Technologie Web
Chapitre 3 : PHP

Cecilia Zanni-Merk

cecilia.zanni-merk@insa-rouen.fr

Bureau BO B R1 04

Contenu

- Introduction à PHP
- Syntaxe
- Fonctions et modularité
- Chaînes de caractères
- Les formulaires
- Les fichiers
- Les exceptions
- Les sessions

Références

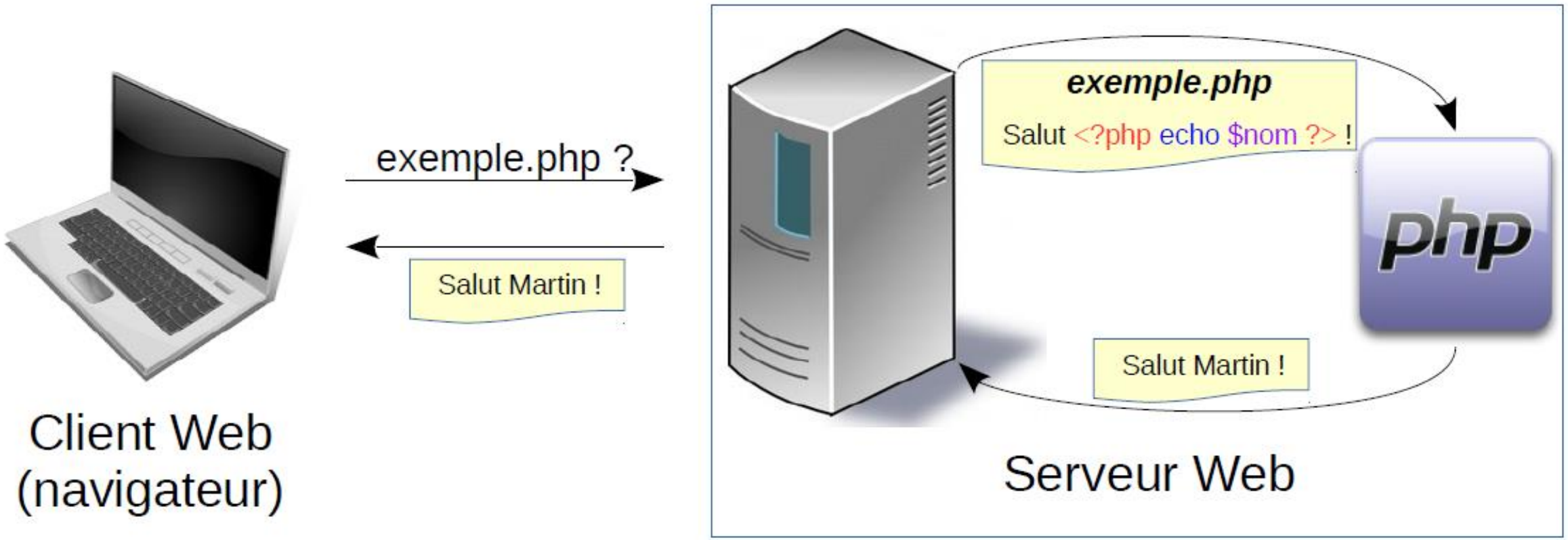
- Cours « Technologies Web » d'Alexandre Pauchet, INSA Rouen Normandie, 2016
- MOOC « Soyez Acteurs du Web », https://www.fun-mooc.fr/

Introduction

Présentation

- PHP (Hypertext Preprocessor)
  - Langage de script intégré à HTML, côté serveur
  - Le serveur *parse* les documents et interprète le code PHP
  - Le client reçoit uniquement le résultat du script (une page “générée”)
  - Le code PHP est inclus dans des balises PHP : <?php code-PHP ?>
  - Peut également fonctionner comme n'importe quel langage interprété de façon locale.
- Fonctionnalités diverses
  - Fonctionnalités équivalentes aux autres langages de scripts CGI
  - Support d’un important nombre de bases de données
  - Nombreuses librairies :
    - gestion des protocoles mail (imap, pop)
    - production de pdf, flash
    - gestion de XML
    - . . .

