UNIVERSITE PARIS 8
Vincennes - Saint-Denis

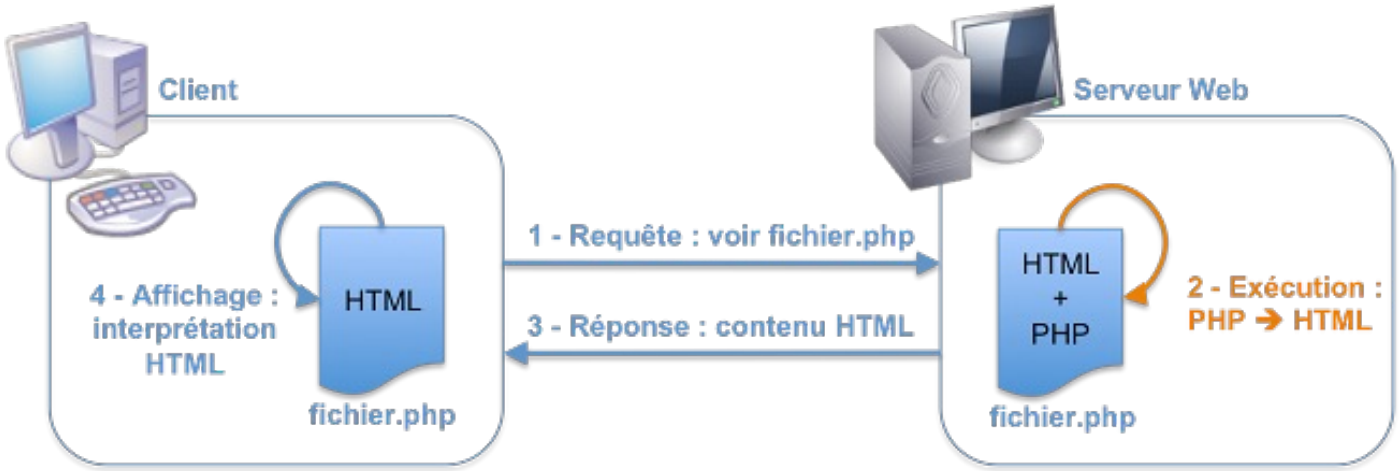
PHP (cours 1)

M2 G2M, Univ. Paris 8
par Isis TRUCK

Inspiré de diverses sources comme
http://php.net/manual/fr/
http://deptinfo.cnam.fr
http://www.w3schools.com
https://apical.xyz/formations/php#chapitre-passer_de_l_information_entre_les_pages_web

Introduction (1)

- PHP : au départ, « *Personal Home Page* » puis, est devenu « *PHP: Hypertext Preprocessor* »
- Langage serveur, interprété, orienté Web
- est **interprété par le serveur** qui génère ensuite du code (HTML, XHTML, CSS, Javascript…) interprété **sur l'ordinateur du client** (= visiteur)



Source: www.enseignement.polytechnique.fr

Introduction (2)

- Fonctionnement – exemple :
  - Le serveur reçoit la requête HTTP demandant à voir la page `req.php`
  - Le serveur cherche dans son arborescence le fichier
  - puis appelle un programme (une librairie)
  - Cette librairie interprète puis exécute le fichier script `req.php`
  - elle produit du code HTML, XHTML, JavaScript… selon ce que contient le script PHP
  - le serveur renvoie au client un fichier (nommé `req.php`) contenant ce code produit
  - Le client, bien que le fichier s'appelle `req.php`, ne voit dans ce fichier que du code HTML, XHTML, JavaScript (jamais de code PHP)
  - Le navigateur du client interprète le code produit (HTML, XHTML, JavaScript…)
  - et affiche à l'écran du client le résultat de cette interprétation

Bases (1)

- Comme en JavaScript, un fichier script PHP peut contenir du HTML, en plus du PHP
- Pour distinguer le code PHP, on l'entoure des balises : `<?php` et `?>`
  - NB1 : on peut écrire `<?` à la place de `<?php` mais non recommandé car moins portable
  - NB2 : on peut également, comme en JavaScript, écrire `<script language="php">` et `</script>`
- L'interprète lit le script au fur et à mesure et « emmagasine » le code se trouvant entre les balises PHP

Bases (2)

