

# Introduction à PHP

## Bases du langage

---

`monnerat@u-pec.fr`

19 avril 2020

IUT de Fontainebleau

Introduction

Historique

Le modèle

Le langage

Variables et types

Chaînes de caractères

Tableaux

Constantes

Opérateurs

Structures de contrôle


Fonctions

# Introduction



## PHP

- <http://www.php.net/> 
- <https://www.w3schools.com/php/> 

## MariaDB/SQL

- <https://mariadb.com/kb/en/sql-statements/> 
- Vos cours.

## HTML, CSS, Javascript, XML, etc.

- <https://developer.mozilla.org/fr/> 
- <http://www.w3schools.com> 

# Cours et supports à l'iut

<https://dwarves.iut-fbleau.fr/git/monnerat/wim21>

monnerat/wim21: Dépôt publicitaire

← → ↻ ⓘ <https://dwarves.iut-fbleau.fr/git/monnerat/wim21>

Tableau de bord Tickets Pull Requests Explorer

monnerat / wim21

Ne plus suivre 4 Voter 1 Fork 3

Fichiers Tickets 0 Pull Requests 0 Wiki Paramètres

Dépôt pour les travaux pratiques de WIM 2.1

42 Commits 1 Branches 0 Publications

Branche: master wim21 Nouveau fichier Téléverser un fichier HTTPS SSH [gogs@dwarves.iut-fbleau.fr](mailto:gogs@dwarves.iut-fbleau.fr)

monnerat	f749da8a35	size	il y a 57 minutes
controle	2bfe5931a	ajout du controle	il y a 2 ans
gcontacts	d4ef17db8c	suppression dependances concise.css	il y a 6 jours
securite	d4ef17db8c	suppression dependances concise.css	il y a 6 jours
tp1	f749da8a35	size	il y a 57 minutes
tp2	d4ef17db8c	suppression dependances concise.css	il y a 6 jours
tp2bis	d4ef17db8c	suppression dependances concise.css	il y a 6 jours
tp3	d4ef17db8c	suppression dependances concise.css	il y a 6 jours
tp4	d4ef17db8c	suppression dependances concise.css	il y a 6 jours
tp5	d4ef17db8c	suppression dependances concise.css	il y a 6 jours
README.md	a559f241ce	README	il y a 1 heure

# Introduction

## Historique

Historique

Le modèle

# Un peu d'histoire



La première version été créée 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 (PHP4,5,7) 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.
- Connection à la majorité des SGBD. (MariaDB, MySql, Oracle, postgres, ODBC, etc.)
- Prise en charge de nombreux protocoles et formats : tcp, http, smtp, ldap, imap, pop, ssl, soap, xml, etc.
- Comme il supporte tous les standards du web et qu'il est gratuit, il s'est rapidement répandu sur la toile.

C'est un langage incrusté au HTML et interprété (PHP3) ou compilé (PHP4,5,7) 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.
- Connection à la majorité des SGBD. (MariaDB, MySql, Oracle, postgres, ODBC, etc.)
- Prise en charge de nombreux protocoles et formats : tcp, http, smtp, ldap, imap, pop, ssl, soap, xml, etc.
- Comme il supporte tous les standards du web et qu'il est gratuit, il s'est rapidement répandu sur la toile.

## **Application WEB ?**

Avant de rentrer dans la syntaxe proprement dite du PHP, examinons le contexte !

# Introduction

Le modèle

# Plan : 1 - Introduction

Historique

Le modèle

# Une application WEB ?

Une application Web est une application clients/serveur(s) **sans états**.



On peut la voir en trois couches ...

Front-End

Back-End

SGBD



PostgreSQL



ORACLE®



SGBD



Back-End





# Front-End

```
graph TD; FE[Front-End] --> S[Structure]; FE --> P[Présentation]; FE --> A[Applicatif];
```

Structure

Présentation

Applicatif

# Front-End



Présentation



Applicatif

# Front-End



Applicatif

# Front-End





Client re-  
quête sur  
`http://`  
`www.arda/`  
`bonjour.php`



Client re-  
quête sur  
http://  
www.arda/  
bonjour.php



Serveur



Client re-  
quête sur  
http://  
www.arda/  
bonjour.php

HTTP  
→



Serveur

  
Client re-  
quête sur  
http://  
www.arda/  
bonjour.php

HTTP  
→

  
Serveur

fichier  
→

```
<html>
  <body>
    <?php
      echo "<b>bonjour</b>";
    ?>
  </body>
</html>
```



