

DBWEB 2 :
Architecture Web Dynamique
Licence 1ère Année

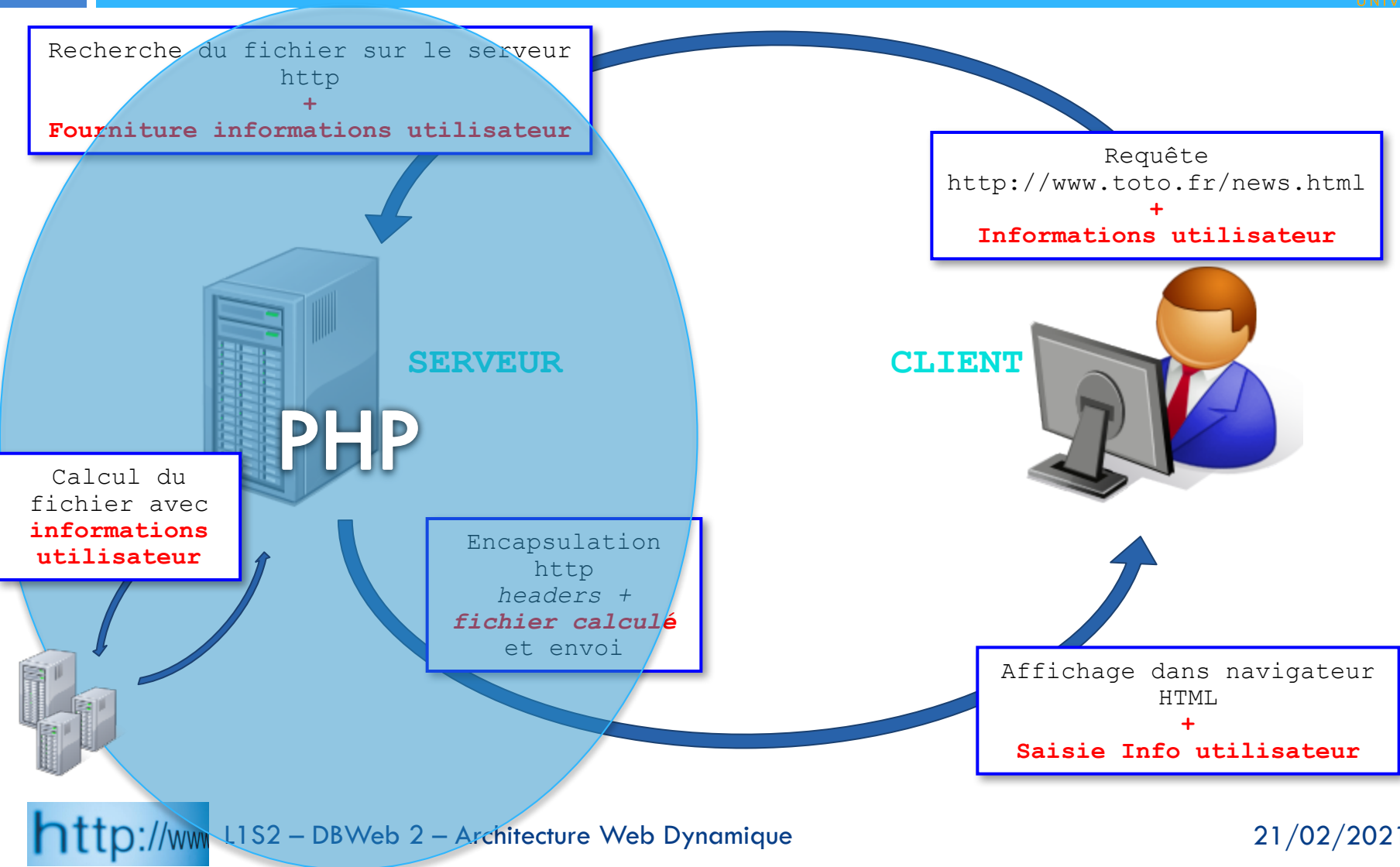
PHP

Fabrice Lefèvre

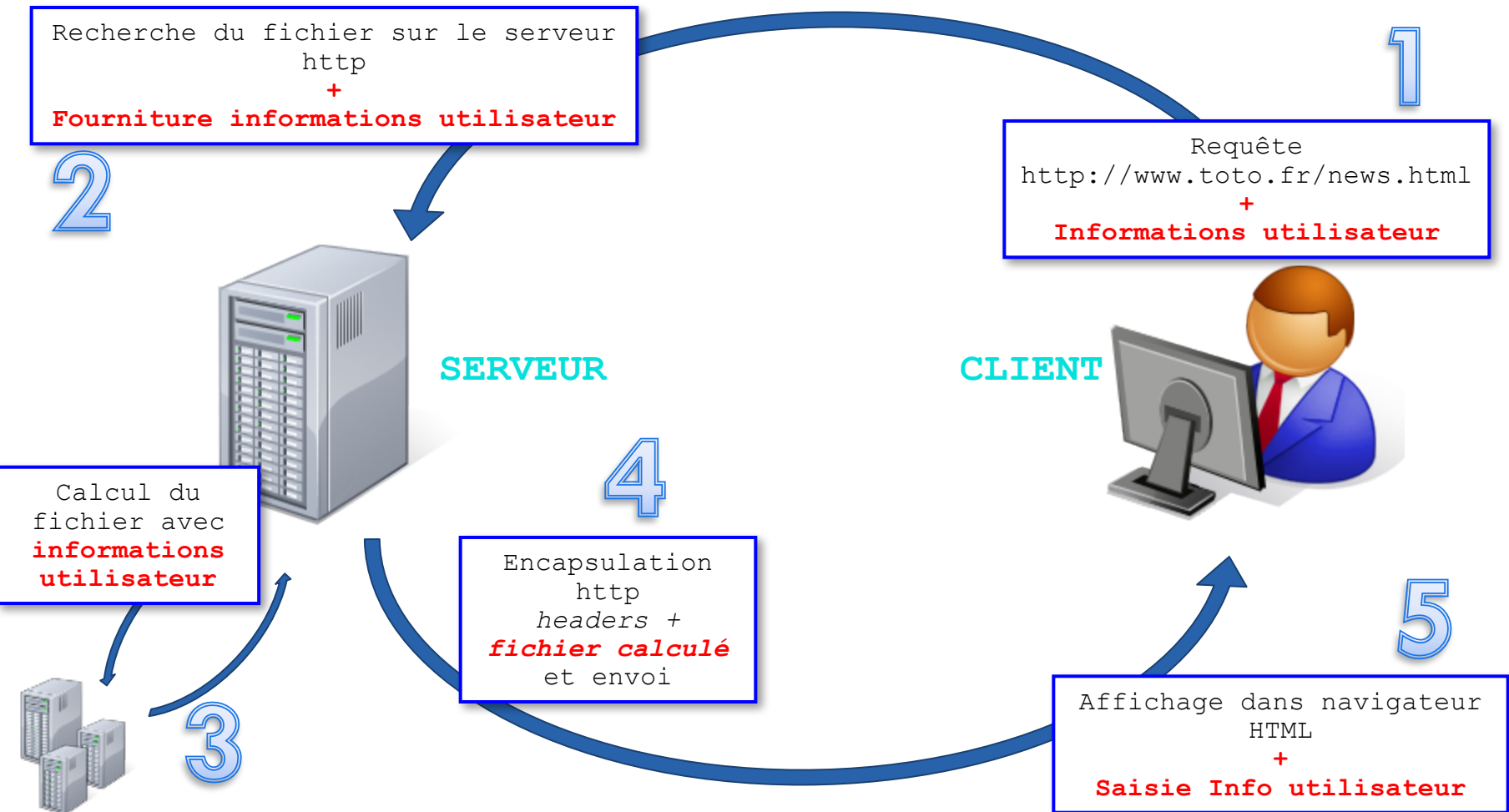
Fabrice.Lefevre@univ-avignon.fr

2021

Interaction dynamique et PHP



Interprétation (1)



Interprétation (2)

```
<html>
```

```
<head>
```

```
<titre>
```

Mon exemple

```
</titre>
```

```
</head>
```

```
<body>
```

```
<h1>Heure</h1>
```

```
<?
```

```
$t = time();
```

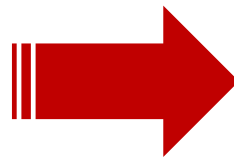
```
echo "il est $t <br/>";
```

```
?>
```

```
</body>
```

```
</html>
```

Interprétation
sur le serveur



3

```
<html>
```

```
<head>
```

```
<titre>
```

Mon exemple

```
</titre>
```

```
</head>
```

```
<body>
```

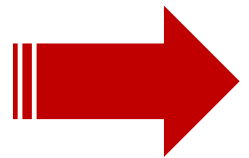
```
<h1>Heure</h1>
```

Il est 16h30


```
</body>
```

```
</html>
```

Transmis
au client



4

Premier code

```
<HTML>                                     @php_hello.php
<HEAD>
  <TITLE>Premier exemple</TITLE>
</HEAD>
<BODY>
  <?php
    echo "<p style='align:center'> Hi word</p>\n";
    echo "<p>Hello \"world\"</p>\n";
  ?>
</BODY>
</HTML>
```

Présentation

- Langage de script → interprété