- Par exemple, ce script fonctionne correctement :

```
<?php if ($expression == true): ?>
  Ceci sera affiché si l'expression est vraie.
<?php else: ?>
  Sinon, c'est cela qui sera affiché.
<?php endif; ?>
```

- Séparateur d'instructions « **;** » obligatoire sauf si **?>** suit la dernière instruction
- La commande `echo` affiche le contenu de ses paramètres à l'écran:
  - Paramètres chaînes de caractères doivent être entourés de simples ou doubles quotes
  - Les simples quotes ne réalisent pas de substitution ni pour les variables, ni pour les caractères de contrôle (\n, \t , \r…)
  - Avec les doubles quotes, les paramètres peuvent inclure des variables : la substitution sera réalisée

Bases (3)

- Ainsi, on peut intégrer un appel de fonction dans un echo, à condition de mettre des guillemets :

```
echo "La variable en minuscules: ",
strtoupper($var);
```

- La commande `print` existe également, mais se comporte comme une fonction, contrairement à `echo`.
- `exit` permet de terminer un script PHP. Exemple:

```
$nomFic = 'rep1/rep1.1/fic.txt';
$fluxFic = fopen($nomFic, 'r')
        or exit("Erreur : impossible d'ouvrir
        le fichier $nomFic");
```

- Commentaires:
  - Sur une seule ligne : `//` ou `#`
  - Sur plusieurs lignes : `/* … */`

Variables (1)

- Pas de zone déclarative dédiée dans un script
- Les variables sont déclarées avec `$`

```
$var=1;
```

- Déclarées dans un script, elles ont une portée globale au script, **sauf dans les fonctions** (les fonctions ne voient pas les variables du script)
- Si une variable est déclarée dans une fonction, celle-ci aura une portée locale à la fonction
- Les variables sont à typage faible (comme en JavaScript) : c'est PHP qui décide du type lors de l'affectation.
- 6 types de variables : entier (int, integer); décimal (real, float, double), chaîne de caractères (string), tableau (array), objet (object), booléen (boolean)

Variables (2)

- Forcer le type d'une variable : fonction `settype` ou bien on « caste » avec (int), (string) , (real), (array)…
- Prototype : `bool settype (mixed &$var, string $type)`
  - `$type` est integer, boolean, float...
  - &$var signifie que $var est passé à la fonction **par référence**, càd que la fonction peut modifier cette variable (voir plus loin les passages des paramètres aux fonctions)
- Il existe également des fonctions de conversion comme:
  - `string strval($var)` : convertir en chaîne
  - `int intval($var)` : convertir en integer
  - `float floatval($var)` : convertir en réel

Fonctions et passage des paramètres

- Passage des paramètres en PHP:
  - Par défaut, passage par **valeur** (la fonction ne modifie pas la variable)
  - Si & est ajouté, passage par **référence**
- NB : Une **référence** à une variable `$a` est un élément qui **pointe sur la zone mémoire** de la variable `$a`
- Possible de passer un nombre variable d'arguments (comme en Javascript) , cf. fonctions `func_num_args()`, `func_get_arg()`, et `func_get_args()`
- Possible de définir des valeurs par défaut pour les arguments (des fonctions) de type *scalaire* (= nombres ou chaînes)

Tester les variables

- `isset()` teste si une variable est positionnée (définie)
- `unset()` supprime la variable, et désalloue la mémoire utilisée pour cette variable
- `gettype()` permet de connaître le type d'une variable:
  - elle renvoie une chaîne de caractères indiquant : "string", "integer", "double", "array" ou "object"‘
  - si la variable n'est pas définie, NULL est renvoyé
- Possible aussi de tester un type particulier avec les fonctions `is_array()`, `is_string()`, `is_int()`, `is_float()`, `is_object()`

Constantes

- Constante : sorte d'identifiant qui n'est affectable qu'une fois pour toutes
- Se comporte comme une « variable », mais sans changement de valeur possible
- Se définit en PHP avec `define` :
  - `define("cste", "M2G2M");`
- Pas de $
- Quelques constantes prédéfinies (constantes magiques):
  - `__FILE__` : contient le chemin complet et le nom du fichier courant
  - `__DIR__` : contient le répertoire du fichier courant
  - `__LINE__` : numéro de la ligne courante

Opérateurs (1)