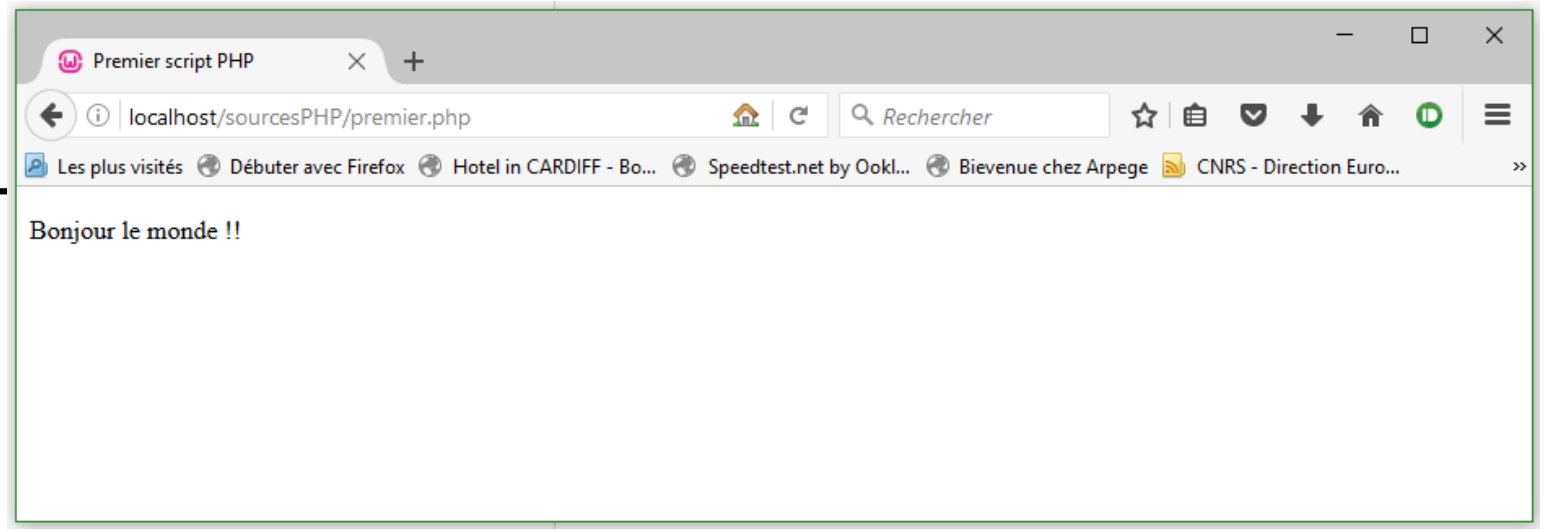
Fonctionnement



Premier script PHP

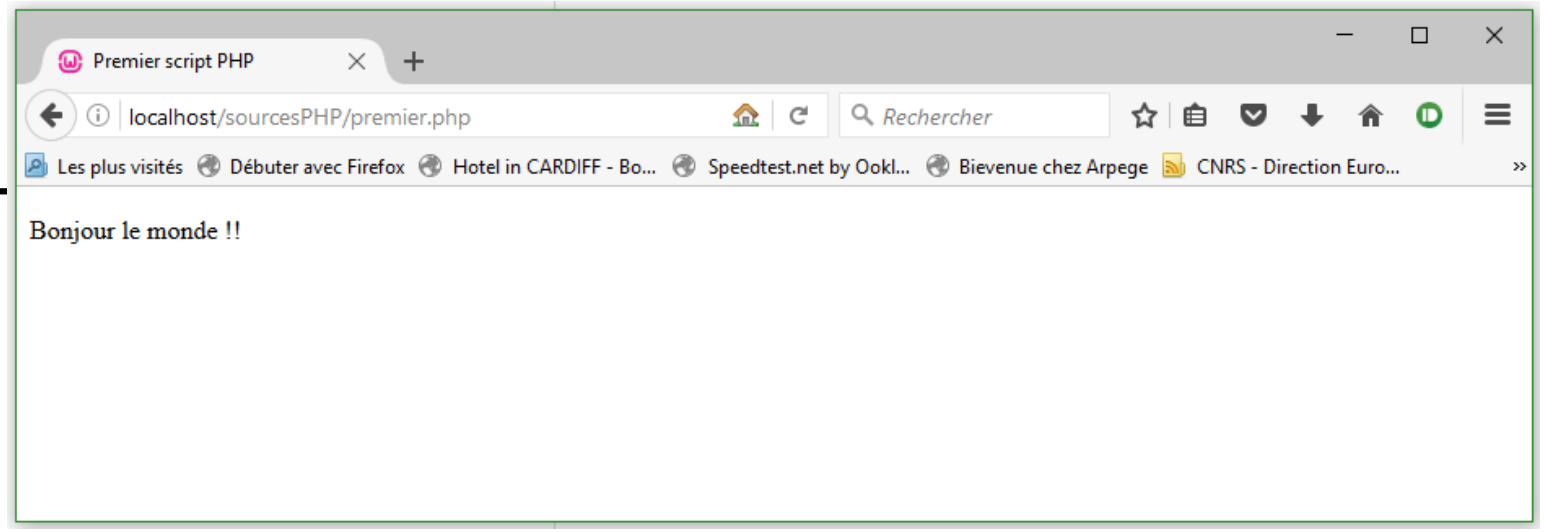
```
<!DOCTYPE html>

<head>
   <title>Premier script PHP</title>
</head>

<body>
   <?php echo "<p> Bonjour le monde !! </p>\n"; ?>
</body>

</html>
```