- Rasmus Lerdorf en 1994
- PHP pour
 - "Personal Home Page", au départ
 - "PHP: Hypertext Pre-processor", maintenant
 - Ca fait plus sérieux !
- Actuellement PHP7 (version objet depuis PHP5)
- Moteur ZEND
- Compilation dynamique (avec cache)
 - cache : fichier temporaire conservé en mémoire pour réutilisation pendant un certain temps
 - Compilation OP CODE avec machine virtuelle (type Java)
- Communication facilitée avec de nombreux applications et protocoles

Intérêts

- ❑ Gratuité : logiciel ouvert
- ❑ Rapidité/optimalité
- ❑ Sécurité
- ❑ Compatibilité
- ❑ Exhaustivité
- ❑ Simplicité
- ❑ Documentation

Intégration HTML/PHP

```
<HTML>                                     @php_integration.php
<HEAD>
  <TITLE>Integration HTML/PHP</TITLE>
</HEAD>
<BODY>
  <H1>Le texte en HTML</H1>
  <?php
    // le code PHP -----
    $heure = date("H\hi");
    print("<span style=\'font-size:\"2\"\'\'>
      et celui en PHP.</span>");
  ?>
  <!-- retour au code HTML -->
  <BR /><font size="2" face="Arial">Il est
  <?php
    echo $heure;
  ?>
  </font>
</BODY>
</HTML>
```

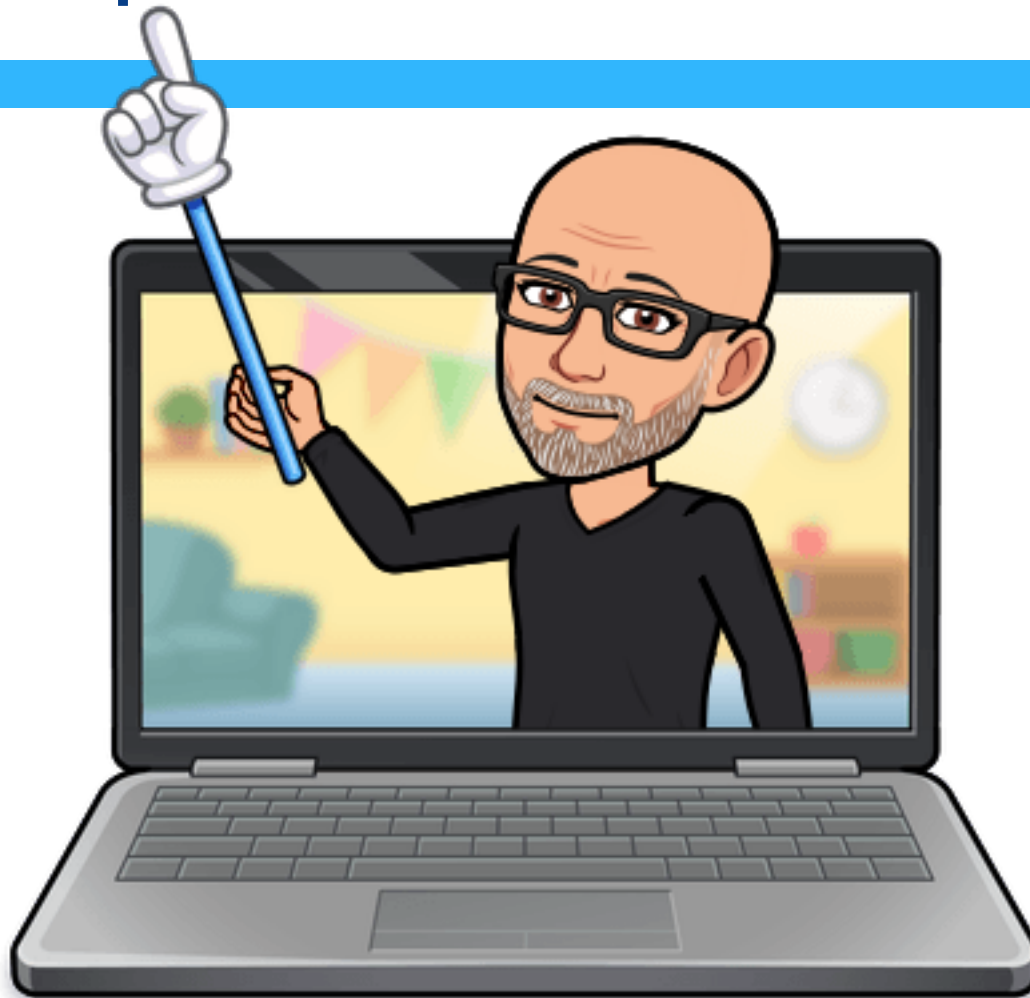
Balises possibles :