- Opérateurs de comparaison
  - == égalité
  - < inférieur strict
  - > supérieur strict
  - <= inférieur ou égal
  - >= supérieur ou égal
  - != négation
- Opérateurs logiques (utilisés dans les tests, par exemple « if (condition) » ou « for (initialisation, condition, incrémentation) »)
  - `&&` ou `and` => et
  - `||` ou `or` => ou
  - `xor` => ou exclusif
  - `!` ou `not` => négation
- Opérateurs
  - de concaténation : **.**
  - ternaire : [test logique] **?** [expression si vrai] **:** [expression si faux]

Opérateurs (2)

- Opérateurs arithmétiques
  - `+` addition
  - `–` soustraction
  - `/` division *NB : renvoie un entier si les 2 opérandes sont des entiers, sinon, renvoie un flottant*
  - `*` multiplication
  - `%` modulo
  - `++` incrément
  - `– –` décrément
- Opérateurs d'affectation
  - `=` affectation
  - `+=` addition puis affectation
  - `– =` soustraction puis affectation
  - `*=` multiplication puis affectation
  - `/=` division puis affectation
  - `%=` modulo puis affectation

Tableaux

- Un tableau est un **ensemble ordonné de couples (clef, valeur)**
- La clef peut être un nombre ou une chaîne de caractères :
  - Lors de la déclaration, si la clef est un nombre ou si elle est omise, il s'agit d'un **tableau indexé**

Clef	Valeur
0	vert
1	jaune
2	orange

  - Si la clef est une chaîne de caractères, il s'agit d'un **tableau associatif**

Clef	Valeur
nom	dupond
prn	jean
age	20

Tableaux indexés

- Déclaration :

```
$couleurs= array();
```

- Affectation :

```
$couleurs[0]= "vert";
$couleurs[1]= "jaune";
$couleurs[] = "orange"; // équivalent à $couleurs[2]= "orange"
$couleurs = array( "vert", "jaune", "orange" );
```

- Fonctions relatives :
  - `count` : Renvoie le nombre d'éléments d'un tableau. Idem que sizeof().

    ```
    $nbElts= count( $tableau );
    ```
  - `is_array` : renvoie true si la variable est de type tableau (ou tableau associatif), false sinon.
  - reset : la fonction `reset($tableau)` place le pointeur interne sur le premier élément du tableau, chaque variable tableau possède un pointeur interne repérant l'élément courant

Tableaux associatifs

- Déclarations :

```
$client= array(); // idem tableau indexé
```

- Affectation :

```
$client["nom"] = "dupond";
$client["prn"] = "jean";
$client["age"] = 20;
```

- Ou bien déclaration + affectation en même temps

```
$client = array(
                "nom" => "dupond",
                "prn" => "jean" ,
                "age" => 20
);
```

Parcours de tableaux

- Le plus simple : fonction foreach() qui passe en revue le $tableau. A chaque itération, la valeur de l'élt courant est mise dans $valeur -- ou dans $clef et $valeur -- et le pointeur interne du tableau est avancé à l'élt suivant :

```
foreach ($tableau as $valeur)
    //commandes
```

ou

```
foreach ($tableau as $clef => $valeur)
    //commandes
```

- Bien sûr, il est toujours possible de faire :

```
<?php
// Calcul de la taille du tableau $couleurs
$tailleTab = count($couleurs);
// Parcours du tableau et affichage
for ($i=0; $i<$tailleTab; $i++) {
  echo $couleurs[$i] ,'<br />';
}
?>
```

Fonctions dédiées aux tableaux (1)

- `end` : la fonction `end($tableau)` place le pointeur interne du tableau sur le dernier élément du tableau.
- `current` : renvoie l'élément courant du tableau.
- `next` : déplace le pointeur vers l'élément suivant, et renvoie cet élément. S'il n'existe pas, la fonction renvoie false.
- `prev` : déplace le pointeur vers l'élément précédent, et renvoie cet élément. S'il n'existe pas, la fonction renvoie false.
- `key` : la fonction `key($tab)` renvoie l'index de l'élément courant du tableau.

- `each` (intéressante mais ***dépréciée*** => remplacée par `current` et `key`) : la fonction `$a=each($tab)` renvoie l'index et la valeur courante dans un tableau à 2 éléments : $a[0] contiendra l'index et $a[1] la valeur.

