

Développement serveur en PHP

IO2

Internet et outils

Matej Stehlik

IRIF, Université de Paris

matej@irif.fr

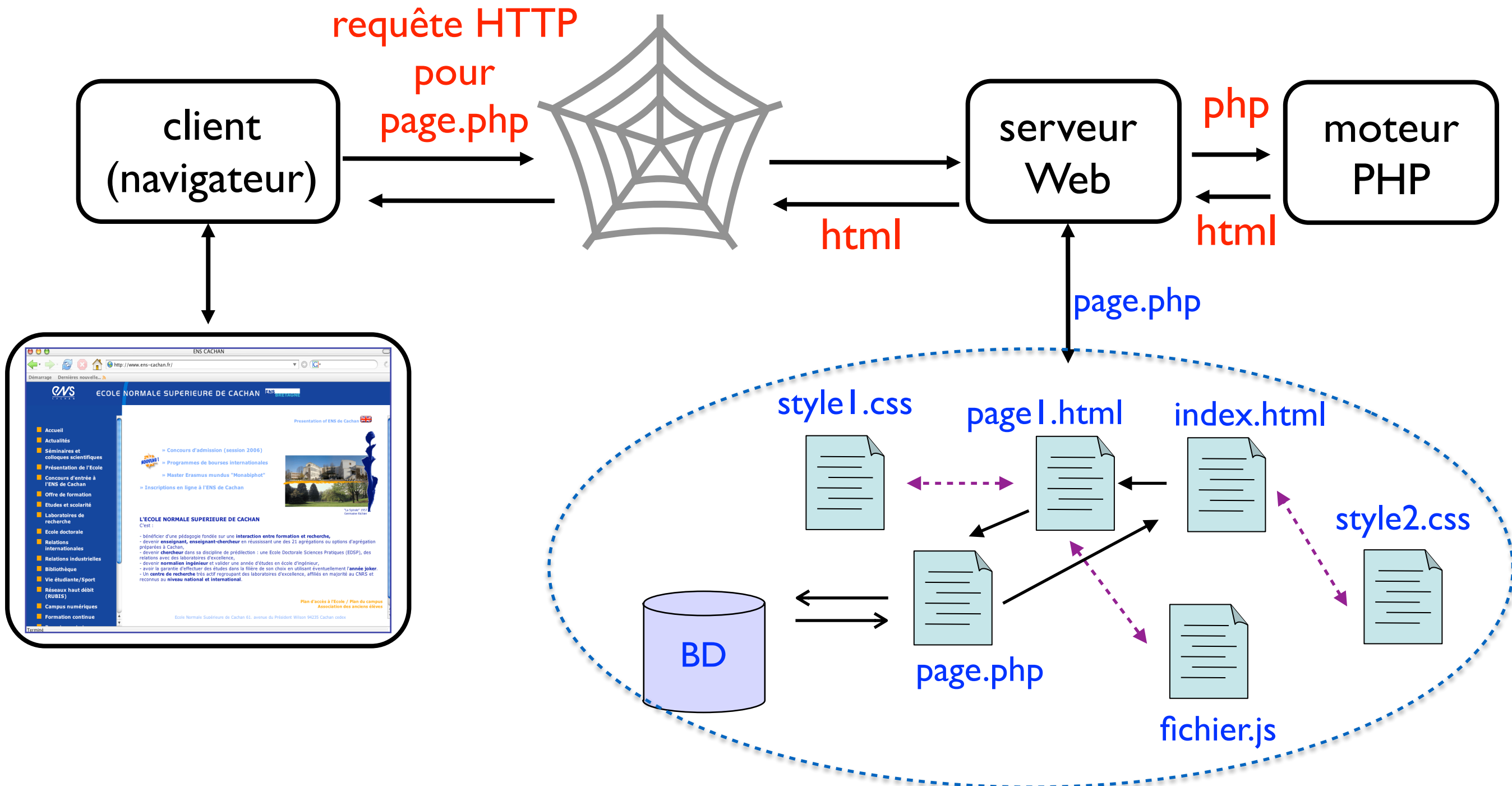
Rappel : structure de base d'une application Web

- Un ensemble de programmes écrits dans un langage de script (e.g. **PHP**) exécutables par le serveur Web, qui implémentent la **logique de l'application** et génèrent des documents HTML dynamiques (**HTML dynamique coté serveur**)
- Eventuellement un ensemble de **documents HTML** statiques
- HTML statique et dynamique, potentiellement enrichi par
 - des **feuilles de style (CSS)** pour définir le style des pages
 - des programmes écrits dans un langage (typiquement **Javascript**) exécutables par le navigateur, qui s'occupent de dynamiser le contenu de la page Web une fois chargée par le navigateur (**HTML dynamique coté client**)
- Possiblement une **base de données** pour le stockage des données d'intérêt de l'application
- Tous ces documents et données : accessibles par un **serveur Web**

PHP est une des langages les plus utilisés pour le développement d'applications Web
(du **PHP pur** au *frameworks*)

Exécution du code coté serveur

- Pour réaliser la logique de l'application, du code est exécuté coté serveur afin de générer les ressources demandées
- Le serveur Web s'appuie sur un moteur de script pour l'interprétation de ce code



PHP Hypertext Preprocessor

- Langage de programmation à la syntaxe proche du Java
- Langage de script
 - ▶ S'utilise et s'exécute côté serveur pour produire un document HTML à renvoyer au client
 - ▶ Imbriqué avec le document HTML
 - ▶ de nombreux modules d'interface avec d'autres outils dont notamment les serveurs de gestion de bases de données (e.g. MySQL)
- Extension des fichiers : `.php`

Script PHP

- Un script PHP est un document incluant des fragments de HTML et des blocs d'instructions PHP

```
<!DOCTYPE html>  
<html>  
<head> <title> Un script PHP </title></head>  
<body>
```

Blocs PHP
délimités par les
pseudo-balises
<?php et ?>

```
<?php ... ..  
... ..  
?>
```

```
<p> un paragraphe </p>
```

```
<?php... ..  
?>
```

```
</body>  
</html>
```

Script PHP

- Seulement les blocs PHP sont exécutés et ils renvoient du HTML

Page HTML
générée par
le script et
renvoyée au
client



```
<!DOCTYPE html>
```

```
<html>
```

```
<head> <title> Un script PHP </title></head>
```

```
<body>
```

HTML

```
<p> un paragraphe </p>
```

HTML

```
</body>
```

```
</html>
```

Output PHP

- Instruction PHP de base pour produire une chaîne de caractères dans de document :

```
echo (“...”);
```

- ou avec plusieurs arguments :

```
echo (“...”, “...”, “...”);
```

- parenthèses optionnelles
- La chaîne de caractères peut comporter :
 - ▶ des balises HTML
 - ▶ des variables PHP qui seront remplacées par leur valeur

Exemple : fichier côté serveur

```
<html>
<head><title>Essai de PHP</title></head>
<body>
<hr>
Nous sommes le <?php echo date ("j / m / Y"); ?>
<hr>
</body>
</html>
```

`date()` est une fonction PHP (prédéfinie)

Exemple : fichier reçu côté client

```
<html>
```

```
<head><title>Essai de PHP</title></head>
```

```
<body>
```

```
<hr>
```

```
Nous sommes le 12 / 02 / 2021
```

```
<hr>
```

```
</body>
```

```
</html>
```

Deuxième exemple

```
<body>
```

```
<h1>Un essai de PHP</h1>
```

```
<?php
```

```
echo "Client : ", $_SERVER["HTTP_USER_AGENT"];
```

```
echo "<br><br>Serveur : ";
```

```
echo $_SERVER["SERVER_NAME"], "<br>";
```

```
?> ...
```

- 3 commandes “echo” séparées par des ;
- Arguments de “echo” séparés par des ,
- **\$_SERVER** : une variable PHP prédéfinie (un tableau)
 - contient des information sur le serveur, le client qui demande la page, la requête HTTP, etc.
 - les entrées de ce tableau sont créées par le serveur web et sont disponibles dans tous les scripts (plus de détails plus loin...)

Deuxième exemple (suite)

```
<body>
```

```
<h1>Un essai de PHP</h1>
```

```
<?php
```

```
echo "Client : ", $_SERVER["HTTP_USER_AGENT"];
```

```
echo "<br><br>Serveur : ";
```

```
echo $_SERVER["SERVER_NAME"], "<br>";
```

```
?>
```

```
<h3>J'ai réussi mon premier programme PHP
```

```
<?php echo "en date du : ", date("d / m / y"), " !" ?>
```

```
</h3>
```

```
</body>
```

On peut insérer plusieurs séquences PHP dans un fichier

Chaînes entre guillemets : "..."

