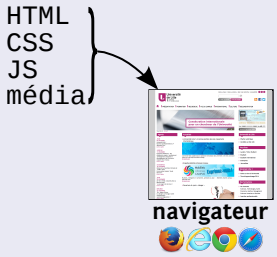
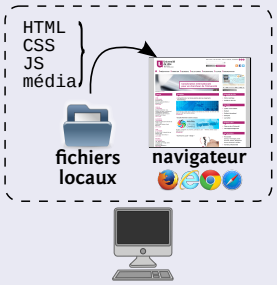
Composition



Une page web est composée à partir de fichiers

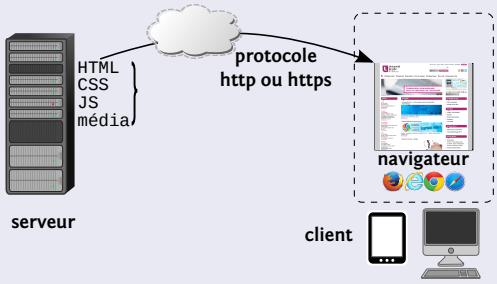
- **(X)HTML** : pour le contenu
- **CSS** : pour la mise en page
- **JS** : pour la dynamicité du document
- **images** et autres **media**

Sources locales



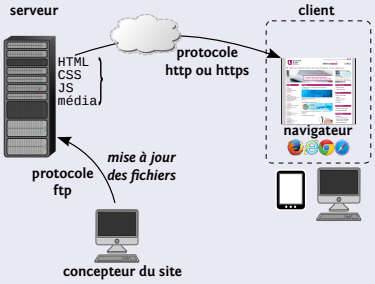
Travailler avec des fichiers locaux peut être utile pour effectuer des tests, mais n'est pas le modèle pour le déploiement de sites.

Modèle client - serveur



- les fichiers sont situés sur un serveur
- le client y accède par le réseau. Il utilise le protocole HTTP ou HTTPS (sécurisé, crypté)

Modèle client - serveur



- Le développeur dispose d'un accès spécifique pour déposer les fichiers : protocole FTP, accès authentifié (login, mot de passe)

Générer des pages côté serveur

Statique vs dynamique

Une page écrite directement en (X)HTML est **statique** : elle n'évolue que quand son auteur la réédite.
On a besoin de pages dont le contenu évolue en fonction

- du temps
- des données disponibles sur le serveur
- de ce que souhaite consulter le visiteur
- ...

Programmation

La page est engendrée par un **programme** qu s'exécute sur le serveur **à chaque fois** qu'un visiteur demande la page.

→ **Le serveur ne contient pas la page HTML mais le programme qui la fabrique.**

Générer des pages côté serveur

Quelques langages côté serveur

- **PHP** : langage de script créé pour cet usage.
- **Java** : technologies JSP ou Servlet
- **javascript** : plateforme **Node.js** notamment
- **ASP** : langage de script (microsoft)
- **C#, VB.net** : technologie microsoft ASP.net
- ...

PHP : un langage de programmation.

- Un langage **impératif**
- Un langage **à typage dynamique** (attention !)
- Sa syntaxe **ressemble un peu** à celle de C/C++/Java/Javascript.
- Un langage avec un aspect **« objet »**.

Des changements importants existent entre les différentes versions de PHP

Ce cours est basé sur PHP 5.0

PHP, c'est

- Le langage de programmation lui-même.
- Des bibliothèques (API).

PHP et le Web

Imbrication

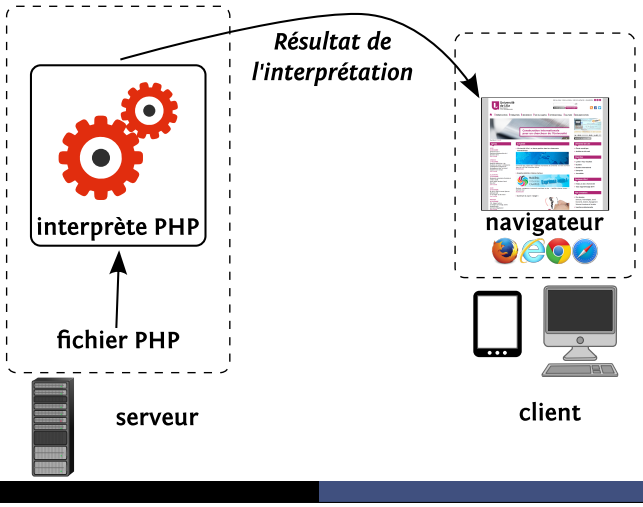
- Les scripts PHP sont **imbriqués** dans les fichiers de syntaxe HTML dans des blocs `<?php ... code PHP ... ?>`
- Chaque bloc PHP est **interprété par le serveur**.
- Il est remplacé par le résultat de son exécution (le texte qu'il produit sur la « sortie standard »).

```
<body><p>Heure :
<?php
date_default_timezone_set
            ('Europe/Paris');
echo date('H:i:s');
?>
</p></body>
```