Fonctions dédiées aux tableaux (2)

- `sort` trie par valeurs croissantes, `rsort` par valeurs décroissantes un tableau indexé :
  `$valRetour=sort($tab); //modifie $tab`
- `asort`, `arsort` sont les équivalents pour les tableaux associatifs (tri sur les **valeurs**)
- `ksort` trie les tableaux associatifs sur les **clefs**
- `isset` : pour tester l'existence d'un élément, on utilise la fonction `isset()` :

```
if (isset($client["prn"])) {
   echo "le prénom du client est", $client["prn"];
}
else {
   echo "pas de prénom saisi pour le client\n";
}
```

Structures de contrôle (1)

- Si :

```
if (<condition>) {
   //le alors
}
```

- Si / sinon

```
if (<condition>) {
   //le alors
}
else {
   //le sinon
}
```

```
if (<condition>) {
   //le alors
}
elseif(<condition>) {
   //le sinon-si
}
```

- Switch

```
switch($var) {
   case <val1> : <instructions>
                 break; //pour stopper l'exécution du switch
   case <val2> : <instructions>
                 break; //break n'est pas obligatoire
   default: <instructions>
}
```

Structures de contrôle (2)

- Boucles ("pour" ou "tant que")
  - `break` permet de sortir d'une boucle (***for***, ***foreach***, ***while***, ***do-while*** ou ***switch***) à tout moment (comme en JavaScript)
  - `continue` permet, dans une boucle (***for***, ***foreach***, ***while***, ***do-while*** ou ***switch***) d'éluder les instructions de l'itération courante et donc, de commencer la prochaine itération (comme en JavaScript).
  - Comme en JavaScript, il existe 4 types de boucles
    - for (<initialisations> ; <cond. d'arrêt> ; <action à faire à chaque tour>) {<instructions>}
    - foreach (cf. page 17)
    - while (<condition>) <instructions>
    - do-while : do { <instructions> } while (<condition>);

Création de fonction

- Mot-clef `function`
- Une fonction peut prendre de 0 à n paramètres
- Elle peut ou non renvoyer une donnée (`return`)
  - Lors de la déclaration du prototype de la fonction (pour un manuel), on écrit le type de l'objet renvoyé et le type des paramètres, mais
  - Lors de la définition (dans le code PHP) de la fonction, on n'écrit pas le type de l'objet renvoyé, ni le type des paramètres.
  - Si rien n'est spécifié, elle renvoie la valeur NULL
- Exemples
  - ```
    function carre ($n) {
        return $n * $n;
    }
    echo carre(4);
    ```
  - ```
    function carre2 ($n) {
        $res=$n * $n;
    }
    echo carre2(4);
    ```
  - Qu'est-ce qui s'affiche à l'écran?

Variables d'environnement (ou prédéfinies) (1)

- On a vu que les variables d'un script ne sont pas visibles dans les fonctions (p. 7)
- Cependant, si l'on veut tout de même accéder à ces variables, il existe le mot-clef `global` ou la variable prédéfinie `$GLOBALS` (tableau associatif)

```
$a = 1;
function increm($b) {
    global $a;
    $a = $a + $b;
}
increm(5);
echo $a;
```

```
$a = 1;
function increm($b) {
    $GLOBALS['a']=$GLOBALS['a']+$b;
}
increm(5);
echo $a;
```

Variables d’environnement (ou prédéfinies) (2)

- Il existe d’autres variables prédéfinies :
  - $GLOBALS — Référence toutes les variables disponibles dans un contexte global
  - $_SERVER — Variables de serveur et d'exécution
  - $_GET — Variables HTTP GET (qu’est-ce que c’est ?)
  - $_POST — Variables HTTP POST (qu’est-ce que c’est ?)
  - $_FILES — Variable de téléchargement de fichier via HTTP
  - $_REQUEST — Variables de requête HTTP
  - $_SESSION — Variables de session (enregistrées dans la session attachée au script)
  - $_ENV — Variables d'environnement
  - $_COOKIE — Cookies HTTP
  - $argc — Le nombre d'arguments passés au script PHP
  - $argv — Tableau d'arguments passés au script PHP