Code reçu par le navigateur

```
<html>
<head><title>Deuxième exemple de PHP</title></head>
<body>
<h1>Un essai de PHP</h1>
Client : Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/
537.36 (KHTML, like Gecko) Chrome/88.0.4324.150 Safari/537.36
<br /><br />Serveur : localhost<br />
<h3>J'ai réussi mon premier programme PHP
  en date du : 12 / 02 / 19 !</h3>
</body></html>
```

En rouge : les parties générées par le code PHP

Rôle du serveur

- Lorsqu'un document HTML est demandé par le client, le serveur se “contente” de trouver le fichier contenant le document correspondant à l'URL et de le renvoyer
- Lorsqu'un script PHP est demandé, le serveur doit d'abord analyser les documents pour :
 - ▶ trouver toutes les occurrences de `<?php... ?>`
 - ▶ faire interpréter les commandes PHP par un interpréteur PHP
 - ▶ substituer les chaînes `<?php... ?>` par le résultat de l'interprétation

PHP

Éléments du langage

Aide, manuel PHP

- Le monde PHP est (très) vaste et (très) riche
- Ce cours n'a pour but que de vous donner quelques bases,
- Si vous cherchez une réponse à une question,
- Si vous cherchez une fonction, une syntaxe,
- Consultez :
 - ▶ Site officiel : <http://www.php.net/>
 - ▶ Pages manuels en français : <https://www.php.net/manual/fr/>
 - ▶ <http://fr.wikipedia.org/wiki/PHP>

Variables

- Noms de variables :
 - ▶ Commencent par \$
 - ▶ Longueur quelconque
 - ▶ Composés de lettres, de chiffres, et _
 - ▶ Lettre ou _ après le \$
 - ▶ Sensibles à la casse (majuscules et minuscules ne sont pas identiques)
- Exemples
 - ▶ \$ma_var, \$Ma_Var, \$mon_1er_nom, \$_une_autre

Variables

- Créer des variables (pas de déclaration de type!) :

```
$note = 19;
```

```
$coeff = 1.53;
```

```
$nom = "Roger Rabbit";
```

```
echo($note);
```

```
echo " Note : ", $note, " Nom : ", $nom,  
    " c : ", $coeff;
```

- Une variable pas définie (`$coeff`) n'a pas de valeur (pas affichée par `echo`)
- Selon les réglages de l'interpréteur PHP, l'utilisation d'une variable non-définie peut générer un message d'alerte :

Notice: Undefined variable: coeff in xxx.php on line 12

Variables

- Supprimer des variables:
 - `unset($note);`
 - `unset($note, $coeff, $nom);`
- En PHP une variable peut être :
 - définie avec une valeur : `$var = "toto";`
 - définie et vide : `$var1 = "";` `$var2 = 0;` `$var3 = false;` ...
 - non définie : `$var;` *variable sans valeur*
- Vérifier le contenu d'une variable :
 - `isset($var)` // Vrai si définie, même vide
 - `empty($var)` // Vrai si vide, ou pas définie

Variables : Exemple

```
$note = 19;
```

```
$coeff;
```

```
$nom = "";
```

```
echo
```

```
isset($note), // vrai
```

```
isset($nom), //vrai
```

```
isset($newvar), //faux
```

```
isset($coeff), //faux
```

```
empty($coeff), // vrai
```

```
empty($nom), //vrai
```

```
empty($newvar); //vrai
```

Types de données

- Il existe plusieurs types de base:
 - ▶ Entier (integer) : `$note = 1728;`
 - ▶ Nombre à virgule flottante (float ou double) : `$temp = 37.6;`
 - ▶ Chaîne de caractères (string) : `$nom = "personne";`
 - ▶ Booléen (boolean) : `$condition = TRUE;` `$condition = FALSE;`
- ainsi que des types composés : Tableaux (`array`), Objet (`object`), ...
- Pas de déclaration de type : le type d'une variable est déterminé par son "contenu"
- La même variable peut prendre des types différents au cours de son cycle de vie :

```
$note = 17;    // $note est de type entier
```

```
$note = "très bien"; // $note est maintenant de type string
```

Vérifier le type d'une variable

`is_int($var)` // Vrai si type entier

`is_float($var)`

`is_string($str)`

...

`gettype($var)` renvoie le type de la variable `$var`

Conversion de type

```
$entier = (integer)$var2;
```

```
$chaine = (string)$var3;
```

```
$flottant = (float)$var4;
```

Attention, on ne peut pas convertir n'importe quelle chaîne en entier (0 si impossible)

```
$entier = (int)"1597"; // OK, $entier = 1597
```

```
$entier = (int)"mille deux cents"; // non! $entier = 0
```

```
$entier = (int)"1597 mille deux cents"; // OK!
```

Exemple

<?php

```
$entier1 = 4; $entier2 = 44;  
$chaine = "2021 une bonne année!";  
$string = "une bonne année 2021";  
  
$entier1 = $chaine; $entier2 = $string;  
  
echo "entier1 : ", $entier1, "<br />",  
     "entier2 : ", $entier2, "<br />";  
  
$entier1 = (int)$chaine; $entier2 = (int)$string;  
  
echo "entier1 : ", $entier1, "<br />",  
     "entier2 : ", $entier2, "<br />";
```

?>

Constantes

```
define("nom_cste", valeur_cste);
```

Ne doivent pas correspondre aux mots-clés de PHP

Exemple :

```
define("USD_TO_EUR", 0.929);  
echo "5 dollars valent : ", 5 * USD_TO_EUR,  
    "euros";
```

⇒ 5 dollars valent : 4.64 euros

Commentaires

- Insérer des commentaires en PHP:

```
/* ceci est un  
commentaire */
```

```
// commentaire jusqu'à la fin de la ligne
```

```
# Ceci également
```

Calculs arithmétiques

Opérateurs :

+ - * / %

\$cpt++; // équivalent à \$cpt = \$cpt + 1;

\$cpt--; // équivalent à \$cpt = \$cpt - 1;

\$cpt /= 2; \$cpt += 77; \$cpt -= 2; \$cpt *= 5;

Quelques fonctions :

sqrt(64) // => 8 racine carrée

ceil(27.68) // => 28 entier immédiatement supérieur

floor(27.9) // => 27 entier immédiatement inférieur

round(24.62) // => 25

Opérateurs et types

- Attention : certains opérateurs peuvent changer automatiquement le type d'une variable :

```
$cpt = 7; echo gettype($cpt), "<br>";
```

```
$cpt /= 2; echo gettype ($cpt), "<br>";
```

```
$cpt = 8; echo gettype($cpt), "<br>";
```

```
$cpt /= 2; echo gettype ($cpt), "<br>";
```

```
$str = "10";echo gettype ($str), "<br>";
```

```
$str += 5; echo gettype ($str), "<br>";
```

- Remarque : / sur un entier n'est pas la division entière !

Chaînes de caractères

- Deux formes : `'...'` et `"..."`
- Différence : `"..."` admet interpolation des variables
 - ▶ une variable présente dans `"..."` sera remplacée par sa valeur

```
$var = "jean"
```

```
echo "Mon nom est $var <br>"
```

⇒ Mon nom est jean

```
echo 'Mon nom est $var <br>'
```

⇒ Mon nom est \$var

- Les guillemets doubles doivent être échappés dans `"..."` mais pas dans `'...'`
- Les guillemets simples doivent être échappés dans `'...'` mais pas dans `"..."`

```
$str = "Cette chaîne n'a qu'un \" guillemet double";
```

```
$str = 'Cette chaîne "s\'écrit" avec une apostrophe';
```

Chaînes de caractères

- Concaténer des chaînes :

```
$str1 = "Hello";
```

```
$str2 = "World!";
```

```
$salut = $str1 . " " . $str2;
```

```
echo $salut;
```

⇒ Hello World!