⇒

```
<body><p>
Heure:
13:03:57
</p></body>
```


PHP et le Web



PHP : Flux de sortie standard

- `echo $args, ...` : une ou plusieurs chaînes de caractères
- `print($arg)` : une chaîne de caractères
- `printf($format, $args...)` : chaîne de caractères formatée

```
<?php
$n = 1234;
echo "le carre de ", $n, " est ", $n*$n ;
echo "\n"; // newline
printf("le carre de %d est %d",$n, $n*$n);
?>
```

⇒

```
le carre de 1234 est 1522756
le carre de 1234 est 1522756
```


Syntaxe PHP : instructions, blocs, commentaires,...

Syntaxe « à la C »

- délimitation de blocs par accolades { }
- affectations : = += -= .=
- *if (cond) instruction else instruction ;*
- *while (cond) instruction ;*
- *for (init ; test ; iter) instruction ;*
- == (égalité simple) ou === (égalité des valeurs et des types)
- commentaires : `/* .. */` ou `//`

Syntaxe PHP : Identificateurs

Distinguer 2 cas

- **Les noms de variables**
  - Commencent par un $
  - **Distinction entre majuscule et minuscule**
  - Exemple : `$Var1` n'est pas équivalent à `$var1`
- **Les noms de fonctions, constantes, méthodes**
  - PAS DE $ initial
  - **PAS DE distinction entre majuscule et minuscule**
  - Exemple : `maFonction()` équivaut à `mafonction()`

Les constantes

- définies par la commande **define**. Exemple : `define(PI_APPROX, 3.14);`
- l'usage est d'écrire les identificateurs de constantes en majuscules.

Variables.

- Il n'y a **pas de déclaration** : les variables sont créées à leur première apparition.
- Il n'y a **pas de typage explicite** : le type d'une variable est déterminé par la valeur qu'elle contient.
- Une même variable peut tantôt contenir un entier, tantôt une chaîne, etc...
- Mais les **valeurs** ont un type : booléen, entier, flottant, chaîne, objet ... ...

Les types simples

Types scalaires

- **Booléens** : boolean.
  les constantes sont notées TRUE et FALSE.
- **Entiers** appelés int ou integer.
  Les constantes entières sont
  - en décimal si elles commencent par un chiffre entre 0 et 9
  - en octal si elles commencent par 0 suivi d'un chiffre
  - en hexadécimal si elles commencent par 0x.
- **Flottants** appelés double ou float.
- **Chaînes** : string

Les nombres

Conversion automatique : attention pièges !

conversion automatique « entier » → « flottant » si le résultat d'une opération arithmétique n'est pas un entier.

- La division de 2 entiers **peut être** un flottant ! 7/2 vaut 3.5 (alors que 6/3 est un entier).
- Les entiers trop grands ou trop petits sont automatiquement convertis en flottants. Les entiers sont généralement codés sur 32 bits. L'intervalle des entiers est $[-2^{31}, 2^{31} - 1]$.

Conversion flottant vers entier : conversion forcée (**cast**)

La conversion explicite d'un flottant en entier est une conversion "vers 0".

- (int) 2.7 vaut 2 (int) 2.1 vaut 2
- (int) -2.7 vaut -2

Les chaînes

Délimitées par simples quotes

- Seulement 2 séquences « spéciales » : \' qui signifie '
  \\ qui signifie \
- Tous les autres caractères sont traités normalement.