Sessions

- Pour conserver des variables sur toutes les pages du site et pendant toute la durée de présence de l'internaute, on utilise les **sessions** (plus facile qu'avec GET ou POST)
- on démarre une session (**sur toutes les pages du site**) avec la fonction `session_start()` (**avant d'écrire tout autre code**) et on la termine avec `session_destroy()`
- `session_destroy()` est appelée automatiquement en cas d'inactivité prolongée, mais on peut aussi l'invoquer en créant un bouton *`Déconnexion`*.
- La **durée d'une session** est définie dans le fichier `php.ini` avec la variable :
  - `session.gc_maxlifetime` (entier) spécifie la durée de vie (en secondes) des données sur le serveur. Après cette durée, les données seront considérées comme obsolètes, et peuvent potentiellement être supprimées. Par défaut, il vaut 1440 secondes (24 min).
  - Donc `session.gc_maxlifetime` indique le nombre de secondes pendant lequel la session peut demeurer active après la dernière action de l'internaute (ex : après le dernier clic effectué).
  - La session demeurera donc toujours active tant que l'usager effectuera au moins une action sur le site Web toutes les 24 minutes.
  - Note : dans les faits, la durée de vie sera aussi influencée par les variables session.gc_probability et session.gc_divisor.

Modifier la durée de vie d'une session

- `session.cookie_lifetime`
  - Cette variable, aussi présente dans le fichier php.ini (mise à 0 par défaut), fonctionne à l'inverse. Elle indique le nombre de secondes après lequel la session sera réinitialisée, que l'internaute soit actif ou pas.
  - La valeur 0 signifie que la session ne sera pas détruite automatiquement tant que le navigateur sera ouvert. Donc, une fermeture du navigateur cause automatiquement la destruction de la session.
  - Si l'on met : `session.cookie_lifetime = 3600` (dans php.ini), alors la durée maximale d'une session sera d'une heure. Après cette durée, une session sera automatiquement détruite, même si l'usager est toujours actif (même s'il a effectué un clic il y a 1 minute). Par contre, l'usager pourra fermer son navigateur puis le réouvrir et, s'il retourne sur le site Web avant que le délai soit dépassé, sa session sera toujours vivante.
- Modifier une configuration de php.ini par programmation
  - La fonction `ini_set()` permet de modifier une configuration de php.ini sans modifier ce fichier. La modification sera active seulement pour la durée du programme PHP.
  - Ex : `ini_set('session.gc_maxlifetime', 7200);`
  - Attention : cette instruction doit être placée AVANT le session_start() pour qu'elle soit prise en compte.

Cookies (1)

- Les sessions permettent donc de stocker des informations pendant toute la durée d'une connexion, et d'échanger ces informations d'une page à l'autre. Mais si l'on veut sauvegarder les informations, même lorsque l'internaute n'est plus connecté ?
- On utilise des **cookies** !
- cookie : petit fichier qui est stocké **sur le disque dur de l'internaute** et non pas sur le serveur
- En principe, on doit sauvegarder dans **un** cookie **une seule information**
  - Par exemple, pour un forum, si l'on veut enregistrer le pseudonyme de l'internaute, son adresse mail, sa date d'inscription et son design (thème) d'affichage préféré, il vaut mieux utiliser *quatre* cookies
- C'est le navigateur du poste client qui gère l'emplacement de stockage des cookies (on n'a donc pas besoin de connaître cet emplacement)
- Chaque cookie a un **nom** et une **valeur**, et possède une **date d'expiration**
- Quand la date expire, il est automatiquement effacé du disque dur du client
- Un cookie contenant une adresse mail correspondra par exemple au couple `('adrMail','isis.truck@univ-paris8.fr')`

Cookies (2)

- On utilise la fonction `setcookie` pour créer un cookie
  - Elle nécessite au minimum 3 paramètres: le nom, la valeur et la date d'expiration, exprimée en secondes écoulées depuis le 1er janvier 1970.
  - On obtient la date d'expiration en appelant la fonction `time()` et en lui ajoutant un temps en secondes
  - Par exemple, pour créer le cookie pour le thème d'affichage préféré, on écrira:
    - `setcookie('themePrefere', 'fleursBleues', time() + 3600*24*7);}`
    - Combien de temps le cookie sera-t-il valide?
- Il existe d'autres paramètres à la fonction `setcookie()`, notamment le 7e et dernier qui correspond à l'option `httpOnly`
  - Pour protéger le cookie, il faut mettre ce paramètre à `true`
  - En effet, ce paramètre prend pour valeur un booléen qui définit si le cookie sera accessible uniquement par le protocole HTTP
  - Cela signifie que le cookie ne sera pas accessible *via* des langages de scripts comme JavaScript
  - Cette configuration permet de limiter les attaques via XSS (bien qu'elle ne soit pas supportée par tous les navigateurs)