- Taille d'une chaîne de caractères (nombre de caractères): `strlen()`

```
echo strlen("Hello world!");
```

⇒ 12

- php offre plusieurs fonctions natives pour travailler avec les chaînes de caractères (cf. plus loin)

Tableaux (array)

- Stockent plusieurs valeurs dans une seule variable
- Les tableaux php peuvent stocker des valeurs de différents types
- Deux formes :
 - ▶ indicés : valeurs associés à des indices 0, 1, 2...
 - ▶ associatifs : valeurs associées à des “clefs” (chaînes de caractères), comme une table de hachage
- Si `$t` est un tableau, `count($t)` renvoie le nombre d'éléments de `$t`
- Comme toute autre variable en PHP, un tableau est créé par affectation de son contenu

Tableaux : création

Créer un tableau indicé :

```
$films[0] = "Casablanca";
```

```
$films[1] = "To Have and Have Not";
```

Pas d'obligation de commencer à 0

Créer un tableau associatif :

```
$capitale["FR"] = "Paris";
```

```
$capitale["UK"] = "Londres";
```

```
$capitale["IT"] = "Rome";
```

Indices et clefs peuvent coexister dans le même tableau :

```
$rues[4] = "Rue Lecourbe"
```

```
$rues["4bis"] = "Rue de la Paix"
```

Tableaux : création

Créer sans indice explicite :

```
$livres[] = "Les Misérables";
```

```
$livres[] = "Notre Dame de Paris";
```

```
$livres[] = "Quatre-vingt treize";
```

```
echo $livres[0] ⇒ Les Misérables
```

```
echo $livres[1] ⇒ Notre Dame de Paris
```

```
echo $livres[2] ⇒ Quatre-vingt treize
```


Tableaux : création

Avec le constructeur `array` (en une seule fois) :

- ▶ Tableau indicé :

```
$livres = array("Les Misérables",  
               "Notre Dame de Paris",  
               "Quatre-vingt treize");
```

```
echo $livres[0] ⇒ Les Misérables
```

- ▶ Tableau associatif :

```
$capitales = array("FR"=> "Paris", "UK"=> "Londre", "IT"=> "Rome");
```

- ▶ Tableau mixte :

```
$rues = array(4 => "Rue Lecourbe", "4bis" => "Rue de la Paix");
```

- ▶ Indices implicites :

```
$rues = array(4 => "Rue Lecourbe", "Rue de la Paix");
```

```
echo $rues[5] ⇒ Rue de la Paix
```

```
$rues = array("4bis" => "Rue de la Paix", "Rue Lecourbe");
```

```
echo $rues[0] ⇒ Rue Lecourbe
```

Tableaux : création

Intervalles :

```
$annees = range(1970, 2038);
```

On peut aussi créer des tableaux multi-dimensionnels (tableaux de tableaux)

```
$personnes = array  
(  
    array("nom"=>"Dupont", "prenom" => "Jean"),  
    array("nom"=>"Portier", "prenom" => "Louis", "age" => 39)  
);
```

```
echo $personnes[1]["nom"]
```

⇒ Portier

Tableaux : création

Intervalles :

```
$annees = range(1970, 2038);
```

On peut aussi créer des tableaux multi-dimensionnels (tableaux de tableaux)

```
$notes = array  
(  
    "Dupont" => array("Exo1" => 12, "Exo2" => 14),  
    "Portier" => array("Exo1"=>16, "Exo2" => 18, "Exo3" => 15)  
);  
  
echo $notes["Portier"]["Exo3"];
```

⇒ 15

Tableaux : création

Création de tableaux multi-dimensionnels par affectation (même effet):

```
$personne[0]["nom"] = "Jean";
```

```
$personne[0]["prenom"] = "Dupont";
```

```
$personne[1]["nom"] = "Louis";
```

```
$personne[1]["prenom"] = "Portier";
```

```
$personne[1]["age"] = 39;
```

```
$notes["Dupont"]["Exo1"] = 12; $notes["Dupont"]["Exo2"] = 14;
```

```
$notes["Portier"]["Exo1"] = 16; $notes["Portier"]["Exo2"] = 18;
```

```
$notes["Portier"]["Exo3"] = 15;
```

Tableaux : impression

```
print_r($livres)
```

```
⇒ Array ( [0] => Les Misérables  
          [1] => Notre Dame de Paris  
          [2] => Quatre-vingt treize )
```

```
var_dump($livres)
```

```
⇒ array(3) { [0]=> string(14) "Les Misérables"  
            [1]=> string(19) "Notre Dame de Paris"  
            [2]=> string(19) "Quatre-vingt treize" }
```

Tableaux : modification

- Modification d'une valeur :

```
$livres[1] = "";
```

```
print_r($livres);
```

```
⇒ Array ( [0] => Les Misérables  
          [1] =>  
          [2] => Quatre-vingt treize )
```

- Suppression d'une case :

```
unset($livres[1]);
```

```
print_r($livres);
```

```
⇒ Array ( [0] => Les Misérables  
          [2] => Quatre-vingt treize )
```

Tableaux : parcours

Une variante de la boucle `for` permet de parcourir aisément les tableaux :

```
foreach($array as $var) {  
    instructions  
}
```

- une itération pour chaque case du tableau
- à l'itération `i`, `$var` prend la valeur de la `i`-ème case du tableau `$array`
- s'applique aussi bien aux tableaux indicés qu'aux tableaux associatifs

Tableaux : parcours

- Exemple avec tableau indicé :

```
$livres = array("Les Misérables",  
               "Notre Dame de Paris",  
               "Quatre-vingt treize");
```

```
foreach($livres as $titre) {  
    echo $titre." ";  
};
```

⇒ Les Misérables Notre Dame de Paris Quatre-vingt treize

- Exemple avec tableau associatif :

```
$capitales = array('FR'=> "Paris", 'UK'=> "Londres", 'IT'=> "Rome");
```

```
foreach($capitales as $ville) {  
    echo $ville." ";  
}
```

⇒ Paris Londres Rome

Tableaux : parcours

Pour récupérer à chaque itération à la fois la valeur et l'indice/clef de la case :

```
foreach($array as $index => $var) {  
    instructions  
}
```

- Exemple avec tableau indicé :

```
$livres = array("Les Misérables",  
               "Notre Dame de Paris",  
               "Quatre-vingt treize");  
  
foreach($livres as $index => $titre) {  
    echo "Livre #", $index, " : $titre<br/>";  
}
```

```
⇒ Livre #0 : Les Misérables  
   Livre #1 : Notre Dame de Paris  
   Livre #2 : Quatre-vingt treize
```

Tableaux : parcours

Pour récupérer à chaque itération à la fois la valeur et l'indice/clef de la case :

```
foreach($array as $index => $var) {  
    instructions  
}
```

- Exemple avec tableau associatif :

```
$capitales = array('FR'=> "Paris", 'UK'=> "Londre", 'IT'=> "Rome");  
  
foreach($capitales as $pays => $ville) {  
    echo "capitale de ", $pays, " : $ville<br/>";  
}
```

```
⇒ capitale de FR : Paris  
   capitale de UK: Londres  
   capitale de IT : Rome
```

Tableaux : parcours

Exercice.

Donné un tableau contenant un nombre arbitraire de lignes de la forme :

```
$notes = array  
(  
    "Dupont" => array("Exo1" => 12, "Exo2" => 14),  
    "Portier" => array("Exo1"=>16, "Exo2" => 18, "Exo3" => 15)  
);
```

Calculer un tableau associatif contenant pour chaque étudiant, la moyenne de ses notes ;

l'afficher d'abord avec `print_r()`; ensuite comme un tableau HTML

Tableaux : parcours

Par manipulation d'un pointeur interne au tableau :

```
current($array); // renvoie la valeur sous le pointeur
```

```
next($array); previous($array);
```

```
//avance/recule le pointeur et renvoie la valeur sous le pointeur  
après le déplacement. Renvoie false s'il n'est pas possible  
d'avancer/reculer
```

```
reset($array); end($array);
```

```
//positionne le pointeur sur le premier/dernier élément du tableau
```

Exemple :

```
$livres = array("Les Misérables", "Notre Dame de Paris",  
                "Quatre-vingt treize");
```

```
reset($livres);
```

```
echo current($livres); // ⇒ Les Misérables
```

```
echo next($livres); // ⇒ Notre Dame de Paris
```

```
echo next($livres); // ⇒ Quatre-vingt treize
```

Tableaux prédéfinis (superglobals)

Tout script PHP a accès à des tableaux associatifs prédéfinis, rendus disponibles par le serveur Web, dont voici quelques uns :

`$_SERVER`

Informations relatives au serveur et à la connexion

Exemple :

```
$_SERVER["HTTP_USER_AGENT"]; $_SERVER["SERVER_NAME"]  
$_SERVER["SCRIPT_NAME"] //le chemin du script courant
```

Tableaux prédéfinis (superglobals)

Tout script PHP a accès à des tableaux associatifs prédéfinis, rendus disponibles par le serveur Web, dont voici quelques uns :

`$_GET`

les paramètres passées au script dans l'url :

`http://host/chemin/page.php?clef1=val1&clef2=val2`

Dans page.php :

`$_GET['clef1']` // a valeur 'val1'

`$_GET['clef2']` // a valeur 'val2'

Tableaux prédéfinis : exemple I

Affiche “Bonjour nom prénom !”

```
<body>
```

```
<h2> Bonjour
```

```
<?php
```

```
echo $_GET['prenom'].' '. $_GET['nom'].' !';
```

```
?>
```

```
</h2>
```

```
</body>
```

<http://localhost/.../04-get.php?prenom=Jean&nom=Dupont>

Tableaux prédéfinis : exemple 2

```
<head>
<style>
body{
background-color:
<?php
echo $_GET['col'];
?>
}
</style>
</head>
```

```
<body>
<h2> Bonjour
<?php
echo $_GET['prenom'].' '. $_GET['nom'].' !';
?>
</h2>
</body>
```

Affiche Bonjour nom prénom !
avec une couleur de fond passée comme paramètre

<http://localhost/.../04-getcol.php?prenom=Jean&nom=Dupont&col=yellow>

Tableaux prédéfinis (superglobals)

Tout script PHP a accès à des tableaux associatifs prédéfinis, rendus disponibles par le serveur Web, dont voici quelques uns :

`$_SERVER`, `$_GET`,

...et d'autres dont on parlera plus tard

`$_POST`, `$_COOKIE`, `$_REQUEST`, `$_SESSION`, ...