  
Client re-  
quête sur  
http://  
www.arda/  
bonjour.php

HTTP  
→

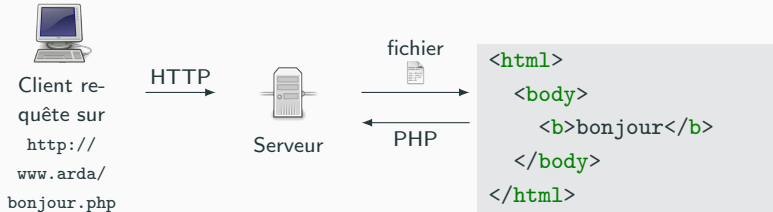


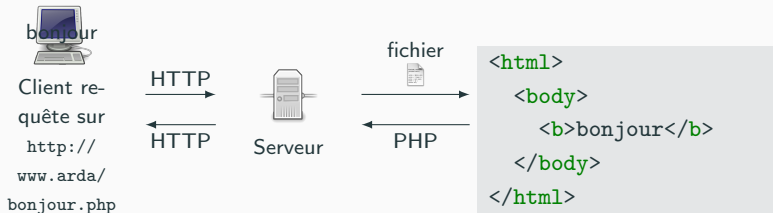
fichier  
→

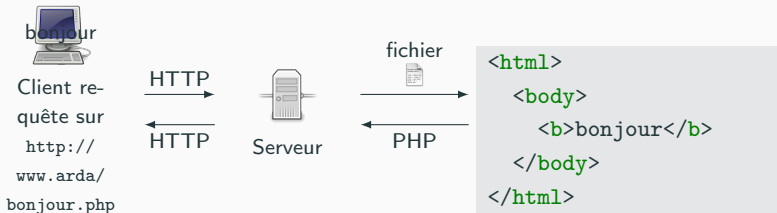
```
<html>
  <body>
    <?php
      echo "<b>bonjour</b>";
    ?>
  </body>
</html>
```



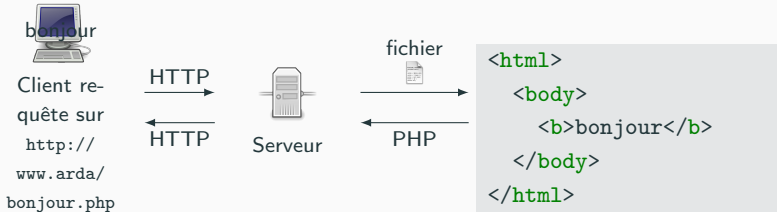
Interprétation





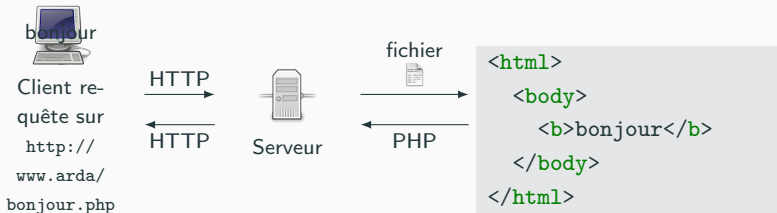


L'intégration de l'interpréteur PHP dans le serveur Web peut se faire de deux manières :



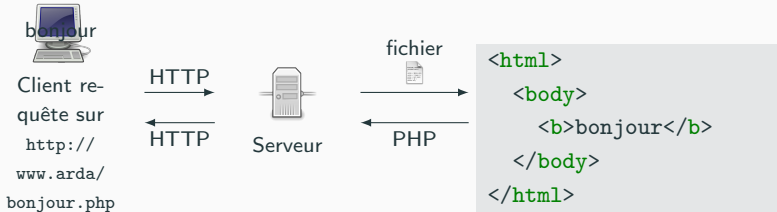
L'intégration de l'interpréteur PHP dans le serveur Web peut se faire de deux manières :

- comme module du serveur : l'interpréteur est directement intégré dans le serveur web. C'est le processus serveur qui interprète le code.