Premier script PH

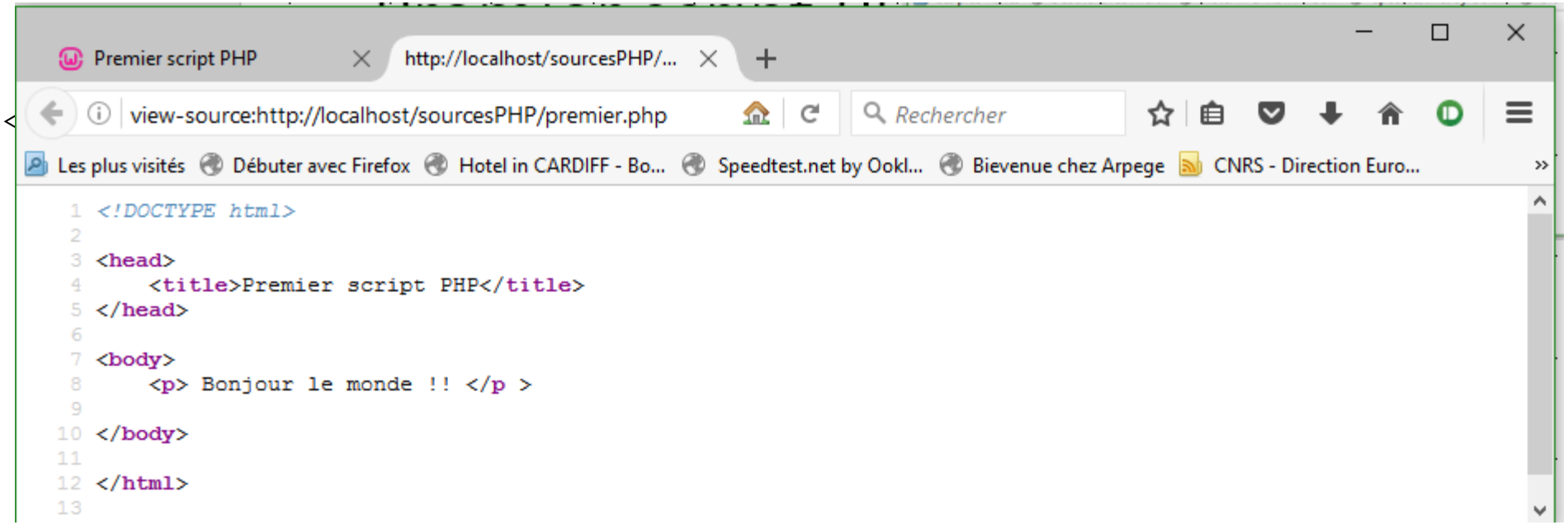



```
<!DOCTYPE html>

<head>
   <title>Premier script PHP</title>
</head>

<body>
   <?php echo "<p> Bonjour le monde !! </p>\n"; ?>
</body>

</html>
```

Premier script PH




```
<!DOCTYPE html>

<head>
    <title>Premier script PHP</title>
</head>

<body>
    <?php echo "<
</body>

</html>
```



Syntaxe

Syntaxe de base

- Insertion d'une commande PHP :

```
<?php code PHP ?>
```

- Séparateur d'instructions : le point virgule " ;"

```
<?php instruction1; instruction2; ... ?>
```

- Commentaires : syntaxe à la C, C++ ou Shell

```
/* ... */
// ...
# ...
```

Les variables

- Le typage des variables est dynamique
- Syntaxe : `$NomDeVariable[=val];`
- règle de nommage : `$[a-zA-Z\_]([a-zA-Z0-9\_])*`
- sensibilité à la casse
- assignation par :
  - Valeur : `$var1=$var2;`
  - référence : `$var1=&$var2;`
- Portée : locale à la fonction où elle est déclarée

Les variables

- Le typage des variables est dynamique
- Syntaxe : `$NomDeVariable[=val];`
- règle de nommage : `$[a-zA-Z\_]([a-zA-Z0-9\_])*`
- sensibilité à la casse
- assignation par :
  - Valeur : `$var1=$var2;`
  - référence : `$var1=&$var2;`
- Portée : locale à la fonction où elle est déclarée

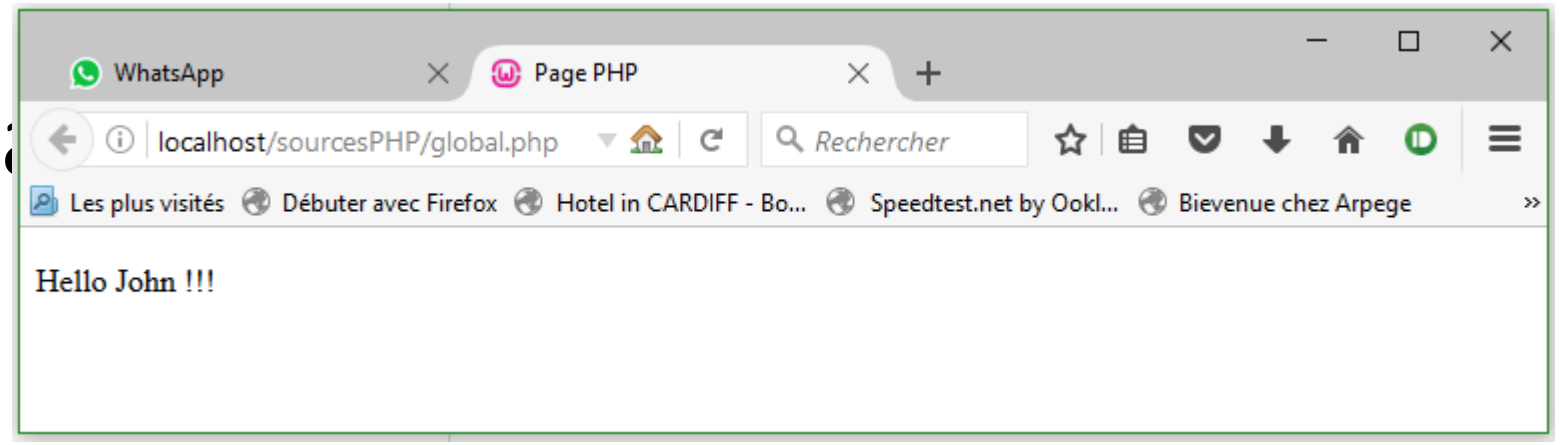
```php
<?php
        $var1 =10;
        $var2 = &$var1 ;
        $var1 =20;
        echo "<p>".$var1." ".$var2." </p>";
?>
```