Cookies (3)

- En général, on met les autres paramètres à `false` ou `null` (consulter la signature de la méthode `setcookie()` pour plus de détails)
- Attention, `setcookie` crée un (seul) cookie et s'écrit **avant tout code HTML**
- Tous les cookies envoyés au serveur par le client seront automatiquement inclus dans un tableau auto-global `$_COOKIE`
- Le serveur lit donc le contenu d'un cookie en regardant le contenu du tableau associatif `$_COOKIE[]`
  - Par exemple, pour afficher ce que contient le cookie de nom `adrMail`, il suffit de taper `echo $_COOKIE['adrMail']`
- NB: comme pour les sessions, il faut d'abord vérifier que le cookie existe bel et bien avec `isset()`

Fichiers (1)

- PHP permet d'accéder au SGF (qu'est-ce que c'est?) avec des fonctions prédéfinies
  - *NB : À quels fichiers accède-t-on ?*
  - Ouvrir et fermer un fichier : fopen/fclose
  - Lire un fichier : fread, fgets, file_get_contents, readfile
  - Écrire dans un fichier: fwrite, fputs
  - Copier un fichier : copy
  - Renommer un fichier : rename
  - Test de l'existence d'un fichier: file_exists
  - Effacer un fichier : unlink

Fichiers (2)

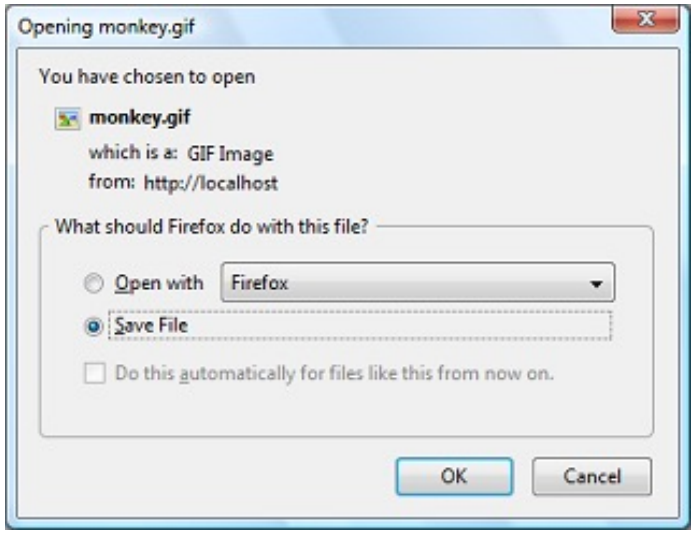
- ouvrir un fichier signifie « ouvrir un robinet » et « l'eau qui coule » est le contenu du fichier. On ouvre un *flux*. Et on peut ensuite lire ou écrire le fichier.
  - `fopen` : ouvre une *ressource nommée* (un *flux*, ici) spécifiée par le paramètre nomFic, dans le mode $mode :
    - `resource` **`fopen`**`(string $nomFic, string $mode)`
    - `$mode` peut valoir : `'r', 'r+', 'w', 'a'`, etc. (chercher les différences)
  - `file_get_contents` : Lit tout un fichier dans une chaîne
  - `file` : Lit le fichier et renvoie le résultat dans un tableau
  - `fgets` : récupère la ligne courante sur laquelle se trouve le pointeur du fichier (que l'on récupère grâce au flux)
    - `string` **`fgets`** `( resource $handle [, int $length ] )`

Fichiers (3)

– readfile : lit un fichier et l'envoie dans le buffer de sortie. Par exemple, on peut forcer le téléchargement d'un fichier :

```
<?php
    $file = 'monkey.gif';

    if (file_exists($file)) {
        header('Content-Description: File Transfer');
        header('Content-Type: application/octet-stream');
        header('Content-Disposition: attachment; filename='.basename($file));
        header('Content-Transfer-Encoding: binary');
        header('Expires: 0');
        header('Cache-Control: must-revalidate');
        header('Pragma: public');
        header('Content-Length: ' . filesize($file));
        ob_clean();
        flush();
        readfile($file);
        exit;
    }
?>
```



Modules – réutilisation de code

- Pour des questions de réutilisation de code, il est préférable d'écrire les scripts dans des fichiers à part.
- Il est également préférable de regrouper dans un même script seulement des instructions homogènes.
- Ensuite, on inclut le fichier script dans le fichier appelant
  - `include` : inclusion *dynamique* du fichier (il est lu puis interprété)
  - `require` : inclusion *avant l'interprétation* du code

`include` **ou** `require` ?