- <script language=php> ...
 </script>
- <?php ... ?>
- <? ... ?> (déconseillé)

Caractères spéciaux : \t, \r, \n

Commentaires : // ou /* ... */

En pratique



DBWEB 2 :
Architecture Web Dynamique
Licence 1ère Année

PHP syntaxe

Fabrice Lefèvre

Fabrice.Lefevre@univ-avignon.fr

2021

Variables et fonctions

```
@php_date.php

<HTML>
<HEAD>
  <TITLE>Variables et fonctions</TITLE>
</HEAD>
<BODY>
  <?php
    $date = date("d-m-Y");
    $heure = date("H:i");
    print("Date: $date - Heure: $heure");
  ?>
</BODY>
</HTML>
```

- Pas de déclaration
- Noms de variable précédés par \$
- Constitués de lettres, de chiffres, de soulignés, et de dollars uniquement.
 - pas d'espaces !
- Ne peuvent commencer par un chiffre
- Sensibles à la casse
- Constantes définies par la fonction `define()` (`define("PI", 3.1411)`)

Typage des variables

- Pas de déclarations mais types implicites (usages)
- Types de données
 - integer `$var = 12`
 - hex `$var = 0x23`
 - octal `$var = 045`
 - float `$var = 12.234002`
 - float `$var = 1.17e-3`
 - boolean `$var = true`
 - string `$var = "petit texte"`
- Attention (en PHP) :
 - `echo "Affiche le $string" //` substitution de la variable
 - `echo 'Affiche le $string' //` sans substitution de la variable

Opérateurs

- opérateurs arithmétiques (+, +=, -, *, /, %)
- opérateur de chaînes (. concaténation)
- opérateur d'assignement (=, =>)
- opérateurs combinés : assignement + arithmétiques (+=, .= ...)
- incrémentation avec \$i++
- décrémentation avec \$i--
- opérateurs de bits (& (ET binaire), | (OU binaire), ^ (OU exclusif), ~ (NOT, inversion) et << ou >> (décalages))
- opérateurs logiques (&& ou and, || ou or, ! ou not, xor)
- opérateurs de comparaison (==, !=, <, >, <= et >=)
 - === : même valeur et même type (id !=)

Chaînes de caractères

- Affichage avec echo et print
- Nombreuses fonctions disponibles :
 - classiques : `strlen()`, `strtoupper()`, `ucfirst()`, `trim()`, `substr()`, `str_replace()`, `ord()`, `chr()`
 - spéciales HTML : `addslashes()`, `stripslashes()`, `strip_tags()`, `htmlentities()`
 - sécurité : `md5()`, `sha1()`

Tableaux (1)

- Importance des tableaux : les fonctions PHP qui retournent plusieurs valeurs le font généralement sous la forme de tableaux.
 - ▣ par exemple les fonctions liées aux bases de données, mais aussi les variables venant du serveur HTTP.
- 3 types de tableaux en PHP :
 - ▣ indexés
 - ▣ associatifs
 - ▣ multidimensionnels

Tableaux (2)

- Tableaux indexés (ou numériques)
 - Tableaux classiques dans lesquels on se déplace en utilisant l'indice de l'élément
 - comme en langage C.
 - le premier élément est à l'indice 0.
 - On peut remplir un tableau en adressant chaque élément un par un, ou d'un coup en fournissant toutes les valeurs (fonction `array()`)

```
$tab=array(); $tab[0] = "P"; $tab[1] = "H";  
$tab[2] = "P"; $tab[3] = "3";  
// Equivalent à $tab = array("P","H","P","3");
```

Tableaux (3)

□ Tableaux associatifs

- Tableau qui contient pour chaque élément une valeur associée qui sert de clé d'accès

```
$vente_hebdo=array("lundi"=>1742,  
                  "mardi"=>1562,  
                  "mercredi"=>1920,  
                  "jeudi"=>1239,  
                  "vendredi"=>2012,  
                  "samedi"=>720) ;
```

- Parcours en utilisant les fonctions dédiées aux tableaux (voir structures de contrôles plus loin) :
 - each(), list(), reset(), foreach.

Tableaux (4)

- Tableaux multidimensionnels
 - ▣ Tableau dont chaque élément est lui-même un tableau.

- ▣ Exemples (2-dimensions) :

```
$tab[0] = array(1, "un", "premier");  
$tab[1] = array(2, "deux", "second");  
$tab[2] = array(3, "trois", "troisième");  
$tab[3] = array(4, "quatre", "quatrième");
```

```
$enfants = array ( "Paul"=>array("Pierre", "Marc", "Lucie"),  
"Fabrice"=>array("Rodrigue"), "Nathalie"=>array("Juliette", "Noa",  
"Emma") );
```

```
$tab[2][3] ?
```

```
$enfants[Nathalie][3] ?
```

Tableaux (5)

- Attention : les tableaux ne sont pas ordonnés selon leur indexation, mais dans l'ordre de leur initialisation
- La représentation interne correspond toujours au schéma “paires de clé – valeur”
 - ▣ clé : peut être de type entier ou string
 - ▣ valeur : tout type possible (y compris array, cas multidimensionnel)

```
$tab[0] = "jojo" ;
```

```
$tab[1] = 37 ;
```

```
$tab[2] = 75.5 ;
```

```
$tab[prenom] = "jojo" ;
```

```
$tab[age] = 37 ;
```

```
$tab[poids] = 75.5 ;
```



```
$tab[ ] = "jojo" ;
```

```
$tab[ ] = 37 ;
```

```
$tab[ ] = 75.5 ;
```

```
$tab[8] = "jojo" ;
```

```
$tab[14] = 37 ;
```

```
$tab[9] = 75.5 ;
```

Tableaux (6)