Affichera …
20 20

Déclaration de variables globales

```
<!DOCTYPE html>
<html >
<head >
   <title >Page PHP </title >
</head >
<body >
<?php
  function affiche () {
     global $var1 ;
     echo "<p> Hello ".$var1." !!! </p >\n";
  }
?>
<?php
      $var1 =" John ";
      affiche ();
?>
</body >
</html >
```

Déclaration de v




```
<!DOCTYPE html>
<html >
<head >
   <title >Page PHP </title >
</head >
<body >
<?php
  function affiche () {
     global $var1 ;
     echo "<p> Hello ".$var1." !!! </p >\n";
  }
?>
<?php
      $var1 =" John ";
      affiche ();
?>
</body >
</html >
```

Variables superglobales

- Les variables superglobales sont utilisables sans le mot clef `global`
- Quelques tableaux superglobaux :
- `$_GLOBAL` contient des références sur les variables de l'environnement d'exécution global (clefs = noms des variables globales)
- `$_SERVER` variables fournies par le serveur web
- `$_GET` et `$_POST` variables transmises par les méthodes `GET` et `POST` du protocole `HTTP`
- Autres : `$_COOKIE`, `$_REQUEST`, `$_SESSION`, `$_ENV`

Les constantes

- Syntaxe : `define("NOM_DE_LA_CONSTANTE", valeur)`
- Les constantes :
  - ne commencent pas par `$`
  - sont définies et accessibles globalement dans tout le code
  - ne peuvent pas être redéfinies
  - ne peuvent contenir que des booléens, des entiers, des flottants et des chaînes de caractères
- Exemple

```
define ("PHP","PHP Hypertext Preprocessor ");
echo PHP;
```

Les types

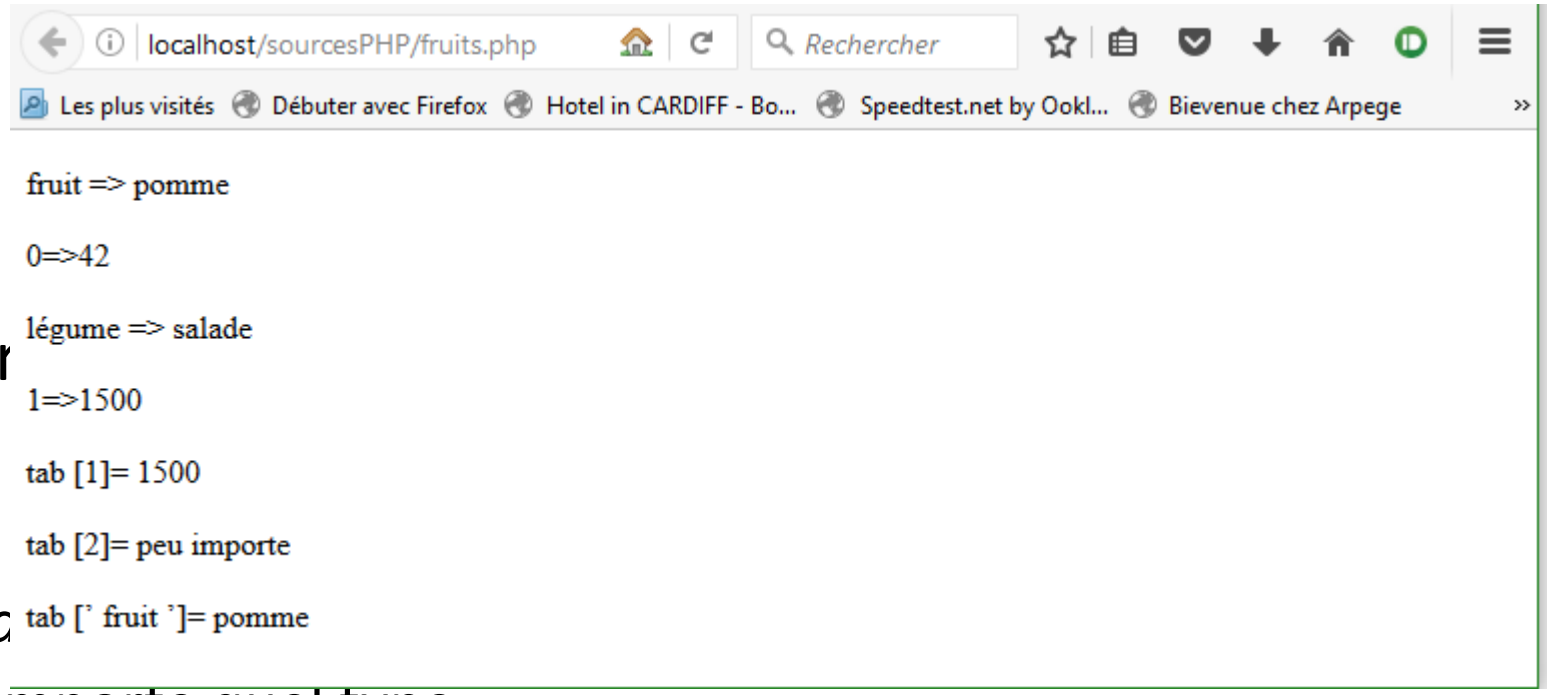
- 4 types simples :
  - entiers : `integer`
  - réels : `float, double`
  - booléens : `boolean` (`TRUE` ou `FALSE`)
  - chaînes de caractères : string
- 2 types complexes :
  - tableaux : `array`
  - objets : `object`
- 2 types spéciaux :
  - Ressources : `resource` (ex : connexion BD)
  - absence de valeur : `null`