- Exemple

```
if( $user == "Administrateur" ) {
    include 'admin_fonctions.php';
}

if( $user == "Administrateur" ) {
    require 'admin_fonctions.php';
}
```

Résultat OK

Fichier inclus dans tous les cas

  - Si le fichier à utiliser est **essentiel** pour le script, qu'il ne peut plus rien effectuer sans avoir exécuté son contenu, alors **`require`**.
  - Sinon, càd si le fichier n'est pas fondamental pour le script, utiliser **`include`**.

POO et PHP

Programmation Orientée Objet

- PHP permet la POO au travers des classes
- Classe = objet possédant des **propriétés** et des **méthodes** et utilisant les mécanismes d'**héritage** et de **polymorphisme**
  - Propriété (ou attribut ou champ ou membre) : caractéristique d'un objet. Pour y accéder, on écrit : `$objet->propriete` ou `$this->propriete`*, où* `objet` *est le nom de la classe en question et* `propriete` *le nom de la propriété (que veut dire $this?)*
  - Méthode: fonction qui s'applique à un objet (accès par `$objet->methode`)
  - Héritage: définition d'un objet comme appartenant à la même famille qu'un autre objet plus général, dont il hérite des attributs et des méthodes (publiques et protégées mais pas privées). **Tant qu'une classe n'écrase pas ces méthodes, elles conservent leur fonctionnalité d'origine.**
  - Polymorphisme: capacité d'un ensemble d'objet à exécuter des méthodes de même nom, mais dont le comportement est propre à chacune des différentes versions. (c'est le cas **d'écrasement** des méthodes).

Exemple : créer une classe

```
<?php
class Fruit {
  // Propriétés
  public $name;
  public $color;

  // Méthodes
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

NB: Dans une classe, les variables sont des **propriétés** et les fonctions des **méthodes**

Objets

- Les objets sont créés depuis les classes (ce sont des ***instances*** de classes). Ils possèdent toutes les propriétés et les méthodes définies dans la classe, mais ils ont des valeurs de propriété différentes.

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;     // $this indique l'objet courant et est disponible seulement à l'intérieur des méthodes
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit();    // apple et banana sont des instances de la classe Fruit
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

Constructeur - Destructeur

- Si un constructeur de classe est défini, c'est lui que PHP appellera lors du `new`.
- Permet d'initialiser les propriétés d'un objet au moment de sa création
- Un constructeur et un destructeur commencent toujours par __ (2 *underscores*)
- Un destructeur est appelé qd l'objet est détruit ou quand le script s'arrête.
- Si un destructeur d'objet est défini, PHP appellera automatiquement cette fonction à la fin du script.

Constructeur : exemple

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit("Apple");
echo $apple->get_name();
?>
```

Destructeur : exemple

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function __destruct() {
    echo "The fruit is {$this->name}.";
  }
}

$apple = new Fruit("Apple");
?>
```

Accès aux données dans une classe

```
<?php
class Fruit {
  public $name; // visibilité totale
  protected $color; // accès uniquement à l'intérieur
                    // de la classe et de ses descendants
  private $weight; // accès uniquement à l'intérieur
                   // de la classe
}

$mango = new Fruit();
$mango->name = 'Mango'; // OK
$mango->color = 'Yellow'; // ERROR
$mango->weight = '300'; // ERROR
?>
```

Héritage

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  protected function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
    // Appel de la méthode protected depuis la classe dérivée - OK
    $this -> intro();
  }
}

$strawberry = new Strawberry("Strawberry", "red"); // OK. __construct() est public
$strawberry->message(); // OK. message() est public et appelle intro() (qui est protected) depuis la classe dérivée
?>
```