Contrôle du flux d'exécution : instruction conditionnelle

```
if (condition) {  
    // code exécuté si la condition est vraie  
} else {  
    // code exécuté si la condition est fausse  
}
```

Contrôle du flux d'exécution : instruction conditionnelle

Conditions :

Égalité de valeur : $\$v1 == \$v2$

Égalité de valeur **et** de type : $\$v1 === \$v2$

Différence de valeur : $\$v1 != \$v2$

Différence de valeur **ou** de type : $\$v1 !== \$v2$

Comparaisons : $>$, $<$, $>=$, $<=$

Opérateurs logiques : $\&\&$, $\|\|$, $!$

Contrôle du flux d'exécution : instruction conditionnelle

Exemple d'utilité de ===

- Certaines fonctions peuvent renvoyer soit une valeur, soit FALSE si l'opération ne réussit pas.
- === nous permet de distinguer entre FALSE et une valeur équivalente à FALSE

Exemple : la fonction `next($tab)`

- ▶ la condition `next($tab) == false` nous permet de comprendre s'il y a un prochain élément dans le tableau ou pas
- ▶ cependant cette condition effectue d'abord une conversion de type : `next($tab)` est d'abord converti en boolean

Qu'est-ce qui se passe si le tableau contient une valeur 0 par exemple?

=== nous permet de résoudre ce problème

Contrôle du flux d'exécution : instruction conditionnelle

<body>

<h2>

<?php

```
if ($_GET['user'] == 'jean') {
```

```
    echo "Bienvenue !";
```

```
} else {
```

```
    echo "Acces interdit";
```

```
}
```

?>

</h2>

</body>

Affiche Bienvenue !
à un seul utilisateur

Remarque : ce n'est pas une bonne idée de passer des informations sensibles (pwd, ...) comme paramètre dans l'url (visible par tous), cf. plus loin

<http://localhost/.../04-if.php?user=jean>

Mettre “en pause” l’interpréteur PHP

On a déjà vu qu’un script php peut alterner des fragments de code et du HTML :

```
<body>
```

```
<h1>Un essai de PHP</h1>
```

```
<p> Bonjour
```

```
<?php
```

```
echo $_GET['prenom'].' '. $_GET['nom'].' !';
```

```
?>
```

```
</p>
```

```
<h2>Vous vous êtes connectés en date du :
```

```
<?php echo date("d / m / y"), " !" ?>
```

```
</h2>
```

```
</body>
```

Ce qui est pratique pour éviter des longues chaines de caractères dans les `echo`

Mettre “en pause” l’interpréteur PHP

Script équivalent au précédent mais moins lisible :

```
<body>
<h1>Un essai de PHP</h1>
<p> Bonjour
<?php
    echo $_GET['prenom'].' '. $_GET['nom'].' !'. ' </p> <h2>Vous vous êtes
connectés en date du :';
    echo date("d / m / y"), " !" ;
?>
</h2>
</body>
```

Mettre “en pause” l’interpréteur PHP

L’interpréteur PHP peut être mis en pause également au milieu d’une instruction de contrôle du flux d’exécution (if, while etc.) :

Reprenons l’exemple de l’affichage à un seul utilisateur :

```
<body>
<?php
if ($_GET['user'] == 'jean') {
    echo "<h2>Bienvenue !</h2>";
} else {
    echo "<h2>Acces interdit</h2>";
}
?>
</body>
```

Supposons que dans le premier cas on veut afficher pas uniquement “Bienvenue !” mais du contenu HTML plus complexe (e.g. une “grande” table)

Mettre “en pause” l’interpréteur PHP

```
<body>
<?php
if ($_GET['user'] == 'jean') {
    echo "<h2>Bienvenue !</h2>";
?>

<table border="1">
<caption>Une table</caption>
<tr><th colspan="2">Lorem ipsum</th><th>dolor</th></tr>
<tr><td>sit</td><td>amet</td><td rowspan="2">consectetur Cras</td></tr>
<tr><td>adipiscing</td><td>elit</td></tr>
<tr><td>ultrices</td><td>lacus</td><td>...</td></tr>
</table>

<?php
} else {
    echo "<h2>Acces interdit</h2>";
}
?>
```

<http://localhost/.../04-pause.php?user=jean>

Contrôle du flux d'exécution : conditionnelle switch

```
switch ($var) {  
    case valeur1:  
        /* bloc... */  
        break;  
    case valeur2:  
        /* bloc... */  
        break;  
    default:  
        /* bloc... */  
}
```

- La valeur de `$var` est comparée avec `valeur1`, `valeur2`, ...
- quand il y a égalité, le bloc d'instructions correspondant est exécuté
- `break;` est nécessaire pour éviter que le bloc suivant soit également exécuté
- le bloc `default` est exécuté si il n'y a pas d'égalité

switch : exemple

```
<head><style>
body{
  background-color:
  <?php
switch ($_GET['col']) {
  case '0' :
    echo "red";
    break;
  case '1':
    echo "yellow";
    break;
  case '2':
    echo "lightblue";
    break;
  default :
    echo "pink";
}
?>
}
</style></head> ...
```

Affiche une couleur de fond
différente selon la valeur d'un
code passé en paramètre

<http://localhost/.../04-switch.php?col=0>

Boucles

```
for (initialisation; condition; fin d'itération) {  
    /* bloc... */  
}  
  
while (condition) {  
    /* bloc... */  
}  
  
do {  
    /* bloc... */  
} while (condition);
```

Contrôle de boucle :

`continue;` // passe à l'itération suivante

`break;` // sort de la boucle

<http://localhost/.../04-boucles.php?n=6>

04-boucles.php

PHP

fonctions et fonctions natives

Fonctions

Une façon de nommer un calcul

Le code peut ensuite être écrit en faisant abstraction de comment ce calcul réalisé

Avantages :

- Rendre le code modulaire

- Eviter de réécrire du code identique (factorisation de code)

- Eviter des erreurs

- Rendre le code plus lisible

- Gagner du temps

Définition / Utilisation de fonction

Définition d'une fonction

```
function nom_fonction($arg1, $arg2, ...) {  
    /* bloc... */  
}
```

Appel (utilisation) d'une fonction

```
nom_fonction($valeur1, $valeur2, ...);
```

Exemple

```
<? php
```

```
function milieu($a, $b) {  
    return floor( ($a+$b) / 2 );  
}
```

```
$x= 31; $y = 50; echo milieu ($x, $y);
```

```
?>
```

⇒ 40

Fonctions et méthodologie de développement

- En PHP les fonctions sont particulièrement utiles pour aider à **séparer la logique de l'application (le code PHP) de la présentation (le HTML)**
- Une bonne méthodologie de développement :
 - ▶ Définir toujours une bibliothèque de fonctions effectuant les différentes tâches nécessaires (affichage du HTML, connexion à la bases de données, calculs spécifique, etc.)
 - ▶ ensuite appeler ces fonctions pour générer la page demandée selon la logique de l'application

04-fonctions.php

- Cette méthodologie peut être poussée jusqu'à définir des structures très génériques de serveurs (pseudo-frames, MVC, ...) qui sont aujourd'hui à la base de la plupart des *frameworks* pour le développement Web

Passage des paramètres à une fonction

Par défaut : **par valeur** (y compris pour les tableaux !)

- ▶ La fonction modifie une copie locale du paramètre, cela n'affecte pas la valeur du paramètre en dehors de la fonction

```
function modifie ($tableau) {  
    $tableau[0]= 2;  
}
```

```
$t= array(0,1);
```

```
modifie($t);
```

```
foreach($t as $case) {  
    echo $case, " ";  
}
```

⇒ 0 1

Passage des paramètres à une fonction : par référence

On peut passer à une fonction une référence vers la variable à mettre à jour

- ▶ La fonction modifie la variable elle-même, et non pas une copie locale, cela affecte la valeur du paramètre en dehors de la fonction

```
function modifie (&$tableau) {  
    $tableau[0]= 2;  
}
```

```
$t= array(0,1);
```

```
modifie($t);
```

```
foreach($t as $case) {  
    echo $case," ";  
}
```

⇒ 2 1

- ▶ Passage par référence possible pour n'importe quelle variable, mais pas pour une constante

Visibilité des variables

```
$a = 6789; //variable globale
```

```
function f ($arg1, $arg2) {
```

```
    $b = 12345; // variable locale
```

```
    echo $a; // n'affiche rien, $a ici est une nouvelle variable locale  
            // non définie
```

```
    ...
```

```
}
```

```
echo $b; //n'affiche rien, $b ici est une nouvelle variable globale  
        // non définie
```

- Les variables internes à une fonction ne sont pas visibles de l'extérieur de la fonction
- Les variables externes ne sont pas visibles dans la fonction
- Une fonction peut acquérir la visibilité des variables globales avec le mot clef “`global`”

Visibilité des variables

```
$a = 6789; //variable globale
```

```
function f ($arg1, $arg2) {
```

```
    global $a; //variable globale $a rendue visible dans f()
```

```
    $a = 33;
```

```
    ...
```

```
}
```

```
f(2, 3);
```

```
echo $a;
```

⇒ 33

Modules PHP et extensions

- Un module est une bibliothèque de fonctions disponibles dans le langage
- L'installation standard de PHP (le “core”) dispose de plusieurs modules natifs (e.g. des fonctions pour travailler avec les chaînes de caractères, les tableaux, ...)
- Il est également possible d'installer un ensemble très riche de modules d'extension
- Voici quelques fonctions utiles appartenant aux modules natifs...

