appel de la fonction qui dit bonjour
//sans paramètre ie, utilisation de la
//valeur par défaut
sayHello();

//Appel de la fonction qui affiche un message
if (!afficheMessage("Je vais vous dire bonjour !")) echo "Erreur";
```

Visibilité des variables

Il existe trois niveaux de définition de variable :

local : C'est le niveau par défaut, la variable n'est visible que dans la fonction en cours.

global : La variable est visible dans la fonction et à l'extérieur.

```
global $msg_erreur;      //Déclaration d'une variable globale
$msg_erreur = "Erreur"; //Initialisation de la variable globale
```

static :
La variable n'est connue que par la fonction mais sa valeur persiste durant tout le temps d'exécution du script.

```
static $id = 0; //Déclaration et initialisation de la variable
$id++;          //Persistante ; Et, incrémentation de sa valeur
```

Passage par copie / référence

Par défaut, PHP passe les paramètres par copie pour tout les types, à l'exception des objets (php5).

Pour passer un paramètre par référence, il suffit de le préfixer d'un & dans la définition de la fonction.

```php
//Déclaration de la fonction
function permute(&$arg1, &$arg2) {
  $tmp = $arg1;      // Permutation du contenu
  $arg1 = $arg2;     // des variables
  $arg2 = $tmp;      // $arg1 et $arg2
}

$a = 2;$b = 1;
echo "$a $b"; //Affiche 2 1
permute($a, $b);  //Appel de la fonction de permutation
echo "$a $b"; //Affiche 1 2
```

Nombre de paramètres indéfini

En PHP, il est possible de déclarer des fonctions dont le nombre de paramètre est indéfini.

Pour manipuler les arguments, on utilise alors les fonctions *func_num_args()*, *func_get_arg()* et *func_get_args()*.

```php
//Déclaration d'une fonction d'affichage ayant
//un nombre de paramètres indéfini
function afficheLn() {
  $args = func_get_args();

  foreach($args as $arg) echo "$arg<br>";
}

//Utilisation de la fonction avec un nombre de
//paramètre variable
afficheLn("Marcel","John",18,true);
afficheLn("Hello word !");
```

Retourner plusieurs valeurs

Pour retourner plusieurs valeurs, on utilise un tableau que l'on peut récupérer avec la fonction *list(...)*.

```php
function champ() {
   $tab[] = 'Nom';
   $tab[] = 'Prénom';
   $tab[] = 'Age';

   return $tab;
}

list($nom, $prenom, $age) = champ();
echo "$nom $prenom $age";
```

Les bases du langage

INCLUSION DE FICHIERS

Inclusion de fichiers

PHP offre la possibilité d'inclure des fichiers depuis des scripts ce qui permet par exemple, de se créer des bibliothèques de fonctions, qui pourront être utilisées par plusieurs scripts.

Les instructions permettant de réaliser des inclusions sont *include(…)* et *require(…)*.

Il existe également *include_once(…)* et *require_once(…)* qui ont la même fonction mais s'assurent en plus que le fichier n'a pas déjà été inclus.

Include ou require ?

include (…) **:**

Inclut et exécute un fichier PHP passé en argument.
Si le fichier n'existe pas, la fonction génère une alerte.

require(…) **:**

Inclut le contenu d'un fichier et évalue le fichier PHP contenant la commande.
Si le fichier n'existe pas, la fonction génère une erreur fatale.

Globalement, *require()* et *include()* sont identiques, sauf dans leur façon de gérer les erreurs. Ils produisent tous les deux une Alerte mais *require()* génère une erreur fatale.

Exemple d'inclusion

Exemple avec *include(…)* :

Fichier vars.php :

```php
<?php
$couleur = 'verte';
$fruit = 'pomme';
?>
```

Fichier test.php :

```php
<?php
echo "Une $fruit $couleur"; // Affiche : Une
include 'vars.php';
echo "Une $fruit $couleur"; // Affiche : Une pomme verte
?>
```

Gestion des fichiers

Les fichiers doivent être ouverts, traités puis fermés

Exemples d'ouverture d'un fichier :

```
$fp = fopen("../fichier.txt","r");
        // "r" => ouverture en lecture seul

$fp = fopen("../fichier.txt","w");
        // "w" => ouverture en écriture seul

$fp = fopen("../fichier.txt","a");
        // "a" => ouverture en écriture avec ajout
        // du contenu à la fin du fichier
```

Gestion des fichiers

Les fichiers doivent être ouverts, traités (lecture et/ou écriture) puis fermés

Exemple : lecture d'un fichier