Les tableaux

- Principe : associations ordonnées de type clé => valeur
- Déclaration : `array( [clé =>] valeur, ...)`
  - la clé est facultative, elle est de type entier ou chaîne de caractères *; en cas d'omission, la valeur sera associée à la clé d'indice max+1*
  - la valeur peut être de n'importe quel type

Les tableaux

- Principe : associations or
- Déclaration : `array(`
  - la clé est facultative, elle
    d'omission, la valeur serc
  - la valeur peut être de n'importe quel type




```
<?php
       $tab = array (" fruit "=>" pomme " ,42," légume "=>" salade " ,1.5e3);
    foreach ( $tab as $cle => $valeur ) {
      echo "<p>". $cle ."=>". $valeur ." </p>";
    }
    echo "<p> tab [1]= ". $tab [1]. " </p>";
    $tab []= " peu importe ";
    echo "<p> tab [2]= ". $tab [2]. " </p>";
    echo "<p> tab [' fruit ']=". $tab [" fruit "]." </p>";
?>
```

Opérations sur les tableaux

- **Attention, un tableau est toujours une référence !**
- `count($array)` : nombre d'éléments
- `sort($array)` : trie le tableau
- `array_pop($array)` : récupère et supprime le dernier élément d'une liste (i.e. fonctionne comme une pile)
- `array_push($array, $elem1, ...)` : ajoute des éléments à la fin d'une liste (i.e. fonctionne comme une pile)
- `array_shift($array)` : récupère et supprime le premier élément d'une liste
- `array_unshift($array, $elem1, ...)` : ajout d'éléments en début de liste
- `array_merge($array1, $array2, ...)` : fusionne plusieurs tableaux
- `in_array($elem, $array)` : recherche d'un élément dans un tableau
- `array_key_exists($key, $array)` : recherche une clé dans un tableau
- `array_flip($array)` : inverse les clés et les valeurs d'un tableau

Détermination du type d'une variable

- Type d'une variable : `string gettype($var);`
- Test : `is_integer($var);is_double($var);` `is_array($var);`…
- Conversion dynamique : `$result = (type-désiré)$var;`
- Instructions de vérification d'existence (formulaires) :
  - `boolean isset($var);` retourne `FALSE` si `$var` n'est pas initialisée ou a la valeur `NULL`, `TRUE` sinon ;
  - `boolean empty($var);` retourne `TRUE` si `$var` n'est pas initialisée, a la valeur `0`, `"0"`, ou `NULL`, `FALSE` sinon ;
  - `boolean is_null($var);` retourne `TRUE` si `$var` n'est pas initialisée ou vaut `NULL`, `FALSE` sinon.

Exemple

val	gettype()	empty()	is_null()	isSet()	(bool)
$x = "";	string	true	false	true	false
$x = null;	NULL	true	true	false	false
var $x ; (not set)	NULL	true	true	false	false
$x = array();	Array	true	false	true	false
$x = false;	boolean	true	false	true	false
$x = 15;	integer	false	false	true	true
$x = 1;	integer	false	false	true	true
$x = 0;	integer	true	false	true	false
$x = -1;	integer	false	false	true	true !
$x = "15";	string	false	false	true	true
$x = "1";	string	false	false	true	true
$x = "0";	string	true	false	true	false !
$x = "-1";	string	false	false	true	true
$x = "foo";	string	false	false	true	true
$x = "true";	string	false	false	true	true
$x = "false";	string	false	false	true	true !