- Fonctions de manipulation de tableaux
 - ▣ Fonctions de tri
 - tableaux numériques : `sort()`, `rsort()`, `natsort()`, `shuffle()`
 - tableaux associatifs : `asort()` ou `arsort()` sur la valeur, `ksort()` sur la clé
 - ▣ Transformations
 - `list()`, `array_merge()`, `implode()`, `explode()`
 - ▣ Fonctions de parcours
 - `each()` ou `foreach()` (expliquées plus loin)
 - principe : le tableau est vu comme un objet qui intègre les informations de manipulations (parcours, dimension...)
 - `current()`, `reset()`, `next()` et `prev()` qui permettent respectivement de connaître la position actuelle du pointeur sur le tableau, de le remettre au début, d'avancer et de reculer ce pointeur d'un élément. `count()` retourne la taille du tableau
 - attention : ces fonctions sont à utiliser avec prudence car elles retournent `false` au premier élément dont la valeur est 0 ou ""

Structures de contrôle (1)

- Les structures de contrôle disponibles :
 - ▣ alternatives : if-else (elseif), ternaire (?), switch-case
 - ▣ itérations : while, do-while, for, foreach
 - ▣ mêmes constructions qu'en C et Java

- Ruptures d'itération :
 - ▣ permettent de modifier le comportement par défaut d'une l'itération
 - ▣ break : déclenche la sortie forcée de la boucle ou du test
 - ▣ continue : dirige l'exécution à la prochaine évaluation du test de continuation en sautant les éventuelles instructions complétant le corps de la boucle

Structures de contrôle (2)

□ Boucles et itérations

while :

```
while (test){  
    actions;  
}
```

do-while :

```
do {  
    actions;  
} while (test);
```

for :

```
for (instruction1; test; instruction2){  
    actions;  
}
```

- instruction1 : au départ; test : test d'arrêt; instruction2 : à chaque itération

Structures de contrôle (3)

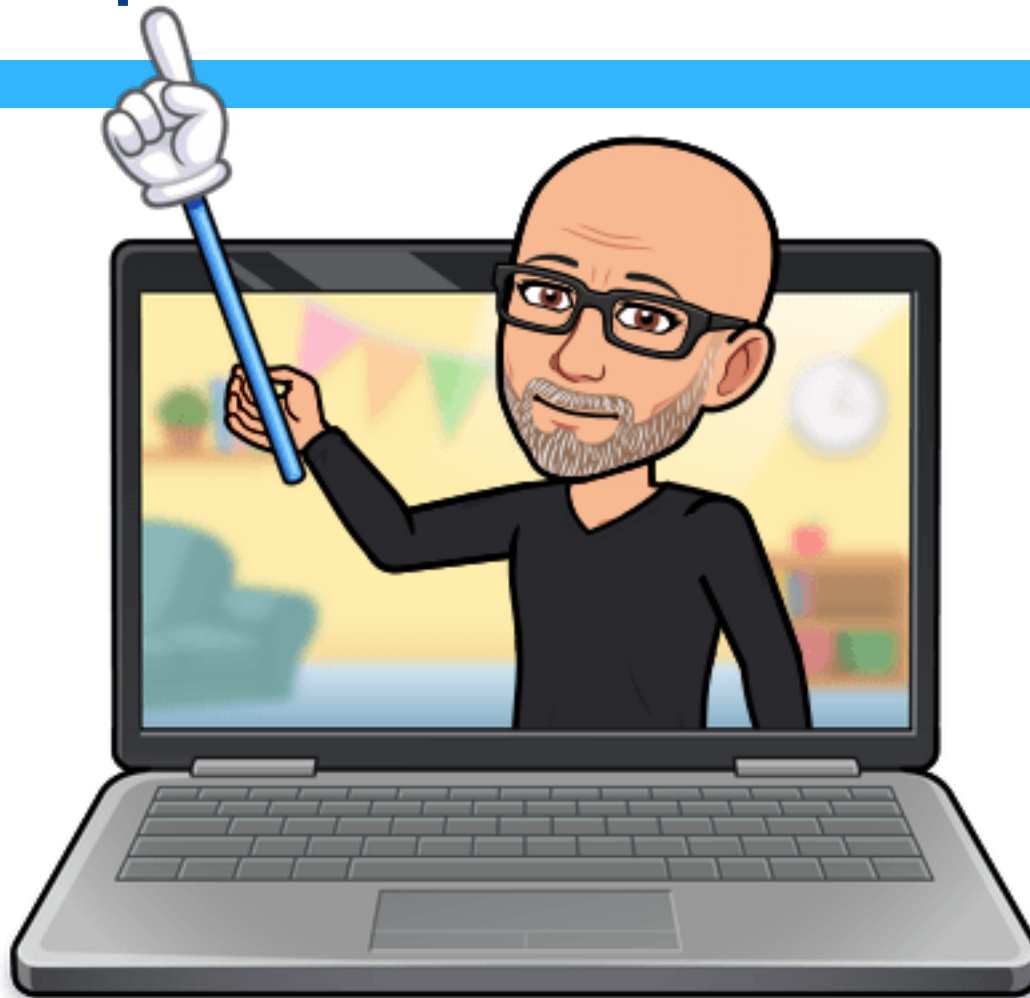
```
@php_exec.php
<HTML>
<HEAD>
  <TITLE>Listing</TITLE>
</HEAD>
<BODY>
  Les fichiers sont <BR />
  <?php
    // exécute la commande shell
    exec('ls -l .', $tab);
    $nb= count($tab) ;
    for ($n=0; $n<$nb; $n++){
      $fic = basename($tab[$n]);
      echo "<A href='$fic'> $fic </A><BR />";
    }
  ?>
</BODY>
</HTML>
```

Structures de contrôle (4)

@php_tabmultidim.php

```
<HTML>
<HEAD>
</HEAD>
<BODY>
  Les entiers ordinaux et cardinaux : <BR />
  <?php
    $tab[] = array(1, "un", "premier");
    $tab[] = array(2, "deux", "second");
    $tab[] = array(3, "trois", "troisième");
    $tab[] = array(4, "quatre", "quatrième");
    $c = count($tab);
    for ($i=0; $i<$c; $i++) {
      $l = count($tab[$i]);
      for ($j=0; $j<$l; $j++) echo $tab[$i][$j]. " ";
      echo "<br/>";
    }
  ?>
</BODY>
</HTML>
```

En pratique



Structures de contrôle et tableaux (1)

□ Boucle foreach

▣ Tableau indexé

```
foreach ($mon_tab as $valeur){  
    ... utilisation de $valeur ...  
}
```

- passe en revue le tableau \$mon_tab
- affecte à \$valeur la valeur de l'élément courant

▣ Tableau associatif

```
foreach ($mon_tab as $cle => $valeur){  
    ... utilisation de $cle et $valeur ...  
}
```

- passe en revue le tableau \$mon_tab
- affecte à \$cle la clé et à \$valeur la valeur de l'élément courant

Structures de contrôle et tableaux (2)

□ Fonctions `each()` et `list()` avec `while`

```
while (list($cle, $valeur) = each($mon_tab))  
{  
    ... actions utilisant les variables $cle et $valeur ...  
}
```