Fonction date

`date($format, $timestamp)`

`$format` : une chaîne de la forme "`spec[-/.]spec[-/.]...`"

Les spécificateurs sont séparés par `–` ou `/` ou `.`

- renvoie la date `$timestamp` sous forme d'une chaîne, au format donné par `$format`
- `$timestamp` est un entier
 - ▶ nombre de secondes depuis 1er jan. 1970 00:00:00 GMT
 - ▶ optionnel
 - valeur per défaut : la date courante

Fonction date

- Spécificateurs : (liste non exhaustive)

- M** abréviation mois en anglais
- F** mois en toute lettre en anglais
- d** jour dans le mois
- l** jour de la semaine en lettres en anglais
- Y** année sur 4 chiffres
- H** heures entre 0 et 24
- h** heures entre 0 et 12
- i** minutes
- S** secondes

- Exemple

```
date('d / M / Y'); // => 14 / Feb / 2020
```

```
date('l / F-d-Y / H.i'); // => Friday / February-14-2020 / 11.15
```

Quelques fonctions natives sur les chaînes de caractères

- `strlen($str)`

Renvoie la longueur de la chaîne `$str`

`strlen("ab de ")` \Rightarrow 7

- `substr($str, $debut [, $long])`

Renvoie le segment de la chaîne `$str` qui commence à l'indice `$debut` et comprend `$long` caractères. Les indices démarrent à 0.

`substr("abcdefgh", 2, 3)` \Rightarrow cde

- `strpos($chaine1, $chaine2)`

Renvoie :

- l'indice du premier caractère de `$chaine2` dans `$chaine1`, si `$chaine2` est contenue dans `$chaine1`
- `FALSE` sinon

Exemple : `strpos("aaabbbccddeebbb", "bb")` \Rightarrow 3

Quelques fonctions natives sur les chaînes de caractères

- `strchr($bottedefoin, $aiguille)` ou
- `strstr($bottedefoin, $aiguille)` (synonymes)
 - ▶ Recherche la chaîne `$aiguille` dans celle `$bottedefoin`
 - ▶ Renvoie :
 - la sous-chaîne de `$bottedefoin` commençant à la première occurrence de `$aiguille`
 - ou `FALSE` si `$aiguille` n'est pas trouvée
 - ▶ Exemple `strchr("www.univ-paris-diderot.fr", ".")`
⇒ `.univ-paris-diderot.fr`
- `strrchr($tasdefoin, $aiguille)`

Idem, mais travaille sur la dernière occurrence

`strrchr("www.univ-paris-diderot.fr", ".");`
⇒ `.fr`

Quelques fonctions natives sur les chaînes de caractères

– strcmp(\$str1, \$str2)

Renvoie 0 si les deux chaînes sont égales

une valeur > 0 si \$str1 > \$str2

une valeur < 0 si \$str1 < \$str2

strcmp("abcde", "bcdef") ⇒ -1

strcmp("abcde", "Abcde") ⇒ 32

Remarque. Pour tester uniquement l'égalité de chaînes de caractères :

== ou === (équivalents si les deux arguments sont du même type)

Quelques fonctions natives sur les chaînes de caractères

- `str_repeat($str, $num)`

`str_repeat("xyz", 5) ⇒ xyzxyzxyzxyzxyz`

- `str_replace($a, $b, $str)`

Remplace toutes les occurrences de la chaîne `$a` par la chaîne `$b` dans la chaîne `$str`

`$a` et/ou `$b` peuvent être des tableaux

`str_replace("m", "M", "Lorem Ipsum") ⇒ LoreM IpsuM`

`str_replace(array("ab", "d"), array("c", "f"), "abd");`

`⇒ cf`

`str_replace(array("ab", "d"), "c", "abd");`

`⇒ cc`

Quelques fonctions natives sur les chaînes de caractères

- Supprimer des caractères “invisibles” (espaces) :

`$str_out = trim($chaine) // supprime au début et à la fin de la chaîne`

`$str_out = ltrim($chaine) // supprime uniquement au début (gauche)`

`$str_out = rtrim($chaine) // supprime uniquement à la fin (droite)`

- Changer la casse

`strtolower($str)`

Renvoie la chaîne `$str` après avoir converti les majuscules en minuscules

`strtolower("LorEM IPsum, dOlor sit aMEt")`

⇒ lorem ipsum, dolor sit amet

`strtoupper($str)`

Renvoie la chaîne `$str` après avoir converti les minuscules en majuscules

`strtoupper("LorEM IPsum, dOlor sit aMEt")`

⇒ LOREM IPSUM, DOLOR SIT AMET

Quelques fonctions natives sur les chaînes de caractères

Manipulation de chaînes pour HTML :

`htmlspecialchars($str)`

- Renvoie une chaîne où `<`, `>`, `&`, et `"` sont remplacés par des entités HTML (`<`, `>`, `&`, et `"`;)
- Avec l'option `htmlspecialchars($str, ENT_QUOTES)`, les guillemets simples `'` sont également remplacés par l'entité correspondante

`htmlentities($str)`

Renvoie une chaîne où tous les caractères ayant une entité HTML correspondante sont remplacés par celle-là

Exemple

```
<body>
```

```
<?php
```

```
$par = 'une chaine qui doit aller dans un <p>';
```

```
$attr = 'une chaîne qui doit aller dans un "attribut HTML"';
```

```
$str = 'une chaîne contenant des symboles qui ont une entité associée,  
mais ne sont pas des symboles spéciaux HTML';
```

```
/*$par = htmlspecialchars($par);
```

```
$attr = htmlspecialchars($attr);
```

```
$str = htmlentities($str);*/
```

```
?>
```

```
<p> voici <?php echo $par?> </p>
```

```
<abbr title="<?php echo $attr?>"> Survole moi! </abbr>
```

```
<p> voici <?php echo $str?> </p>
```

```
</body>
```

<http://localhost/.../04-htmlent.php>

view source

04-htmlent.php

view source

Quelques fonctions natives sur les chaînes de caractères

Manipulation de chaînes pour HTML :

`urlencode($str)`

- ▶ Encode une chaîne pour l'utiliser comme partie d'un URL
 - Les caractères non alphanumériques sont remplacés par `%xx`
 - Les espaces remplacés par le caractère `+`
- ▶ Ce codage est le standard pour les URL, et doit être respecté
- ▶ Attention à encoder les paramètres de l'URL séparément pour ne pas échapper `?` et `&`

```
$p1 = "[lorem ipsum]";  
$p2 = "{ }";  
$query_string = 'par1=' . urlencode($p1) . '&par2=' . urlencode($p2);  
$query_string = htmlentities($query_string);  
echo '<a href="04-urlencget?' . $query_string . '>' . 'lien avec  
parametres </a>' ;
```

<localhost/.../04-urlenc.php>

Quelques fonctions natives sur les chaînes de caractères

Il existe beaucoup d'autres fonctions pour manipuler les chaînes de caractères !