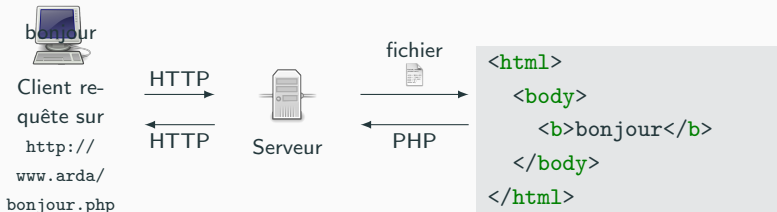
L'intégration de l'interpréteur PHP dans le serveur Web peut se faire de deux manières :

- comme module du serveur : l'interpréteur est directement intégré dans le serveur web. C'est le processus serveur qui interprète le code.
- comme un programme CGI (Common Gateway Interface) : le serveur lance un processus annexe, qui de façon externe interprète le code php. Le serveur récupère alors le résultat.



L'intégration de l'interpréteur PHP dans le serveur Web peut se faire de deux manières :

- comme module du serveur : l'interpréteur est directement intégré dans le serveur web. C'est le processus serveur qui interprète le code.
- comme un programme CGI (Common Gateway Interface) : le serveur lance un processus annexe, qui de façon externe interprète le code php. Le serveur récupère alors le résultat.
- CGI est défini dans la rfc 3875.



L'intégration de l'interpréteur PHP dans le serveur Web peut se faire de deux manières :

- comme module du serveur : l'interpréteur est directement intégré dans le serveur web. C'est le processus serveur qui interprète le code.
- comme un programme CGI (Common Gateway Interface) : le serveur lance un processus annexe, qui de façon externe interprète le code php. Le serveur récupère alors le résultat.
- CGI est défini dans la rfc 3875.
- L'exemple précédent est **crétin**....



# Adressage des documents

## Solution

- **URL** : *Uniform Resource Locator* = adressage universelle de ressources.
- 3 parties : le protocole (comment), le nom (où) et le nom du document (quoi).
- **URL**  $\subset$  **URI** *Universal Resource Identifier*.

## Exemples

- `http://www.iut-fbleau.fr/sitebp`
- `https://www.google.fr:8888/img/plan.php?x=12&y=20`
- `ftp://user:password@www.iut-fbleau.fr/account/`

# Composition d'une URL

protocole://hostname:port/path/extrapath?arguments

- La racine / de path est définie par la configuration du serveur Web. (rien à voir a priori avec la **racine du système de fichier**)
- path peut contenir un point d'ancrage.
- extrapath et arguments permettent de passer des informations à un programme qui s'exécute sur le serveur (script php par exemple).

- Client (souvent, mais pas que) : un navigateur qui fait des requêtes sur un serveur qui interprète le php.

- Client (souvent, mais pas que) : un navigateur qui fait des requêtes sur un serveur qui interprète le php.

- Serveur(s) :

---

HTTP(S)

pour les pages HTML et PHP : Apache, nginx, hiaawatha

---

SGBD

serveur sql : mysql, oracle, Postgres

---

LDAP

OpenLdap (authentification )

---

On peut aussi utiliser php indépendamment d'un serveur web, comme un langage de script à part entière :

Depuis le bash :

```
$php -f helloworld.php
```

Avec un exécutable :

```
#!/usr/bin/php
<?php
echo "hello world !!";
?>
```

la commande php embarque un **serveur web interne** (uniquement pour tester) :

```
php -S localhost:8080 -t directory
```

Un serveur http(s) (Apache) sur la machine `dwarves.arda`  
(`dwarves.iut-fbleau.fr`)

qui sert le répertoire `public_html` à votre racine, via les urls :

```
http[s]://dwarves.iut-fbleau.fr/~votre_login
```

dans lequel se trouve un fichier `.htaccess` qui demande une authentification via ldap.

Des serveurs de bases de données :

- MariaDB sur `dwarves`,
- Oracle sur `lorien`.