■ `each()` :

- permet de parcourir tous les éléments d'un tableau
- retourne la combinaison clé-valeur courante
- retourne la valeur `false` lorsque la fin du tableau est atteinte

■ `list()` :

- permet d'affecter plusieurs variables en un coup (par exemple, les éléments du tableau dans des valeurs distinctes)

Structures de contrôle et tableaux (3)

```
<?php
    $mon_tab= array("a","b","c","d");
    //on crée un tableau avec 4 valeurs
    while(list($cle, $valeur) = each($mon_tab))
    {
        echo"$cle: $valeur"." ";
    }
?>
```

OU

```
<?php
    $mon_tab= array("a","b","c","d");
    //on crée un tableau avec 4 valeurs
    foreach($mon_tab as $cle => $valeur)
    {
        echo"$cle: $valeur"." ";
    }
?>
```

Même résultat :
0:a 1:b 2:c 3:d

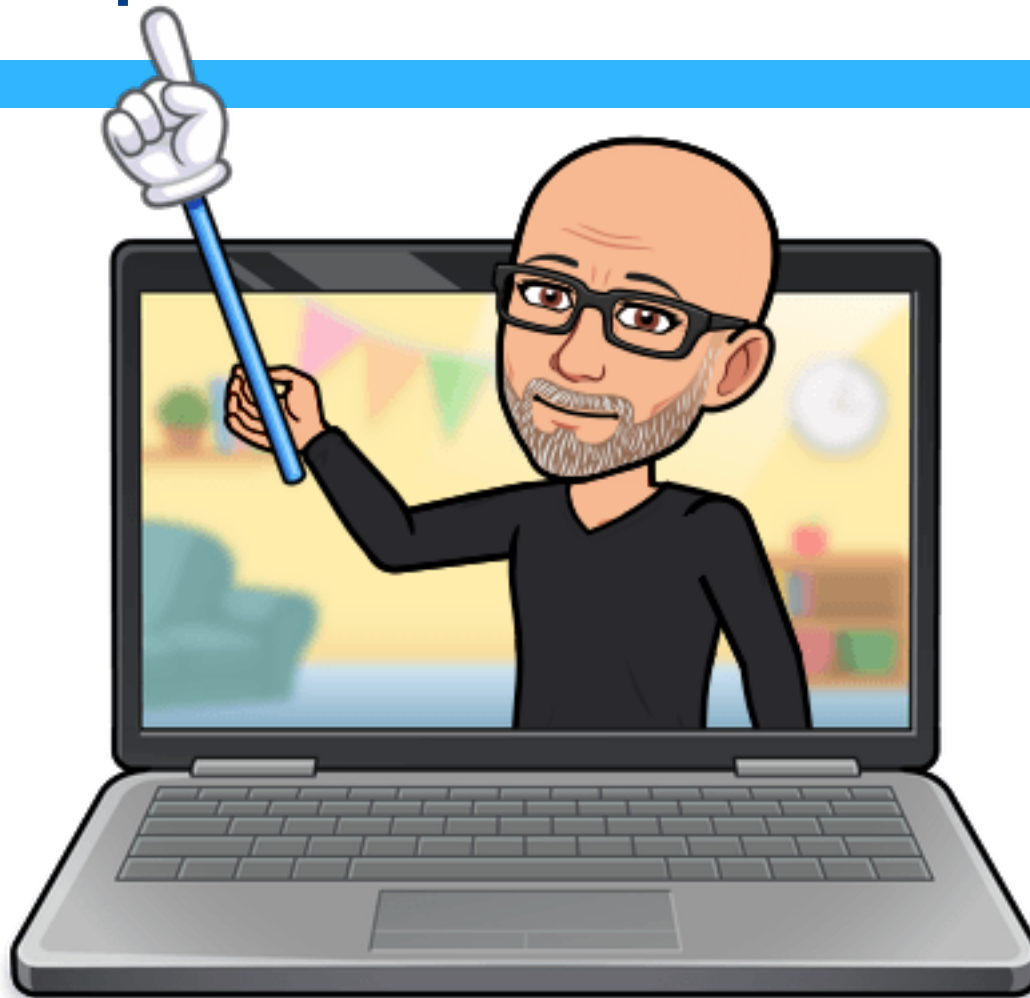
Structures de contrôle et tableaux (4)

@php_foreach.php

```
<HTML>
<HEAD>
  <TITLE>Listing</TITLE>
</HEAD>
<BODY>
  <?php
    $vente_hebdo=array("lundi"=>1742, "mardi"=>1562,
                      "mercredi"=>1920, "jeudi"=>1239,
                      "vendredi"=>2012, "samedi"=>720) ;

    $vente_totale= 0;
    reset($vente_hebdo);
    //while(list($key, $value) = each($vente_hebdo)) {
    foreach($vente_hebdo as $key => $value) {
      echo"<BR /> Ventes $key : $value unités\n";
      $vente_totale+= $value;
    }
    echo"<BR />Vente totale : $vente_totale unités\n"
  ?>
</BODY>
</HTML>
```

En pratique



DBWEB 2 :
Architecture Web Dynamique
Licence 1ère Année

PHP variables d'environnement

Fabrice Lefèvre

Fabrice.Lefevre@univ-avignon.fr

2021

Variables d'environnement

- Variables d'environnement = variables transmises par le serveur http et directement accessibles dans les scripts
- Les principales variables d'environnement disponibles pour PHP :
 - `$_SERVER` : informations sur le serveur web lui-même
 - `$_SERVER['HTTP_USER_AGENT']` : permet d'avoir des informations sur le type de navigateur utilisé par le client, ainsi que son système d'exploitation
 - `$_SERVER['REQUEST_METHOD']` : le résultat est GET ou POST
 - `$_SERVER['QUERY_STRING']` : récupère l'ensemble des informations transmises depuis un formulaire (visibles au niveau du navigateur en utilisant la méthode GET)
 - `$_POST` ou `$_GET` : tableau associatif global contenant les données transmises

Variables des formulaires (1)

- Lorsqu'un fichier PHP est traité suite à une requête comprenant des données d'un formulaire, les champs des formulaires deviennent des variables globales pour les scripts PHP du fichier
 - ▣ variables scalaires → une variable
 - ▣ variables multiples → un tableau
 - ▣ fichiers → un nom de fichier temporaire
- Les input de type text sont toujours des chaînes (qui peuvent être vides)
- Les checkbox n'existent que s'ils sont cochés (ils valent alors 'on')

Variables des formulaires (2)