Référence complète :

<https://www.php.net/manual/fr/book.strings.php>

Quelques fonctions natives sur les tableaux

- `count($array)`
`sizeof($array)`

Retourne la taille d'un tableau

- `implode($sep, $array)`

Concatène les éléments du tableau, séparés par `$sep`

```
echo implode ("/", array("dossier", "fichier.html"));
```

⇒ dossier/fichier.html

- `explode($sep, $str)`

Décompose la chaîne `$str` et retourne un tableau contenant les sous-chaînes de `$str` séparées par `$sep`

```
$tab = explode ("/", "dossier1/dossier2/fichier.html");
```

```
foreach($tab as $str) echo $str, ' ';
```

⇒ dossier1 dossier2 fichier.html

Quelques fonctions natives sur les tableaux

- `array_diff($array1, $array2)`

Retourne un tableau avec les éléments de `$array1` qui ne sont pas dans `$array2`

- `array_sum($array)`

Retourne la somme des éléments du tableau `$array`

- `array_unique($array)`

Retourne un tableau ne contenant que les éléments distincts du tableau initial

Quelques fonctions natives sur les tableaux

– `sort($array);`

trie le tableau

Variantes : `asort`, `ksort`, `rsort`, `arsort`, `krsort`

a Maintient l'association clé / valeur

k Tri par clé (au lieu de par valeur)

r Tri en ordre inverse

Quelques fonctions natives sur les tableaux

```
$livres = array("Les Misérables", "Notre Dame de Paris",  
               "Quatre-vingt treize");
```

```
rsort($livres);
```

```
var_dump($livres);
```

```
⇒ array(3) { [0]=> string(19) "Quatre-vingt treize"  
             [1]=> string(19) "Notre Dame de Paris"  
             [2]=> string(14) "Les Misérables" }
```

```
$livres = array("Les Misérables", "Notre Dame de Paris",  
               "Quatre-vingt treize");
```

```
arsort($livres);
```

```
var_dump($livres);
```

```
⇒ array(3) { [2]=> string(19) "Quatre-vingt treize"  
             [1]=> string(19) "Notre Dame de Paris"  
             [0]=> string(14) "Les Misérables" }
```

Inclusion

On peut inclure un fichier PHP dans un autre, avec la fonction

```
include("fichier.php");
```

ou

```
require("fichier.php");
```

(parenthèses optionnelles)

- ▶ Le fichier inclus doit contenir les balises `<?php... ?>`
- ▶ Le script qui utilise l'instruction `include("fichier.php")` est exécuté comme si le code dans `fichier.php` (à part les balises `<?php... ?>`) était “recopié” à la place de cette instruction
- ▶ Cela a un impact sur la visibilité des variables

Inclusion

- Exemple : inclure des constantes et variables

```
//fichier.php
```

```
<?php
    define("MAGIC_NUMBER", 49346739);
    define("COULEUR", "rouge");
    $pwd = "ajhd3k";
?>
```

```
//app.php
```

```
<?php
include("fichier.php");

echo COULEUR; // =>rouge

echo 5* MAGIC_NUMBER; // => 246733695

echo $pwd; // => ajhd3k

?>
```

\$pwd est une variable **globale** dans app.php

Inclusion

- Exemple : inclure des constantes et variables

```
//fichier.php
```

```
<?php
    define("MAGIC_NUMBER", 49346739);
    define("COULEUR", "rouge");
    $pwd = "ajhd3k"
?>
```

```
//app.php
```

```
<?php
function f () {
    include("fichier.php");
    echo $pwd; // => ajhd3k
}
...
?>
```

\$pwd est une variable **locale** à la fonction **f()** dans app.php

Inclusion

- Exemple : inclure un ensemble de fonctions (une “bibliothèque”)

```
//binaire.php
```

```
<?php
```

```
function param() { ... }
```

```
function binary($n) { ... }
```

```
function afficherChiffre ($image) { ... }
```

```
function afficherNombre ($bits) { ... }
```

```
?>
```

```
//app.php
```

```
<?php
```

```
    include(“binaire.php”);
```

```
?>
```

```
<!DOCTYPE html>
```

```
...
```

```
<?php
```

```
$m = param();
```

```
$bits = binary($m);
```

```
afficherNombre($bits);
```

```
?>...
```


Inclusion

Différence entre `include` et `require` :

si une erreur est générée dans l'exécution du script inclus,

- `require` arrête l'exécution de tout le script,
- `include` génère seulement une alerte

Donc : préférer `require` !

```
include_once("fichier.inc");
```

```
require_once("fichier.inc");
```

Identique à `include` (resp. `require`) mais évite les inclusions multiples inutiles ou dangereuses

```
//fichier0.php  
<?php  
function f() { ... };  
...  
?>
```

```
graph TD; F0["//fichier0.php<br/><?php<br/>function f() { ... };<br/>...<br/>?>"] --> F1["//fichier1.php<br/><?php<br/>require 'fichier0.php';<br/>...<br/>?>"]; F0 --> F2["//fichier2.php<br/><?php<br/>require 'fichier0.php';<br/>...<br/>?>"]; F1 --> F3["//fichier3.php<br/><?php<br/>require 'fichier1.php';<br/>require 'fichier2.php';<br/>...<br/>?>"]; F2 --> F3;
```

```
//fichier1.php  
<?php  
require "fichier0.php";  
...  
?>
```

```
//fichier2.php  
<?php  
require "fichier0.php";  
...  
?>
```

```
//fichier3.php  
<?php  
require "fichier1.php";  
require "fichier2.php";  
...  
?>
```

Probleme !
fichier0 inclus deux fois

```
//fichier0.php
<?php
function f() { ... };
...
?>
```

```
graph TD; F0["//fichier0.php<br/><?php<br/>function f() { ... };<br/>...<br/>?>"] --> F1["//fichier1.php<br/><?php<br/>require_once<br/>\"fichier0.php\";<br/>..."]; F0 --> F2["//fichier2.php<br/><?php<br/>require_once<br/>\"fichier0.php\";<br/>..."]; F1 --> F3["//fichier3.php<br/><?php<br/>require \"fichier1.php\";<br/>require \"fichier2.php\";<br/>...<br/>?>"]; F2 --> F3;
```

```
//fichier1.php
<?php
require_once
"fichier0.php";
...
```

```
//fichier2.php
<?php
require_once
"fichier0.php";
...
```

```
//fichier3.php
<?php
require "fichier1.php";
require "fichier2.php";
...
?>
```

OK !
fichier0 inclus
une seule fois

Inclusion : avertissement

- **Mauvaise pratique** très répandue (exemples, tutoriels sur le Web, ...) : utiliser l'inclusion partout!
 - pour copier un fragment de code là où on a besoin qu'il soit exécuté
- Cela est une violation de tout principe de modularité et bonne structuration du code !
- Les fonctions (concept primordial en programmation) existent exactement pour cela, c'est la bonne approche à adopter!

include / require devrait être utilisé essentiellement pour :

- inclure des bibliothèques de fonctions et classes, des constantes et variables globales,
- éventuellement des parties statiques du script destinées à l'affichage (entete de la page, pied de page, menus, ...)
- et c'est tout!

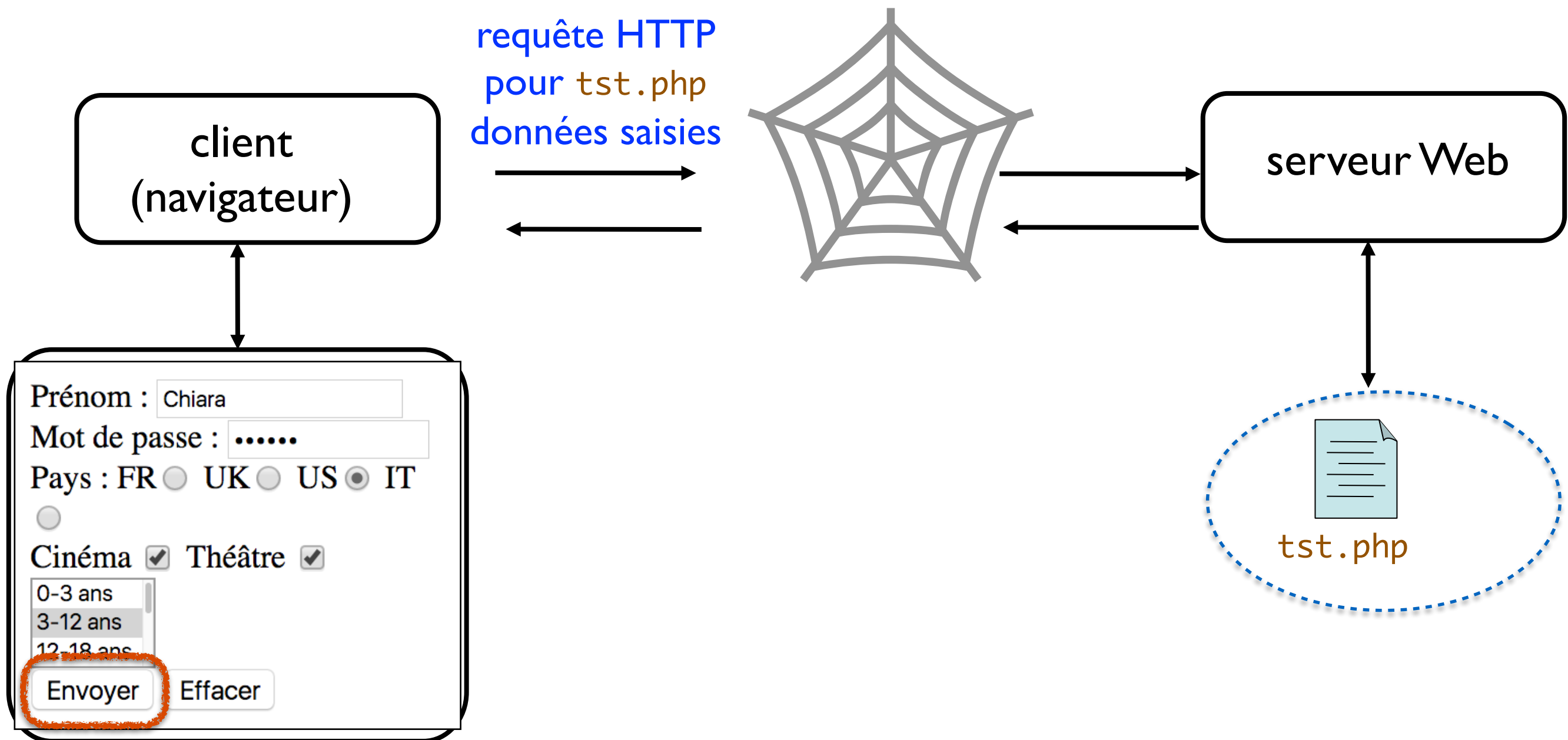
Traitement de formulaires en PHP

Rappel : envoi d'un formulaire

- À l'envoi d'un formulaire HTML le navigateur envoie une nouvelle requête HTTP au serveur pour charger la page indiquée dans l'attribut `action` du formulaire