# Le langage

# Intégration de php à html

Les pages web sont au format html/xhtml. Les pages web dynamiques sont au format php sur le serveur. Le code source php est directement insérer dans le flux html grâce au conteneur de la norme XML : `<?php ... ?>`

```
<!DOCTYPE html>
<html>
  <head>
    <title>hello world en php !</title>
  </head>
  <body>
    <?php
      echo "<b>hello world !<b>"; /* original !! */
    ?>
  </body>
</html>
```



Autre syntaxe d'intégration :

- `<? ... ?>`
- `<script language="php"> ... </script>`
- `<% ... %>`

# Le langage

## Variables et types

# Variables et types

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

Type	Description	Exemple
<b>NULL</b>	une seule valeur	NULL
<b>boolean</b>	booléen	True/False
<b>integer</b>	entier	1234,056,0xaf23c
<b>double</b>	réel	1.234,1.2e3
<b>string</b>	chaîne de caractères	"toto"
<b>array</b>	tableau	[1234,"denis"]
<b>object</b>	objet	

Les identificateurs de variable sont précédés du symbole \$ (dollars) en lecture **et** en écriture. Exemple : \$toto.

C'est le "contexte" qui détermine le type d'une variable ou d'une expression.

```
$a;           // a est de type NULL
$a=1;         // a est un entier
$a=1.2;       // a est un double
$a="toto";    // a est une chaîne
```

On peut caster comme en C.

Les fonctions

- `var_dump` affiche les informations structurées d'une variable.
- `print_r` affiche des informations lisibles pour une variable.

Les nombres entiers, comme en C, peuvent être représentés en décimal (`$a=153`), en octal (`$a=015`) ou en hexadécimal (`$a=0xa3`).

```
$a=array(1.5,"aaaa",true,10=>0x14);  
print_r($a);  
var_dump($a);
```

var\_dump

```
array(4) {  
    [0] =>  
        float(1.5)  
    [1] =>  
        string(4) "aaaa"  
    [2] =>  
        bool(true)  
    [10] =>  
        int(20)  
}
```

print\_r

```
Array  
(  
    [0] => 1.5  
    [1] => aaaa  
    [2] => 1  
    [10] => 20  
)
```

# Variables et types

Fonction	Description
<code>empty(\$var)</code>	renvoie vrai si la variable est vide
<code>isset(\$var)</code>	renvoie vrai si la variable existe
<code>unset(\$var)</code>	détruit une variable
<code>gettype(\$var)</code>	retourne le type
<code>settype(\$var, "type")</code>	convertit la variable (cast)

On a aussi les fonctions (d'interrogations) : `is_long()`, `is_double()`, `is_string()`, `is_array()`, `is_object`, etc.

Chaîne comme identificateur `${$var}=valeur`

```
$toto="annee";  
${$toto}=2007;  
echo $annee; // $annee vaut 2007
```

Pour la majorité des variables, la portée concerne la totalité d'un script PHP. Mais

- 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.
- Il est possible néanmoins d'avoir recours à des variables globales (cf fonctions)

Les variables ont une **vie temporaire**. Une fois "la page affichée" (la réponse calculée), elles cessent d'exister. (pas tout à fait vrai avec la notion de variables de sessions)

# Le langage

Chaînes de caractères



# Chaînes de caractères

Elles sont généralement délimitées par des guillemets.

```
<?php  
echo "Super le <b>php</b> !!!";  
?>
```

Comme en Perl (ou Shell), ce qui est précédé par \$ est expansé ; c'est à dire remplacé par sa valeur.

```
<?php  
$a=10;$b=20;$c=$a+$b;  
echo "la somme de $a et de $b vaut $c";  
?>
```

Comme en C, on peut protéger certains caractères avec le caractère d'échappement \.

```
<?php
echo "je mets des \$var et des guillemets \"";
echo "et des antislashes \\";
?>
```

On peut protéger l'ensemble de la chaîne en utilisant des quotes (il faut alors protéger la quote elle-même si besoin).

```
<?php
echo 'je mets ce que je veux : \ \n $var';
?>
```

Afficher une chaîne :

- echo
- printf : même syntaxe qu'en C

```
printf('%03d %s %03.4f',1,'bonjour',1.0);
```

```
$texte='Super le PHP';  
echo $texte{2};  
echo $texte[0]; // depuis php5  
echo ord('A');  
echo chr(68);
```

# Fonctions sur les chaînes

Il en existe beaucoup. Se référer à la documentation du langage.

## Protection et échappement (Injection)

⇒ SQL :

- La fonction `addslashes` permet de protéger automatiquement les guillemets, quote et antislashes.
- La fonction `stripslashes` fait l'opération inverse.

Il existe cependant de meilleures alternatives suivant la base et l'api utilisée :

- `mysqli_escape_string` pour mysql,
- `pg_escape_string` pour postgres, etc.