Opérateurs

- Opérateurs identiques à ceux du C/C++/Java :
  - opérateurs arithmétiques : `+ - * / %`
  - in/décrémentation : `var++ var-- ++var --var`
  - opérateurs logiques : `&& || !`
  - comparaisons : `== != <= >= <>`
  - concaténation de chaînes de caractères : `.`
  - affectation : `= += -= *= . . .`
- Opérateurs spécifiques :
  - ‘commande shell‘ (ex : `$a=‘ls -ul‘`)
  - `===` : teste la valeur et le type

Instructions de décision

- **Proches du C/C++/Java**
- Si-sinon-alors :

```
if( condition ) {
     instructions
}[
     elseif ( condition ) {
          instructions
     }]
     [ else {
          instructions
     }]
```

- Switch-case :

```
switch ( expression ) {
     case ’valeur1 ’:
          Instructions
          break ;
     ...
     default :
          Instructions
          break ;
}
```

Boucles

- **Proches du C/C++/Java**
- Boucles for :

```
for ($i =0; $i <N; $i++) {
      Instructions
}

foreach ( $tab as $val ) {
      Instructions
}

foreach ( $tab as $cle =>$val ) {
      Instructions
}
```

- Boucles while

```
while ( condition ) {
      Instructions
}

do {
      Instructions
} while ( condition );
```

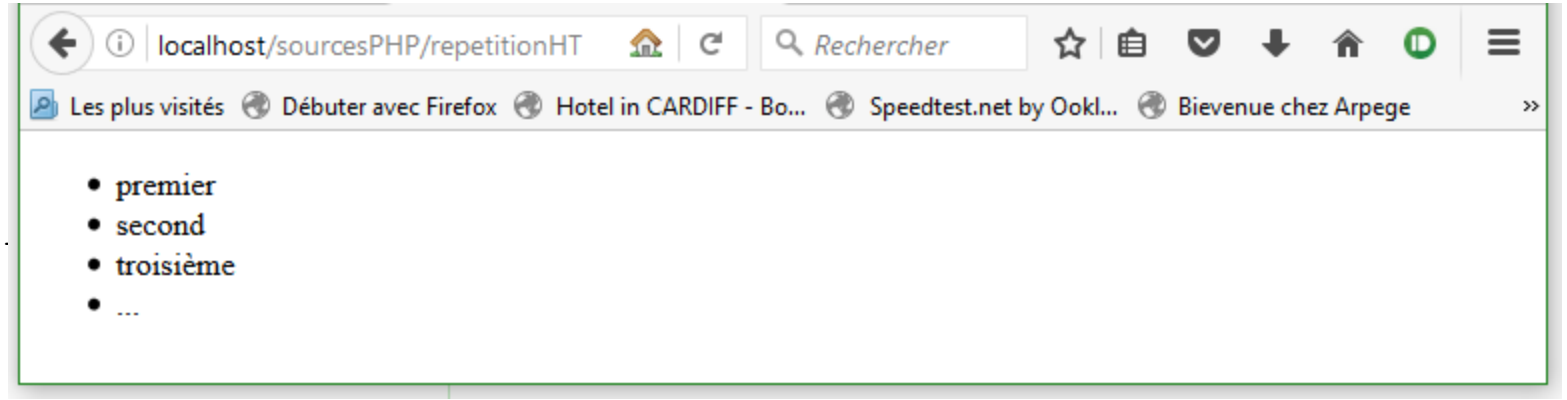
Répétition de code HTML

- Utilisation des boucles pour répéter du code HTML : Entrelacement code PHP / code HTML

```
<!DOCTYPE html>

<html>

<head>
   <title >Répétition HTML </title>
</head>

<body>
   <?php
      $tab = array (" premier "," second "," troisième ","...");
   ?>

   <ul>
       <?php foreach ( $tab as $elem ) { ?>
              <li ><?php echo $elem ; ?></li >
       <?php } ?>
   </ul>
</body>

</html>
```

elacement

```
<!DOCTYPE html>

<html>

<head>
    <title >Répétition HTML </ti
</head>

<body>
    <?php
        $tab = array (" premier "," second "," troisième ","...");
    ?>

    <ul>
        <?php foreach ( $tab as $elem ) { ?>
                <li ><?php echo $elem ; ?></li >
        <?php } ?>
    </ul>
</body>

</html>
```



Fonctions et modularité

Les fonctions

- Syntaxe

```
<?php
function nomDeFonction (arg1 , arg2 , ... , argN ) {
      instructions ;
      [ return VALEUR ;]
  }
?>
```

Les fonctions

- les noms de fonctions sont insensibles à la casse
- une fonction peut être utilisée avant sa définition
- les arguments sont non typés et supportent une valeur par défaut
- la surcharge de fonctions n'est pas supportée
- passage d'arguments par valeur et référence supporté (`&`)
- les fonctions supportent un nombre variable d'arguments
  - Utilisation de ... pour accéder aux arguments variables
- retour d'une unique valeur par la directive `return`

Exemples

- Télécharger et tester les fichiers
  - `fonctionsMath.php`
  - `exemplesFonctions.php`
  - `nbrArgsVar.php`

Inclusion de code externe

```
<!DOCTYPE html>

<html>

<head>
   <title>Page PHP </title>
</head>

<body>
   <?php include_once ("Entete.inc.php"); ?>
   <p>contenu normal de la page </p>
   <p align ="right">
   <i> <?php include_once ("Pieddepage.inc.php"); ?> </i>
      </p>
</ body >

</ html >
```