Délimitées par doubles quotes

- Les séquences spéciales reconnues :

\"	"
\\	\
\n	newline
\t	tabulation
\$	$
$	début de variable
nb octal ou hexa	caractère de ce code ascii

- **Les noms de variables sont remplacés par la valeur**

Chaînes et noms de variables

Uniquement pour les chaînées délimitées par "

- Le $ indique un début de nom de variable.
- La chaîne contiendra la **valeur** de cette variable au lieu de son nom.
- Ex : après
  `$i = 355;  $s = "La valeur de \$i est $i";`
  le contenu de $s est `La valeur de $i est 355`
- Le nom de variable peut être entouré d'accolades
- Ex : après
  `$i = 355;  $s = "La valeur de \$i est {$i}";`
  le contenu de $s est `La valeur de $i est 355`
- Accolades indispensables pour les éléments de tableaux ou attributs d'objet
- Ex : après `$i = 2; $t[2] = 666;`
  `$s = "dans la case \$t[{$i}] : {$t[$i]}";`

Les chaînes

Concaténation

- **L'opérateur est le point**
- `"LM" . "D" === "LMD"`
- Affectation `.=` → équivalent du `+=` pour les nombres
- `$s = "hello"; $s .= " world"; //$s == "hello world"`

Un seul type de chaînes

Plusieurs façons de délimiter les chaînes, mais un seul type !

- `"abcd" === 'abcd'`
- `'href="machin.fr"'  ===  "href=\"machin.fr\""`
- si `$x == 12`, alors `"\$x=$x" === '$x=12'`

Opérateurs numériques

opérateurs : + - * / %

Type du résultat :

- Si l'une des opérandes est un flottant, les résultat est un flottant.
- Si le résultat ne peut être représenté par un entier, il est flottant (certaines divisions, débordement de capacité des entiers).

Exemple

```
$i = 2 * 3  ; // entier
$i = 2 * 3.0; // flottant
$i = 2 / 3;   // flottant
$i = 6 / 3;   // entier
$i = 6.0 / 3; // flottant
```


Opérateurs de comparaison/ Opérateurs logiques

Opérateurs de comparaison (résultat booléen).

- Égal : ==
- Différent : != ou <>
- Relations d'ordre : <=, <, >=, >
- Identité : === (même valeur et même type).

Opérateurs logiques (opérandes et résultat booléens)

- et : && and
- ou : || or
- non : !
- ou exclusif : xor

Attention : && et and ont des priorités différentes (de même que || et or). Ne pas mélanger les 2 notations sans parenthéser. Il est, de toutes manières, conseillé de parenthéser.

Conversion avant opération

Si une opérande n'est pas du type attendu par un opérateur, **une conversion de la valeur est d'abord réalisée vers le type attendu**. Par exemple

- pour un opérateur arithmétique, les 2 opérandes sont d'abord converties en nombre
- pour la concaténation, les 2 opérandes sont d'abord converties en chaîne.
- pour un opérateur logique, les 2 opérandes sont d'abord converties en booléen.

Exemple

```
$i = TRUE + 3;  // ???
```

L'opérateur + attend deux opérandes numériques. Pour première opérande on prend en compte la **conversion** de true en numérique et on réalise l'addition avec la valeur obtenue

Conversion avant opération(2)

Autre exemple

- ```
  $res = 12;
  $s = "Résultat : " . $res;
  ```
- La concaténation (.) attend 2 chaînes. On calcule tout d'abord la conversion de l'entier 12 en chaîne
- le résultat est utilisé ensuite pour la concaténation.

Conversion explicite (cast)

Mettre le nom du type entre parenthèses. Exemple :

```
$res = 12;
$s = "Résultat : " . (string) $res;
// équivalent à l'expression précédente.
// autre exemple :
$s = "Résultat : " . (string) TRUE;
```


Règles de conversion

Booléen	→ Numérique	→ Chaîne
TRUE	→ 1	→ "1"
FALSE	→ 0.	→ ""
Numérique	**→ Booléen**	**→ Chaîne**
x	→ x != 0	→ représentation décimale de x.
Chaîne	**→ Booléen**	**→ Numérique**
s	→ s !="" && s !="0"	→ *n* : si s commence par la représentation du nombre *n*.
	→	→ 0 : sinon.
Flottant	**→ Entier**	
x	→ arrondi "vers 0" de x	

Dangers !

La conversion est donc possible de tout type scalaire dans tout autre.
En plus, elle est effectuée **automatiquement** (si nécessaire) avant les opérations. Cela conduit à des résultats parfois déroutants.

```
$i= 3;
echo $i + "  10 petits cochons"; // ???
// exemple classique
```