⇒ HTML : Les fonctions `htmlentities` et `htmlspecialchars` permettent de convertir les caractères spéciaux en entités HTML.

# Le langage

## Tableaux

# Les tableaux

Deux types d'indexages : numérique ou alphabétique.

```
$tableau=array(10,12,13,$variable,10=>"bonjour",1.2);  
echo $tableau[1];
```

Quel est l'indice de 1.2 ?

```
$individu=array(  
    "prenom"=>"Michel",  
    "ville"=>"Fontainebleau",  
    "code postale"=>"77100");  
echo $individu['prenom'];
```

On peut mixer les deux.

PHP dispose d'instructions pour parcourir les éléments d'un tableau.

```
/* seulement les valeurs */
foreach ($arr as $value) {
    echo "Valeur : $value<br />\n";
}

/* valeurs et clefs */
foreach ($arr as $key => $value) {
    echo "Clef : $key Valeur : $value<br />\n";
}

/* modification des valeurs */
foreach ($arr as &$amp;value) {
    $value=$value+2;
}
```

Il existe de nombreuses fonctions dédiées à la gestion des tableaux.

# Le langage

## Constantes



# Constantes

La fonction `define` permet de définir des chaînes constantes.

```
define("chaineconstante",valeurconstante)
```

```
define ("PI", 3.14);  
define ("JAUNE", "#FFFF00");  
define ("UPEC", "Université Paris Est Créteil");  
//exemple d'utilisation  
echo ("Le perimetre du cercle est :" . 2*PI*$rayon);  
echo ("<Body Bgcolor=" . JAUNE . ">");
```

- Les constantes ne sont préfixées par \$.
- Il est fréquent de construire des fichiers qui ne contiennent que des constantes pour gérer des paramètres de configuration ou de traduction de manière centralisée.

**Le langage**

**Opérateurs**

# Opérateurs arithmétiques

Ce sont les mêmes qu'en langage C.

<code>\$a + \$b</code>	addition
<code>\$a - \$b</code>	soustraction
<code>\$a * \$b</code>	produit
<code>\$a / \$b</code>	division
<code>\$a % \$b</code>	division
<code>\$a ++ , ++\$a</code>	post,pre incrémentation
<code>\$a-- , --\$a</code>	décrémentation

```
echo '<table>';
for ($i=0;$i<10;$i++){
    $class = ($i % 2) ? "impair" : "pair" ;
    echo "<tr class='$class'><td>$i</td></tr>";
}
echo '</table>';
```

# Opérateurs logiques et binaires

## Opérateurs booléens

<code>  </code> ou <code>OR</code>	OU logique
<code>&amp;&amp;</code> ou <code>AND</code>	ET logique
<code>XOR</code>	OU exclusif

## Opérateurs bit à bit

<code> </code>	OU bit à bit
<code>&amp;</code>	ET bit à bit
<code>^</code>	OU exclusif bit à bit
<code>~</code>	Négation bit à bit
<code>&lt;&lt;, &gt;&gt;</code>	Décalages

# Concaténation

Il s'agit de l'opérateur `.`. Il permet de concaténer deux chaînes de caractères.

```
<?php
$a="Hello";
$b="world !!";
echo $a.' '.$b;

$table="etudiant";
$login="toto";
$req="select * from ".$table." where login='$login'";

echo "Nous sommes le ".date('G \h i').".";
?>
```

# Relation de comparaisons

==	relation d'égalité (après transtypage=)
<,<=	infériorité stricte, infériorité
>,>=	supériorité stricte, supériorité
!=	différence
===	identité : mêmes valeurs et mêmes types
!==	valeurs différentes, ou types différents

Depuis PHP7, on a un nouvel opérateur <=> (space ship)

```
<?php
// Entiers
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1
?>
```

# Opérateurs d'assignation

L'opérateur d'affectation est "=".

```
$s="bonjour";  
$a = ($b = 4) + 5;  
// $a est maintenant égal à 9, et $b vaut 4.
```

Tout comme en langage C, si  $op \in \{+, -, *, /, \&, \%, |, .\}$  :

```
$a = 3;  
$a += 5;  
$b = "Bonjour ";  
$b .= " tout le monde!";
```