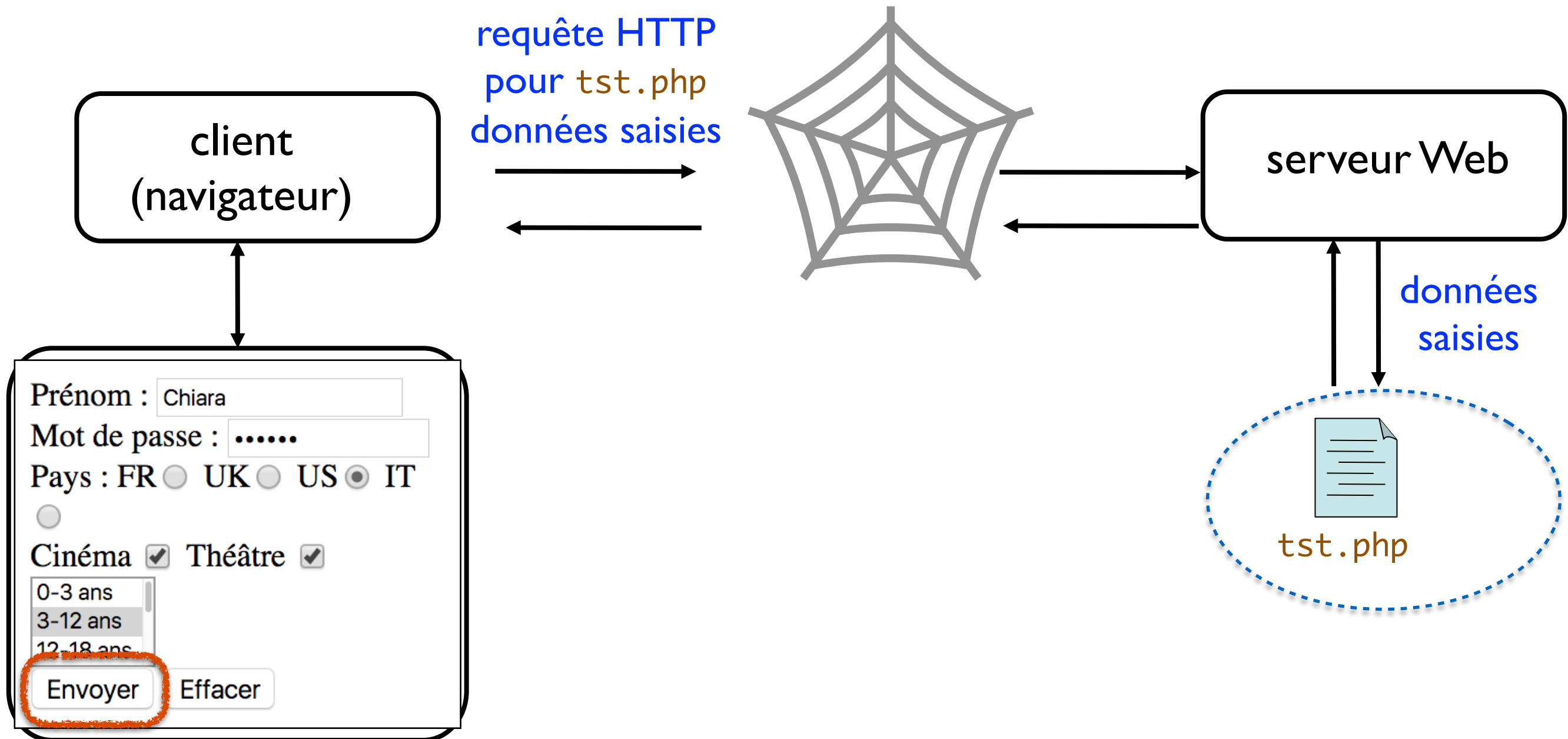
```
<form action="tst.php" method="..."> ... </form>
```

- et inclut les données saisies par l'utilisateur dans cette requête



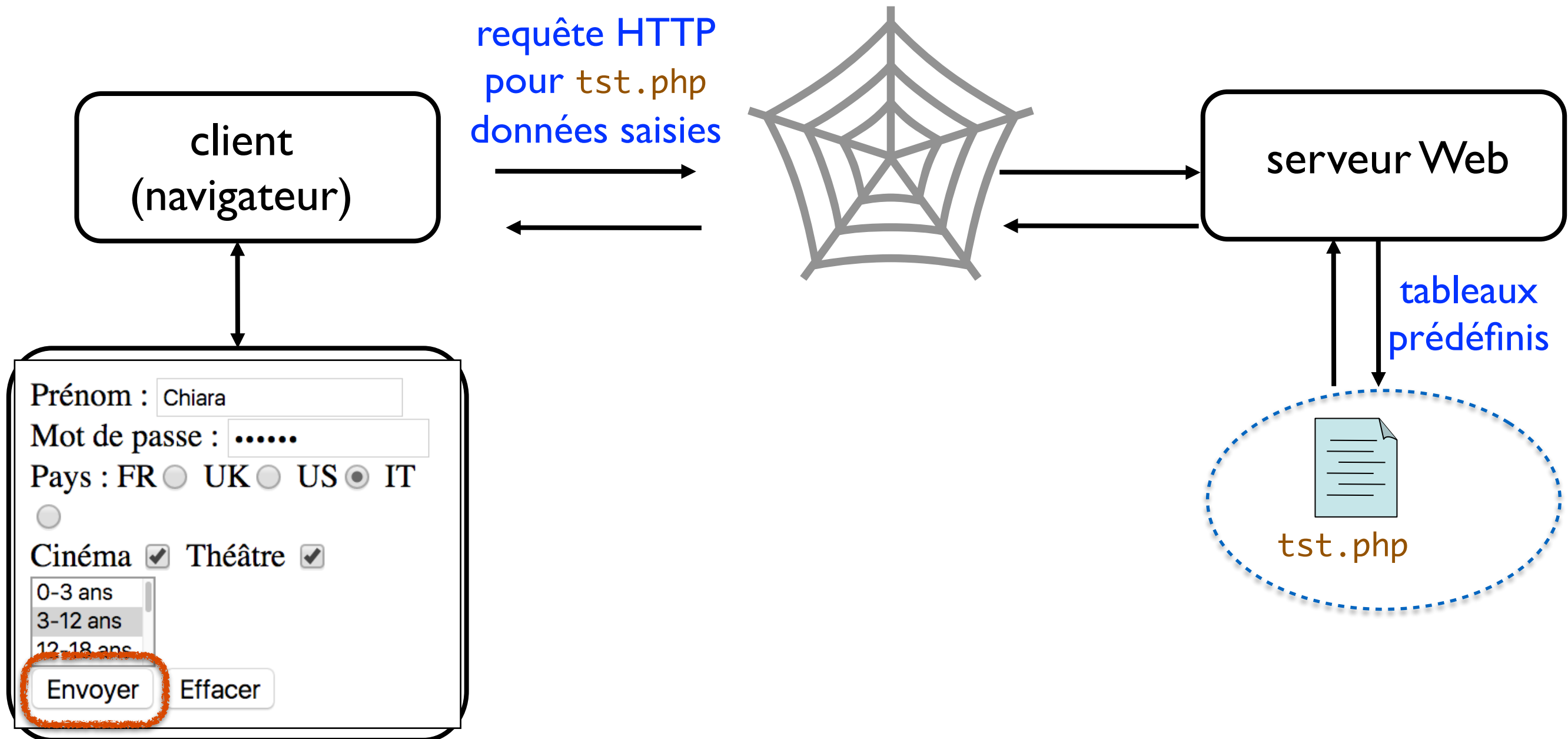
Envoi d'un formulaire

- Les données saisies sont reçues par le serveur qui les “passe” au script `tst.php` pour être traitées