Archétype des situations dangereuses auxquelles on peut être confronté (en codant par exemple + au lieu de .).
Le programmeur n'est pas prémuni contre ses petites erreurs de programmation.
```

Les tables

- Conception différente des tableaux de Java, C ou Pascal
- Ce sont plutôt des listes ou des **tables de hachage** que des tableaux à adressage direct.
- Particularités des tables PHP :
  - Leur **longueur varie dynamiquement**.
  - On peut indexer par des entiers naturels **non consécutifs**.
  - On peut **indéxer par des chaînes**.

Mapping

- Les tables PHP permettent de réaliser des associations **clé - valeur**.
- À une clé correspond **au plus** une valeur (**mapping**).
- Une clé peut être un **entier naturel** ou une **chaîne de caractères**.
- Une valeur peut être d'un type quelconque.

Tables : syntaxe

« agrégat »

`array(`***clé*** `=>` ***valeur,*** ***clé*** `=>` ***valeur, ...***`)`

- Exemple :

```
$fibo = array(0=>1, 1=>1, 2=>2, 3=>3,
              4=>5, 5=>8, 6=>13);
// équivaut à
$fibo = array(1, 1, 2, 3, 5, 8, 13);
```

- Si la clé est omise, la plus petite clé entière non encore utilisée dans la table sera utilisée.
- Autre exemple (indices non consécutifs) :

```
$racineExacte = array(0=>0, 1=>1, 4=>2,
                      9=>3, 16=>4, 25=>5);
```


Tables : syntaxe

Notation [ ]

nomDeLaTable`[` *clé* `]`

- Par exemple :

```
for ($index=0; $index<=6; $index++)
   echo $fibo[$index];
```

ou encore

```
$racineExacte[36]=6;
$racineExacte[49]=7;
```

- Crochets vides. Ex : `$t[]= 100`
  Le premier indice numérique libre est utilisé.

Nombre d'éléments : fonction **count(table)**

Ex : `count($racineExacte)` vaut 8.

Absence de valeur. Suppression

Que se passe-t-il si on invoque un élément de la table qui n'existe pas ?

- Exemple :

```
echo $racineExacte[7];
```

- Une valeur particulière appelée `NULL` est retournée. NULL désigne « l'absence de valeur ».

destruction : fonction **unset**

Toute variable ou tout élément de table peut être supprimé par

`unset(` *nomDeVariable* `)`

Exemple : `unset($fibo[6]);`
Les autres associations (clé,valeur) ne sont pas perturbées.

Parcours du tableau.

Plusieurs manières

- Classique, par index croissant ... quand c'est possible !

```
for ($index=0; $index<=5; $index++)
   echo $fibo[$index];
```

- Par boucle **foreach**
  - `foreach (`***nomDeLaTable*** `as` ***varClé*** `=>` ***varValeur*** `)`
    à chaque tour de boucle
    - ***varClé*** **désigne l'une des clés**
    - ***varValeur*** **l'élément correspondant.**
  - `foreach (`*nomDeLaTable* `as` *varValeur* `)`
    à chaque tour de boucle *varValeur* désigne la **valeur** de l'un des éléments.

foreach

Exemple

```
foreach ($racineEntiere as $racine)
 echo $racine;
 // imprime : 0 1 2 3 4 5 6 7

 // Variante :
 //  récupération des clés et de valeurs:

foreach ($racineEntiere as $carre => $racine)
  echo "la racine de {$carre} est {$racine} ";
 // affiche: ... la racine de 4 est 2 ...etc...
```

foreach : plus de détails

- Dans un boucle foreach, **toute modification de la variable de boucle est sans effet sur la table elle-même**.
- Exemple

```
foreach ($fibo as $nombreFibo)
 $nombreFibo=0; //inutile. À éviter.
// ne modifie pas le tableau $fibo.
```

- Pour modifier le tableau, il faut y accéder par la notation [ ] habituelle.
- Toutefois cette modification est sans effet sur le déroulement de la boucle en cours.
-

```
foreach ($fibo as $indice=>$nombreFibo)
  $fibo[$indice]=0;
 // modifie le tableau $fibo.
```


Ordre de rangement et de parcours des tables

- Les tables possèdent un **ordre de rangement** des clés.
- Cet ordre de rangement **n'est pas** nécessairement l'ordre croissant des clés, mais l'**ordre dans lequel elles ont été définies**.
- **La boucle foreach parcourt la table dans son ordre de rangement.**
-