$\$a \ op = \$b$  équivaut à  $\$a = \$a \ op \ \$b$

# Le langage

Structures de contrôle



# Instructions

Les instructions d'un script php se termine (généralement) par un ;.

```
<?php  
    echo 'Ceci est un test';  
?>
```

De plus, plusieurs instructions peuvent être regroupées en bloc, délimité par des accolades ("{}"). Un bloc est considéré comme une instruction.

```
{  
    $b = "Bonjour ";  
    $b .= " tout le monde!";  
}
```

# Conditionnelles

## Simple

```
if ($animal == "chien"){  
    echo "ouaf ouaf !";  
}
```

## Double

```
if ($a%2 == 0) echo "pair !";  
else echo "impair !";
```

## Multiple

```
$nombre=mt_rand(0,4);  
switch($nombre){  
case 0,2,4 : echo "nombre pair";break;  
default    : echo "nombre impair";  
}
```

Comme en langage C

- `while` et `do while`
- `for`
- On peut sortir inconditionnellement avec `break`, et passer à l'itération suivante avec `continue`

Comme en Perl

- `foreach` permet de parcourir un à un les éléments d'un tableau.
- Deux syntaxes possibles :
- `foreach($array as $element) instructions`
- `foreach($array as $key=>$element) instructions`

# Syntaxe alternative pour les blocs

PHP propose une autre manière de rassembler des instructions à l'intérieur d'un bloc pour les structures : if, while, for, foreach et switch

```
<?php if ($a == 5): ?>  
<p>a égale 5</p>  
<?php endif; ?>
```

```
<ul>  
  <?php foreach($this->list as $contact) : ?>  
    <li><?php echo $contact->nom; ?></li>  
  <?php endforeach; ?>  
</ul>
```

# Le langage

## Fonctions

# Fonctions

```
<?php
function pgcd($a,$b=0){
    while($b!=0){
        $r=$a%$b;
        $a=$b;
        $b=$r;
    }
    return $a;
}
echo "le pgcd de 1520 et 448 est  ".pgcd(1520,448). "<br/>";
?>
```

Le passage par défaut se fait par valeur mais on peut déclarer un paramètre comme devant être passé par référence

```
<?php
function swap(&$a,&$b)
{
    $aux=$a;
    $a=$b;
    $b=$aux;
}
$i=2;
$j=3;
swap($i,$j);
echo "i=$i et j=$j";
?>
```

# Variable globale

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 :

```
<?php
function change() {
    global $var;           // définit $var comme globale
    $GLOBALS["toto"] ++;   // incremente la variable globale $toto
    $var++;                 // cela sera repercute dans
}                           // le reste du programme
?>
```



# Variable statique

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

```
<?php
function change() {
    static $var=0; // definit $var comme statique
    $var++;        // sa valeur sera conservee
                  // jusqu'au prochain appel
?>
```

# Réutilisation de code

Il est pratique et propre d'écrire des fonctions dans un fichier à part (bibliothèque) de manière à les réutiliser suivant les besoins [ fonctions de connexion à une base, authentification, etc.]

- `include` et `include_once` : inclut et évalue le code dans un script.

```
<?php  
include_once("mesfonctions.php");  
?>
```

- `require` et `require_once` : idem
- Lequel utiliser : `include` ou `require` ? : il n'y a plus de différence depuis PHP 4.0.2. en cas d'erreur, `include` génère seulement un warning, tandis que `require` génère une erreur fatale.

On peut inclure aussi du code html par ce biais.

- Les fichiers sont inclus suivant le chemin du fichier fourni ; si aucun n'est fourni, l'`include_path` sera vérifié avant.
- Il est fréquent d'inclure un fichier par rapport au fichier en cours :

```
<?php  
include_once dirname(__FILE__) . '/menu.php';  
?>
```

# Espace de noms

Pour éviter les collisions et regrouper des fonctions et constantes (entre autres), on peut utiliser des espaces de noms :

```
namespace cercle{

const PI=3.14;

function perimetre($r){
    return 2*PI*$r;
}
function surface($r){
    return PI*$r*$r;
}
}
```

```
include 'cercle.php';  
use function \cercle\surface as aireDisque;  
$r = 5;  
$p = \cercle\perimetre($r);  
$s = aireDisque($r);  
  
echo "perimetre = ".$p;  
echo "surface = ".$s;
```