```
31 <?php
32 if ($monfichier=@fopen("data.txt",'r'))
33     {
34      while (!feof($monfichier))
35        {     // 255 caractères max. ou bien fin de ligne.
36         $ligne = fgets($monfichier,255);
37         echo $ligne.'<br />';
38        }
39      fclose($monfichier);
40     }
41 else
42     {die("impossible d'ouvrir le fichier");}
43 ?>
```

Attention : le caractère @ sert à "*échapper*" le message d'erreur php

Warning: fopen(datea.txt): failed to open stream: No su
impossible d'ouvrir le fichier

Gestion des fichiers

Exemple : écriture d'un fichier

```php
<?php
if ($monfichier=@fopen("monfichier.txt", "w"))
    {
     // on écrit deux lignes
     fputs($monfichier,"ligne 1");
     fputs($monfichier,"ligne 2");
     fclose($monfichier);
    }
  else
    {die("impossible d'ouvrir le fichier");}
?>
```

Gestion des fichiers

Quelques fonctions liées aux fichiers :

fopen($file [,$mode]) : ouverture du fichier identifié par son nom $file et dans un mode $mode particulier, retourne un identificateur $fp de fichier ou FALSE si échec
fclose($fp) : ferme le fichier identifié par le $fp
fgets($fp, $length) : lit une ligne de $length caractères au maximum
fgetc($fp) : lit un caractère
fputs($fp, $str) : écrit la chaîne $str dans le fichier identifié par $fp
feof($fp) : teste la fin du fichier
file_exists($file) : indique si le fichier $file existe
filesize($file) : retourne la taille du fichier $file
filetype($file) : retourne le type du fichier $file
unlink($file) : détruit le fichier $file
copy($source, $dest) : copie le fichier $source vers $dest
readfile($file) : affiche le fichier $file
rename($old, $new) : renomme le fichier $old en $new
file ($file) : retourne le fichier $file dans un tableau. Chaque élément du tableau correspond à une ligne du fichier, et les retour-chariots sont placés en fin de ligne.

Accès aux dossiers

Il est possible de parcourir les répertoires grâce à ces quelques fonctions :

chdir($str) : Change le dossier courant en $str. Retourne TRUE si succès, sinon FALSE.

getcwd() : Retourne le nom du dossier courant (en format chaîne de caractères).

opendir($str) : Ouvre le dossier $str, et récupère un pointeur $d si succès, FALSE sinon et génère alors une erreur PHP qui peut être échappée avec @.

closedir($d) : Ferme le pointeur de dossier $d.

readdir($d) : Lit une entrée du dossier identifié par $d. C'est-à-dire retourne un nom de fichier de la liste des fichiers du dossier pointé. Les fichiers ne sont pas triés. Ou bien retourne FALSE s'il n'y a plus de fichier.

rewinddir($d) : Retourne à la première entrée du dossier identifié par $d.

Accès aux dossiers

Exemple de parcours du contenu d'un dossier :

```php
<?php
  if ($dir=@opendir('.'))    // ouverture du dossier courant
      {
       while($file=readdir($dir))    // lecture d'une entrée
         {  echo $file.'<br />'; }  // affichage du nom de fichier

       closedir($dir);       // fermeture du dossier
      }
?>
```

$dir est un pointeur vers la ressource dossier

$file est une chaîne de caractères qui prend pour valeur chacun des noms de fichiers retournés par readdir()

On affiche le nom des fichiers et sous dossiers

Travailler avec le tampon de sortie

ob_clean — **Efface le tampon de sortie**
ob_end_clean — **Détruit les données du tampon de sortie et éteint la temporisation de sortie**
ob_end_flush — **Envoie les données du tampon de sortie et éteint la temporisation de sortie**
ob_flush — **Envoie le tampon de sortie**
ob_get_clean — **Lit le contenu courant du tampon de sortie puis l'efface**
ob_get_contents — **Retourne le contenu du tampon de sortie**
ob_get_flush — **Vide le tampon, le retourne en tant que chaîne et stoppe la temporisation**
ob_start — **Enclenche la temporisation de sortie**

Exemple d'utilisation :

Récupération du contenu d'un fichier.

```php
function get_include_contents($filename) {
    if (is_file($filename)) {
        ob_start();
        include $filename;
        $contents = ob_get_contents();
        ob_end_clean();
        return $contents;
    }
    return false;
}

.
.
.
$string = get_include_contents('somefile.php');
```