Conventions de nommage des fichiers

- Attention : Les fichiers dont l'extension n'est pas .php ne sont pas parsés, et donc pas directement lisibles par une requête HTTP
  - Bonne pratique : .inc.php
  - Bonne pratique : .conf.php
  - Bonne pratique : .class.php
- Mauvaise pratique (par exemple) : .inc

Les chaînes de caractères

Déclaration

- Les chaînes peuvent être déclarées avec :
  - Simples quotes : `$t='texte';`
  - Doubles quotes : `$t="texte";`
- Fonctionnement différent : entre doubles quotes, les variables et les caractères échappatoires sont interprétés
- Exemples pour `$t = "Mot";`
  - `"Texte"` → `Texte`
  - `"Texte $t"` → `Texte Mot`
  - `'Texte'` → `Texte`
  - `'Texte $t'` → `Texte $t`

Opérations sur les chaînes de caractères

- Longueur : `int strlen(string $ch)`
- Répétition : `string str_repeat(string $cr, int nb)`
- Minuscules : `string strtolower(string $ch)`
- Majuscules : `string strtoupper(string $ch)`
- Initiales en majuscules : `string ucwords(string $ch)`
- 1ere lettre en majuscule : `string ucfirst(string $ch)`
- Suppression de caractères en début de chaîne : `string ltrim(string $ch, string liste)`
- Suppression de caractères en fin de chaîne : `string rtrim(string $ch, string liste)`
- Suppression de caractères en début et fin de chaîne : `string trim(string $ch, string liste)`

Sous-chaînes de caractères

- Recherche sensible à la casse (retourne tous les caractères de `$ch` depuis la 1ere occurrence de `$ch2` jusqu'à la fin) :
  `string strstr(string $ch, string $ch2)`
- Recherche insensible à la casse :
  `string stristr(string $ch, string $ch2)`
- Extraction de chaînes de caractères :
  `string substr(string $chr, int indice, int N)`
- Décompte du nombre d'occurrences d'une sous-chaîne :
  `int substr_count(string $ch, string $ssch)`
- Remplacement : `string str_replace(string $oldssch, string $newssch, string $ch)`
- Position : `int strpos(string $ch, string $ssch)`

Exemples

- Téléchargez et testez les scripts
  - `traitementString.php`
  - `traitementString2.php`

Les expressions régulières (aka) RegEx

- Permettent de définir un motif de caractères, représentatif d'un ensemble de chaînes de caractères.
  - Caractère(s) : `""` ou `‘ ’` (ex : `"a"`, `"ab"`)
  - Caractères spéciaux : `\.`, `\$`, `\^ `, `\?`, `\\`, `\[`, `\]`, `\(`,`\)`,`\+` et `\*`
  - Classe de caractères : `[]` (ex : `[xyz]`, `[a-z]`)
  - Classes de caractères prédéfinies :
    - `[[:alnum:]]` : caractères alphanumériques
    - `[[:alpha:]]` : caractères alphabétiques
    - `[[:ctrl:]]` : caractères de contrôle
    - `[[:digit:]]` : chiffres
    - `[[:punct:]]` : caractères de punctuation
    - `[[:space:]]` : caractères d'espaces
    - `[[:upper:]]` : majuscules

RegEx : modèles généraux

- N’importe quel caractère : `.`
- 0 ou 1 fois : `?` (ex : `"https?"`)
- Au moins une fois : `+`
- 0, 1 ou plusieurs fois : `*` (ex : `"mat.*"`)
- Exactement n fois : `"{n}"`
- Au moins n fois : `"{n,}"`
- Entre n et m fois : `"{n,m}"`
- Groupements : () (ex : `"(ma)*"`)
- Alternative : `|` (ex : `"(\.net)|(\.com)"`)

RegEx : modèles généraux

- N'importe quel caractère : `.`
- 0 ou 1 fois : `?` (ex : `"https?"`)
- Au moins une fois : `+`
- 0, 1 ou plusieurs fois : `*` (ex : `"mat.*"`)
- Exactement n fois : `"{n}"`
- Au moins n fois : `"{n,}"`
- Entre n et m fois : `"{n,m}"`
- Groupements : () (ex : `"(ma)*"`)
- Alternative : `|` (ex : `"(\.net)|(\.com)"`)

```
"[[:digit:]]{2}/[[:digit:]]{2}/[[:digit:]]{4}"
"[[:alnum:]]*\.[[:alnum:]]*@asi\.insa-rouen\.fr
```