- Si la méthode d'envoi est POST, il faut mettre comme nom
 `$_POST['nom_du_champ']`
 - ▣ attention : possibilité d'utiliser directement `$nom_du_champ`
 - obsolète et dangereux, à proscrire !
 - ▣ pour simplifier le nom des variables, on fait
 `$nom = $_POST['nom_du_champ']`
pour utiliser uniquement `$nom` ensuite.
 `@$nom = $_POST['nom_du_champ']`
pour neutraliser erreurs temporairement
- Il faut contrôler les informations soumises par le visiteur pour éviter au maximum les erreurs.
 - ▣ les contrôles les plus fréquents s'effectuent sur les URL et les email
 - par exemple tester si un champ texte email comporte bien un "@" et un point

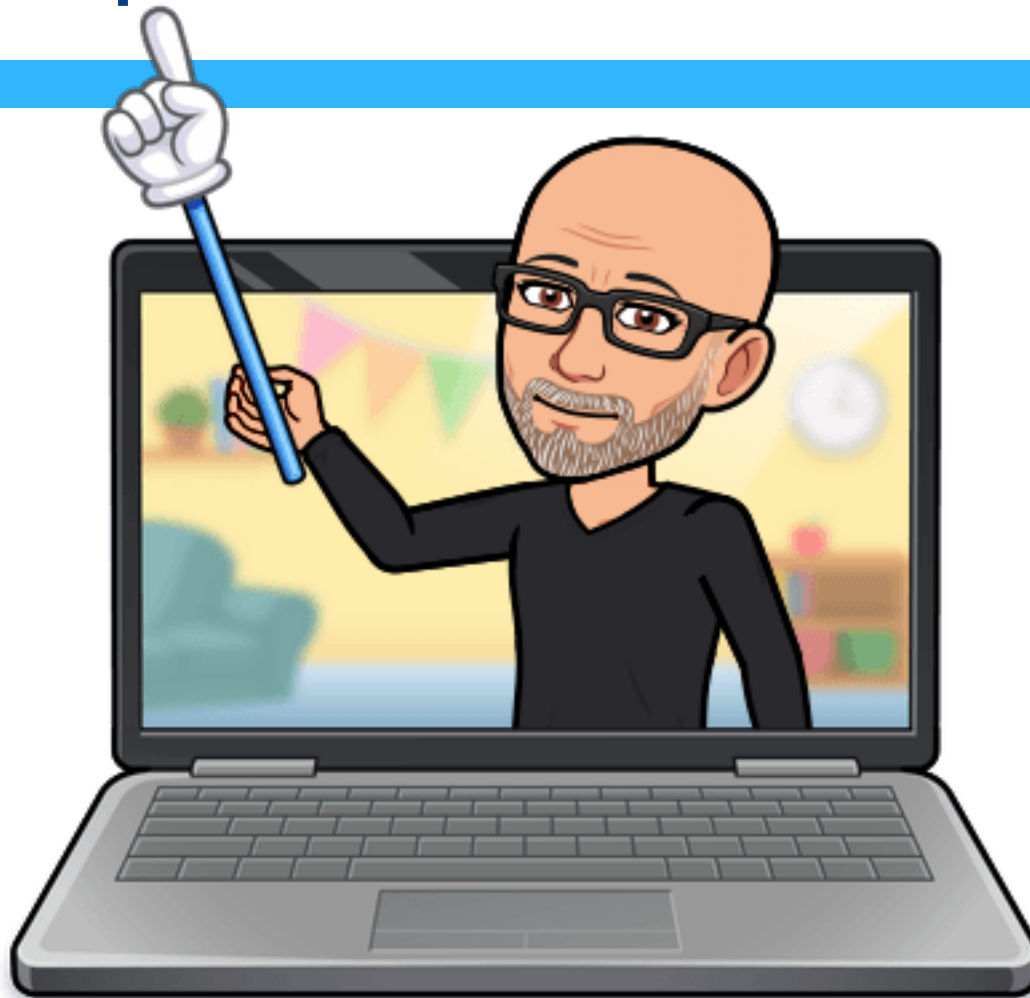
Variables des formulaires (3)

```
<HTML>                                     @php_nomprenom.php
<HEAD></HEAD>
<BODY>
  <FORM method="post" action="php_verif.php">
    Nom : <INPUT type="text" name="nom" size="12"><BR />
    Prenom: <INPUT type="text" name="prenom" size="12"><BR />
    <INPUT type="submit" value="ok">
  </FORM>
</BODY>
</HTML>
```

Fichier php_verif.php :

```
<HTML>
<HEAD></HEAD>
<BODY>
  <?php
    $prenom=$_POST['prenom'];
    $nom=$_POST['nom'];
    print("<CENTER>Bonjour $prenom $nom</CENTER>");
  ?>
</BODY>
</HTML>
```

En pratique



Traitements des variables

- Problème : beaucoup de variables sont fournies aux scripts PHP par l'environnement (serveur http)
 - ▣ besoin de vérifier leur état avant de les utiliser

- Quelques fonctions utiles :
 - ▣ tests de typage : `gettype()`, `is_long()`, `is_double()`, `is_string()`, `is_array()`, `is_object()`
 - ▣ `isset()` permet de savoir si une variable a été affectée (renvoie true si c'est le cas)
 - ▣ `empty()` c'est l'inverse de `isset`, renvoie true si la variable n'existe pas ou, si elle a pour valeur 0 ou chaîne vide
 - ▣ `unset()` détruit une variable, libérant ainsi la mémoire qui lui était allouée

Collecte des informations (1)