```
$carres = array(1=>1, 5=>25, 4=>16, 3=>9);
$carres[2]=4; $carres[0]=0;
foreach ($carres as $elt=>$carre)
  echo "({$elt},{$carre})";
// affiche (1,1)(5,25)(4,16)(3,9)(2,4)(0,0)
```


Tris des tables

préservent les associations (clés,valeurs)
ne modifient QUE l'ordre de rangement

- **asort**(table) : tri par ordre croissant des valeurs.
- **ksort**(table) : tri par ordre croissant des clés.
- **arsort**(table) : tri par ordre décroissant des valeurs.
- **krsort**(table) : tri par ordre décroissant des clés.

modifient les clés (donc les associations)

Le tableau est **réindexé par des entiers successifs**. Méthodes **inadaptées aux tableaux associatifs**, en général.

- **sort**(table) : tri par ordre croissant des valeurs, avec réindexation à partir de 0.
- **rsort**(table) : tri par ordre décroissant des valeurs, avec réindexation à partir de 0.

(retenir : r = "reverse").

Exemples de tris

```
$longueurMois["janvier"]=31;
$longueurMois["février"]=28;
$longueurMois["mars"]=31;
$longueurMois["avril"]=30;
foreach($longueurMois as $nom=>$lg)
    echo "({$nom}:{$lg})";
 //(janvier,31)(février,28)(mars,31)(avril,30)
ksort($longueurMois);
foreach($longueurMois as $nom=>$lg)
    echo "({$nom}:{$lg})";
 //(avril:30)(février:28)(janvier:31)(mars:31)
asort($longueurMois);
foreach($longueurMois as $nom=>$lg)
    echo "({$nom}:{$lg})";
 //(février:28)(avril:30)(mars:31)(janvier:31)
```


Tables utilisés comme éléments.

Elles permettent de réaliser des structures "à plusieurs dimensions".

```
$cnp= array(array(1),array(1,1),
            array(1,2,1),array(1,3,3,1),
            array(1,4,6,4,1)
           );
echo $cnp[2][1];  // = 2
echo $cnp[4][2];  // = 6
```


Les fonctions

`function` *nomFonction*(*arguments*){*code*}

- Les noms de paramètres ont la syntaxe des noms de variables ($).
- L'instruction `return` *expression* ; fixe le résultat, dont le type est quelconque.
- Par défaut, les paramètres sont passés « par valeur »
- Possibilité de donner des valeurs par défaut aux paramètres
- Pas d'accès automatique aux variables globales
- La surcharge des fonctions **est interdite**.

Les paramètres par défaut

- La valeur par défaut est précédée du signe =
- Le paramètre peut alors être omis au moment de l'appel.
- Quand on a défini une valeur par défaut pour un paramètre, il faut le faire pour tous les suivants.
- Lors de l'appel, les valeurs présentes sont attribuées aux paramètres formels, de gauche à droite

exemple

```
function println($s = "")
{
  echo $s . "<br />\n";
}
```


Paramètres passés par référence

- faire précéder le nom du paramètre formel par le caractère &.
- toute modification du paramètre formel, agit sur le paramètre effectif
- le paramètre effectif est nécessairement une variable (pas une constante !)
- pas de valeur par défaut possible pour ces paramètres

```
function doublerVariable(& $i)
{
  $i = $i*2;
}
$x = 5;
doublerVariable($x);
echo $x; //affiche 10
```


Portée des variables

une « variable globale » n'est pas visible à l'intérieur d'une fonction

```
$a=12; $b=20;
function print_ab(){
  $a=0;
  echo "|".$a."| |".$b."|";
}
print_ab();
echo "|".$a."| |".$b."|";

//affiche |0| || |12| |20|
```

Pour accéder à une variable globale depuis une fonction, on doit la « déclarer » précédée du mot `global`.

Mot clé `global`. / Variables super globales

« variable globale » rendue visible à l'intérieur d'une fonction

```
$a=12; $b=20;
function print_ab(){
  global $a;
  echo "|".$a."| |".$b."|";
  $a=100;
}
print_ab();
echo "|".$a."| |".$b."|";
//affiche |12| || |100| |20|
```

« variables super globales », visibles de partout

- uniquement quelques variables prédéfinies
- commencent par $_ et s'écrivent en majuscules
- $_REQUEST $_GET $_PUT $_SESSION ...

Variable **rémanente**

retrouve sa valeur d'une exécution à l'autre de la fonction

- la déclarer, précédée du mot-clé `static`
- initialisation obligatoire. Réalisée uniquement lors du premier appel à la fonction.
- cela reste une variable locale à la fonction

```
function testRemanent(){
  static $a = 0;
  echo "|".$a."|";
  $a++;
}
testRemanent(); testRemanent(); testRemanent();
// affiche |0| |1| |2|
```


Les objets et les classes

- modèle de **classes** comportant **attributs** et **méthodes**.
- les composants peuvent être publics ou privés
- le **constructeur** unique s'appelle `__construct()`
- dans la classe, `$this` désigne l'objet lui-même.
- usage de `$this` obligatoire pour accéder aux attributs depuis les méthodes
- Syntaxe :
  - la désignation d'un composant (méthode ou attribut) utilise la notation `->` et non le point (≠ java)
  - les attributs commencent par $, dans la déclaration. Mais le $ disparaît quand l'attribut est précédé de `->` (étrange ! !)