RegEx : recherche et remplacement

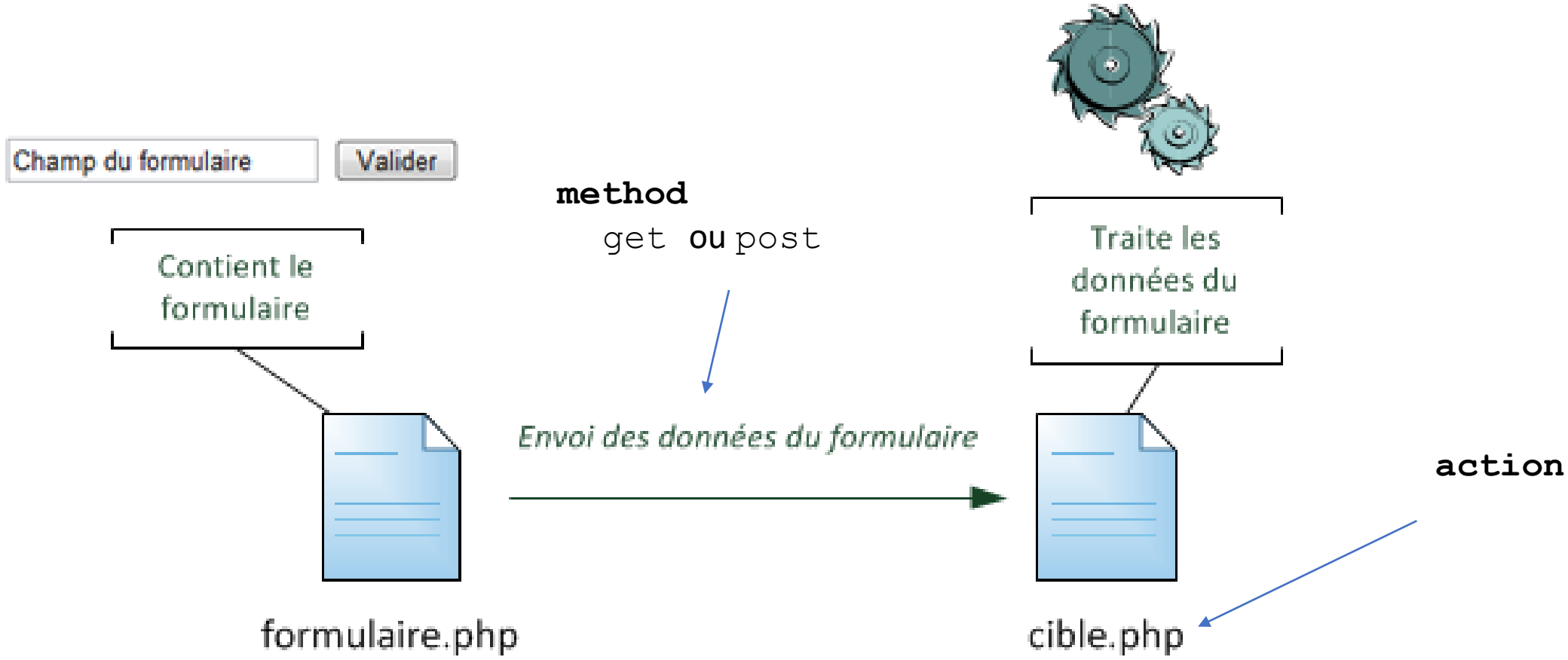
- Fonctions de recherche
  - `int preg_match ( string $modeleregex , string $chaine [, array & $matches ] )`
- Fonctions de remplacement
  - `mixed preg_replace ( mixed $modeleregex , mixed $replacement , mixed $chaine )`
- Remarque
  - Le modèle d'une expression rationnelle est déclarée entre `/ /`, à l'intérieur d'une chaîne de caractères.
  - Ex : `"/ftp:\/\/.*:.*/"`

Exemple

- Téléchargez et testez le script `expRegulieres.php`

Les formulaires

Rappel


Champ du formulaire
Valider
Contient le formulaire
method
get ou post
Traite les données du formulaire
Envoi des données du formulaire
action
formulaire.php
cible.php

GET/POST

- Les champs d'un formulaire sont disponibles à travers les variables superglobales `$_GET` et `$_POST`
- Remarques :
  - Si le submit est une image, les coordonnées du click sont transmises via `$sub_x` et `$sub_y`
  - Il est possible d'utiliser des tableaux à une dimension pour des formulaires ayant par exemple des listes à choix multiples
  - La balise HTML `<pre>...</pre>` peut être utilisée pour interpréter les espaces, tabulations et sauts de ligne.

Dangers des formulaires

- Nous ne pouvons pas supposer que nous allons recevoir ce que nous attendons dans $_POST
- En plus, n'importe qui peut récupérer le code HTML de mon formulaire, le modifier, le publier ailleurs et toujours pointer son formulaire malveillant vers mon script de traitement de données, avec les dangers conséquents !!

NEVER TRUST USER INPUT

- Mon script de traitement doit valider toutes les entrées

La faille XSS

- XSS : cross-site scripting
  - Technique qui consiste à injecter du code HTML contenant du JavaScript dans une page pour le faire exécuter aux visiteurs.
- Dans les champs texte saisis par un utilisateur, il pourrait inclure du code HTML, qui s'exécuterait du côté client
  - En particulier avec des balises `<script>` qui pourraient faire exécuter du code Javascript !! ← dangereux !!
- Il faut « échapper » le code HTML et afficher les balises, à la place de les exécuter
  - Utiliser `htmlentities` ou `htmlspecialchars`

Exemple

- Téléchargez et testez les scripts
  - Formulaire.html
  - traiterForm.php

Références

Webographie

- http://www.php.net/