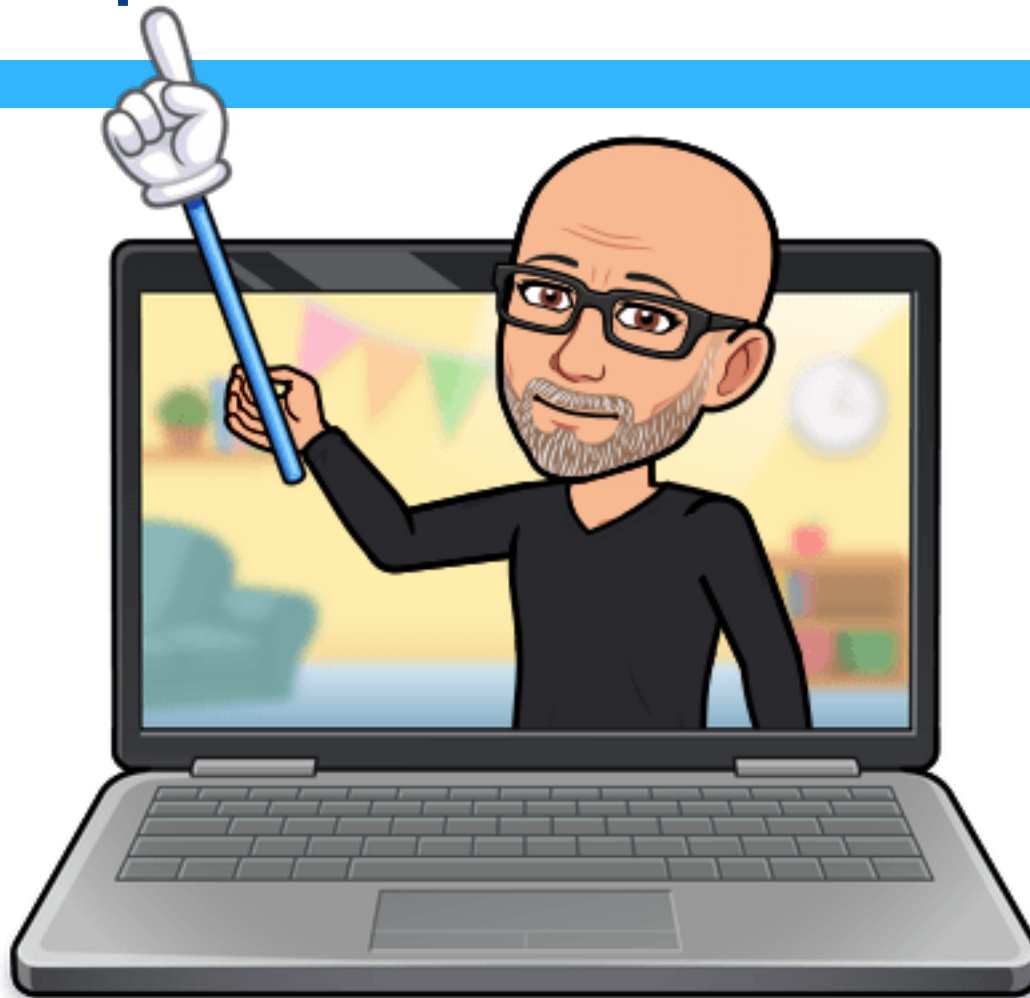
```
<HTML>                                     @php_email.php
<HEAD></HEAD>
<BODY>
  <?php
    @$email = $_POST['email'];
    if (empty($email)) {
      echo "Pas d'email ?";
    } else {
      $point = strpos($email, ".");
      $aroba = strpos($email, "@");
      if ($point == '') {
        echo "Votre email doit comporter un point";
      } elseif ($aroba == '') {
        echo "Votre email doit comporter un @";
      } else {
        echo "Email: '<A href=\"mailto:\"" . $email . "\"> $email</A>'";
      }
    }
  ?>
</BODY>
</HTML>
```

`empty()` permet de contrôler si un champs est vide
`strpos()` retourne la position d'un caractère dans une chaîne si celui-ci existe, sinon retourne ''.

Collecte des informations (2)

```
<HTML>                                                                 @php_traiteform.php
<HEAD><TITLE>Formulaire traité par PHP</TITLE></HEAD>
<BODY>
  <FORM action="<?php echo $_SERVER['PHP_SELF'] ?>" method="post"
enctype="application/x-www-form-urlencoded">
    Nom: <INPUT type="text" name="nom" size="40" /><br/>
    Débutant: <INPUT type="radio" name="niveau" value="débutant" />
    Initié:<INPUT type="radio" name="niveau" value="initié" /><BR />
    <INPUT type="reset" value="Effacer" />
    <INPUT type="submit" value="Envoyer" />
  </FORM>
  <?php
    if(isset($_POST["nom"]) && isset($_POST["niveau"])){
      echo"<h2> ".htmlentities($_POST["nom"])." est
".$_POST["niveau"]." en PHP</h2>";
    }
  ?>
</BODY>
</HTML>
```

En pratique



DBWEB 2 :
Architecture Web Dynamique
Licence 1ère Année

PHP fichiers

Fabrice Lefèvre

Fabrice.Lefevre@univ-avignon.fr

2021

Fichiers (1)

- Ouvrir un fichier :
 - ▣ `int fopen(string nom_fichier, string mode)`
 - ▣ retourne un descripteur de fichier. Cette valeur est ensuite utilisée pour accéder au fichier dans le script:
 - ▣ Modes possibles :
 - `r` : ouverture en lecture seulement
 - `w` : ouverture en écriture seulement (la fonction crée le fichier s'il n'existe pas)
 - `a` : ouverture en écriture seulement avec ajout du contenu à la fin du fichier (la fonction crée le fichier s'il n'existe pas)
 - `r+` : ouverture en lecture et écriture
 - `w+` : ouverture en lecture et écriture (la fonction crée le fichier s'il n'existe pas)
 - `a+` : ouverture en lecture et écriture avec ajout du contenu à la fin du fichier (la fonction crée le fichier s'il n'existe pas)

Fichiers (2)

- Fermer un fichier :
 - ▣ `bool fclose (int fp)` : un fichier ouvert avec la fonction `fopen` doit être fermé par la fonction `fclose` en lui passant en paramètre "int" retourné par la fonction `fopen`
- Lire dans un fichier :
 - ▣ `string fgets (int fp, int length)` : retourne la chaîne lue jusqu'à la longueur `length - 1` octet, ou bien la fin du fichier, ou encore un retour chariot (le premier des trois qui sera rencontré)
 - ▣ `array file (string filename [, int use_include_path])` : retourne le fichier dans un tableau. Chaque élément du tableau correspond à une ligne du fichier, et les retour-chariots sont placés en fin de ligne

Fichiers (3)

- Écrire dans un fichier :
 - ▣ `int fwrite (int fichier, string string [, int length])` ou
 - ▣ `int fputs (int fichier, string string [, int length])`
 - ▣ écrit le contenu de la chaîne "string" dans le fichier "fichier". Si la longueur "length" est fournie, l'écriture s'arrêtera après length octets, ou à la fin de la chaîne (le premier des deux)

Fichiers exemples (1)