```
class C {
  public $att = 10; // ...
}
```

```
$obj = new C();
$x = $obj->att;
```

Les objets et les classes

```
class Fraction {
  private $numerateur = 0;
  private $denominateur = 1;
  public function numerateur() { //accesseur
    return $this->numerateur;
  }
  public function denominateur() {/* à compléter...*/}
  public function enNumerique() {
    return $this->numerateur / $this->denominateur;
  }
  function __construct($num = 0, $den = 1) {
    if ($den == 0)
       throw new Exception('Denominateur nul');
    $p = pgcd($num,$den);
    $this->numerateur = $num / $p;
    $this->denominateur = $den / $p;
  }
} // fin de la classe Fraction
```


Organisation du code : quelques bonnes pratiques

Idée générale : séparation maximale du code PHP/HTML

- Éviter les « lasagnes de code » (PHP/HTML/PHP/HTML/..)
- Modulariser le code PHP
- Distinguer les quelques fichiers qui doivent être directement accessibles et protéger tous les autres

Comment ?

- Faire des fichiers de « **bibliothèques** de fonctions »
- **Une seule classe** par fichier, nommé `NomDeLaClasse.class.php`
- Bibliothèques et classes dans un **sous-répertoire protégé**.
- Dans une page HTML/PHP mettre les calculs dans un block PHP initial et limiter ensuite l'insertion de PHP dans le HTML à quelques `echo`.

Organisation du code : modularité

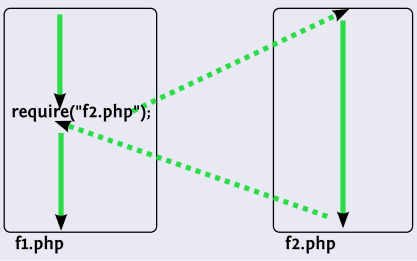
Inclusion de fichier

- `require(`*nom de fichier*`)` inclut le contenu du fichier. Stoppe le programme en cas d'absence du fichier.
- même règle que pour les autres fichiers PHP :
  - le contenu des blocs `<?php ... ?>` est interprété. Même contexte d'exécution que celui du script principal.
  - le texte hors de ces blocs est envoyé tel quel sur la sortie standard.
- Variante : `require_once` (garantit une seule inclusion du fichier par run PHP),
- Utiliser `require` et `require_once` plutôt que `include`, `include_once`

Inclusion automatique des fichiers de classes

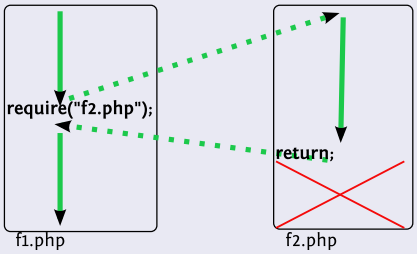
Règles à definir avec `spl_autoload_register()` (PHP>5.1.2) ou `__autoload()`

Déroulé d'un programme avec `require`



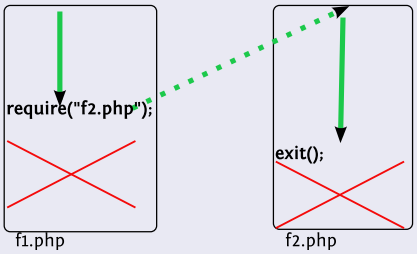
L'instruction `return`

- Dans une fonction, `return` provoque l'arrêt de la fonction et fournit le résultat.
- Hors d'une fonction : `return` provoque l'arrêt du script du fichier courant. Si celui-ci a été appelé par un `require` (ou `include`), l'interprétation du script appelant continue.



L'instruction `exit`

- L'instruction exit met fin à l'exécution du script et des scripts appelants.



Revenons sur les variables

Test d'existence - Suppression

- La fonction booléenne `isset($v)` `teste si $v est définie`
- La fonction `unset($v)` `supprime $v`

Connaître le type de la valeur contenue

- Les fonctions `is_int($v)`, `is_string($v)`, `is_array($v)`, `is_object($v)`,... etc.. permettent de tester le type de contenu