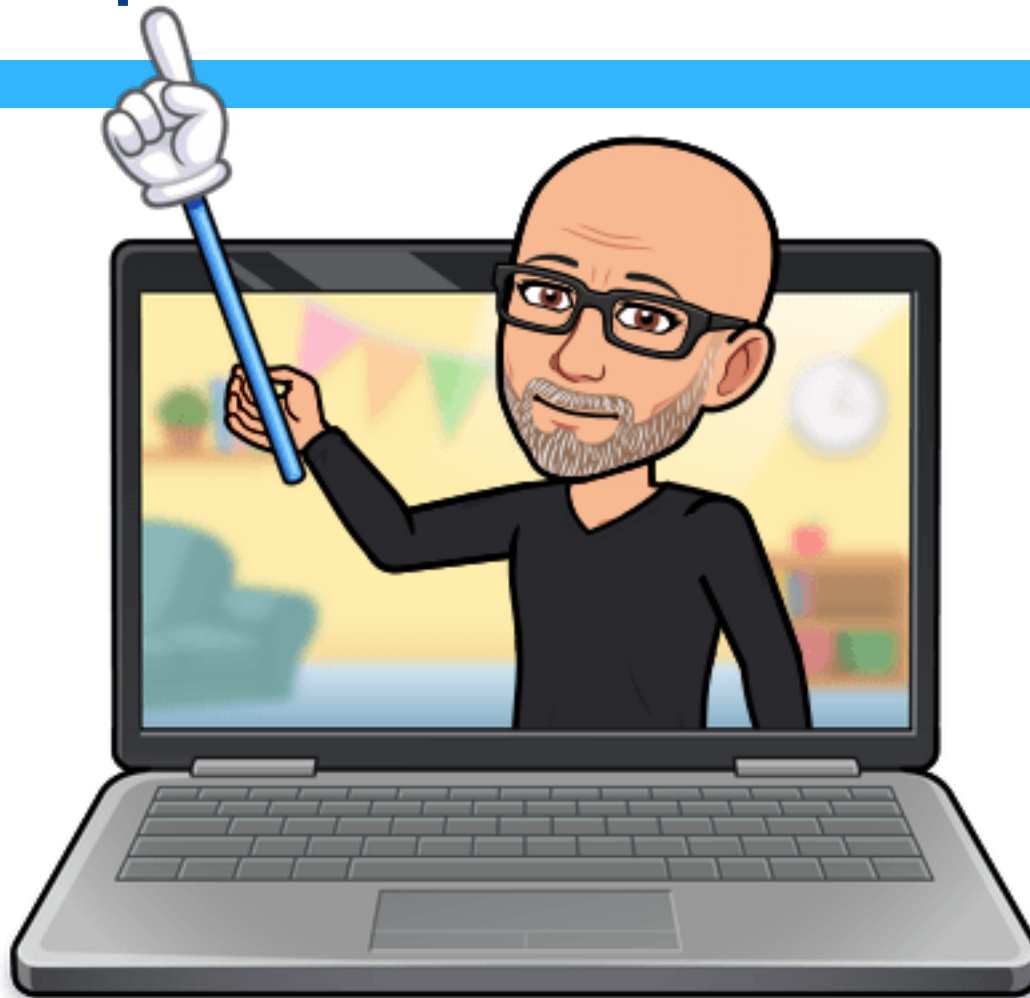
```
@php_file.php
<HTML>
<HEAD></HEAD>
<BODY>
  <?php
    $fp = fopen("php_filedata.txt","r");
    $donnees = fgets($fp,255);
    fclose($fp);
    echo 'Le fichier contient :'.$donnees;
  ?>
</BODY>
</HTML>
```

Mêmes fonctions
que le C/C++

Fichiers exemples (1)

```
<HTML>                                     @php_compteur.php
<HEAD></HEAD>
<BODY>
  <?php
    $fp = fopen("php_compteur.txt","r+");
    $hits = fgets($fp,11); //Lecture
    $hits++; // Modification
    fseek($fp,0);
    fputs($fp,$hits); //Reecriture
    fclose($fp);
    print("$hits visiteurs<BR />\n");
  ?>
</BODY>
</HTML>
```

En pratique



Inclusion de fichiers (1)

- PHP manipule des fichiers en général
- Mais aussi possibilité de factoriser certaines parties de codes et de contenus
- 2 modes d'inclusion : require et include
 - ▣ insèrent le contenu du fichier dans le fichier courant
 - ▣ identique sauf vis à vis des erreurs :
 - require → arrêt sur erreur
 - include → avertissement sur erreur
 - ▣ versions `_once`

Inclusion de fichiers (2)

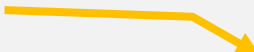
```
<HTML>                                     @php_fonction.php
<HEAD></HEAD>
<BODY>
  <?php
    require("php_lafonction.php");
    print ChgFont("14","red","Texte en rouge de taille moyenne...");
    print ChgFont("24","#0F74A3","Un grand texte...");
  ?>
</BODY>
</HTML>
```

Fichier php_lafonction.php :

```
<?php
  function ChgFont($size,$color,$texte){
    return "<span style=\"font-family:Verdana; font-size:$size;
color:$color;\"> $texte </span>";
  }
?>
```

Inclusion : Pseudo-frames

```
<DIV class="entete">                                     @php_index.php
    <?php include('php_entete.php'); ?>
</DIV><DIV class="colonneGauche">
    <?php include('php_menu.php'); ?>
</DIV><DIV class="colonneDroite">
    <?php
        $pageOK= array('news' => 'php_news.php',
                       'accueil' => 'php_accueil.php');
        include($pageOK[$_GET['page']]);
    ?>
</DIV><DIV class="pied">
    <?php include('php_pied.php'); ?>
</DIV>
```



```
if ((isset($_GET['page'])) &&
    (isset($pageOK[$_GET['page']]))) {
    include($pageOK[$_GET['page']]);
}
```

= Plus sûr !

Transfert de fichier (1)

```
<FORM ACTION="get_file.php" METHOD=POST" ENCTYPE="multipart/form-  
data">  
  <INPUT TYPE="hidden" NAME="MAX_FILE_SIZE" value="100000">  
  Titre : <INPUT TYPE=TEXT SIZE=20 NAME="titre"> <BR />  
  Affiche : <INPUT TYPE=FILE SIZE=20 NAME='affiche'>  
  <H1>Votre choix</H1>  
  <INPUT TYPE=SUBMIT VALUE='Valider'>  
  <INPUT TYPE=RESET VALUE='Annuler'>  
</FORM>
```

Transfert de fichier (2)

```
<?php
    $uploadaddir= '/var/www/uploads/';
    $uploadfile= $uploadaddir.basename($_FILES['affiche']['name']);
    if(move_uploaded_file($_FILES['userfile']['tmp_name'],
    $uploadfile)){
        echo "Le fichier est valide, et a été téléchargé avec succès.\n";
    }else{
        echo "Attaque potentielle par téléchargement de fichiers.\n";
    }
    echo 'Voici quelques informations de débogage :';
    print_r($_FILES);
?>
```


Et ensuite... ?

- Connexion avec base de données
 - ▣ utilisation de fichiers pour ce semestre
- Quelques standards normes complémentaires :
 - ▣ XML, représentation des données
 - ▣ JSON, transfert d'informations
- Meilleure synchronisation entre clients et serveurs :
 - ▣ AJAX
- Notions de base vues dans le cours seront ensuite implémentées dans des "bibliothèques" pour faciliter leur mise en œuvre pratique :
 - ▣ Bibliothèques JS : JQuery, Prototype...
 - ▣ Framework PHP : Zend, Symfony...