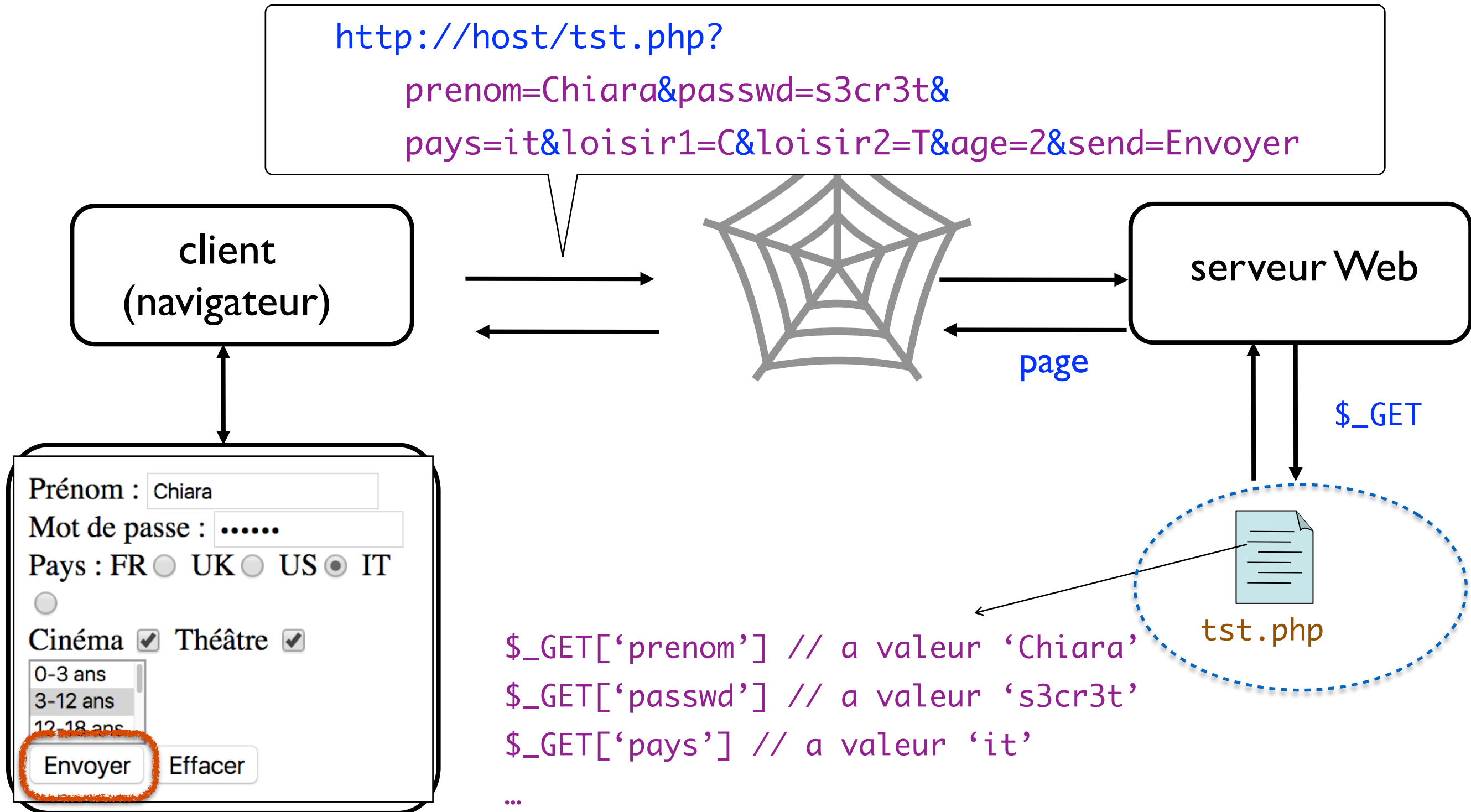
Envoi du formulaire - suite

- le serveur Web range les données saisies dans les tableaux prédéfinis `$_GET`, `$_POST`, `$_REQUEST`, ...
- Le script `tst.php` accède à ces tableaux pour récupérer et traiter les données saisies



Envoi du formulaire - méthode GET

Si la méthode d'envoi est GET : `<form action="tst.php" method="get"> ... </form>`
les données saisies sont envoyées comme des couples nom=valeur dans l'URI de la requête. **Elle sont donc rangées dans le tableau `$_GET`**

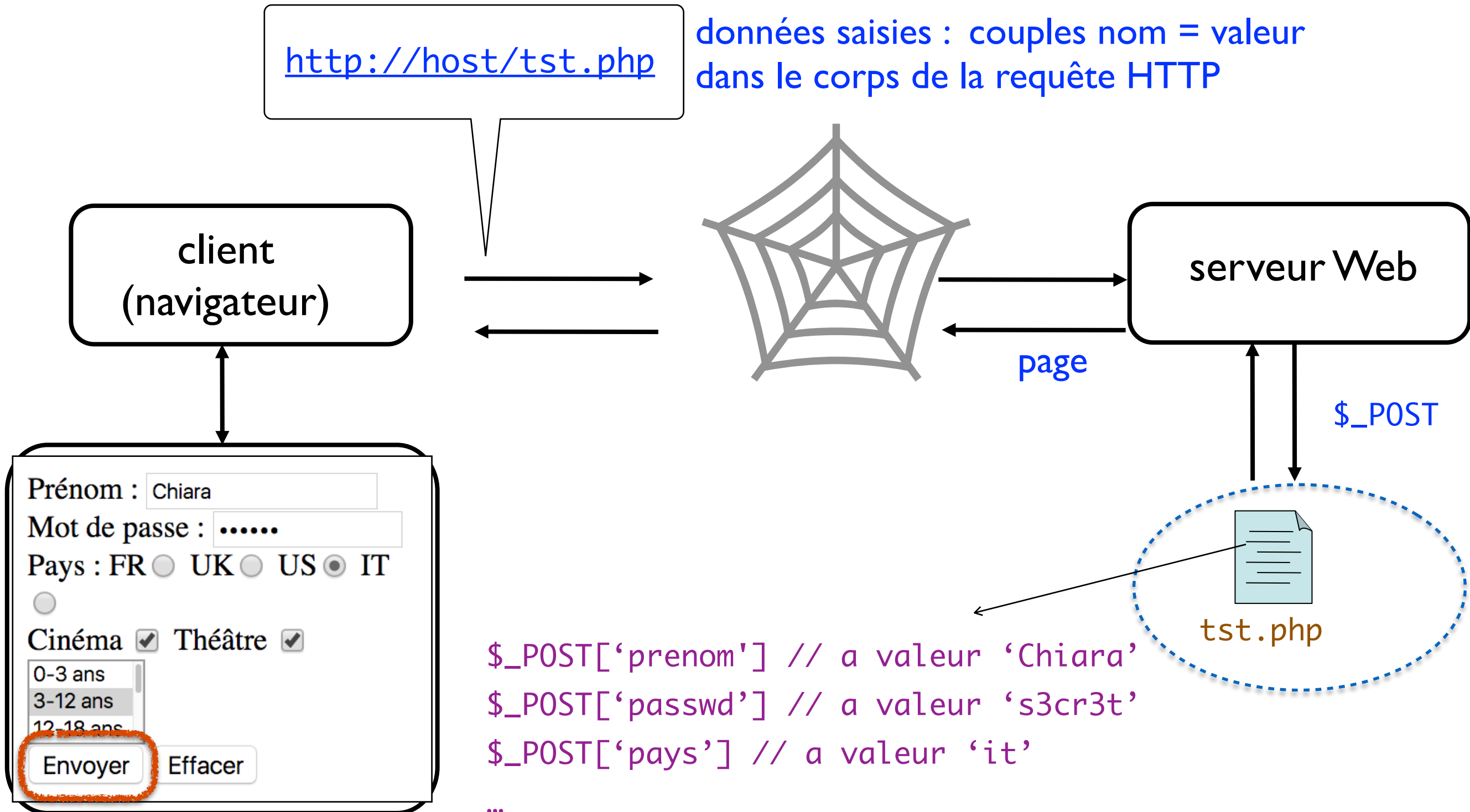


Envoi du formulaire - méthode POST

Si la méthode d'envoi est POST : `<form action="tst.php" method="post"> ... </form>`
les données saisies sont envoyées dans le corps la requête HTTP et pas visibles dans l'URI. Elles sont rangées dans le tableau `$_POST`

<http://host/tst.php>

données saisies : couples nom = valeur
dans le corps de la requête HTTP



Correspondance entre noms et clefs

Remarque. la valeur saisie dans un élément du formulaire avec attribut `name = "nom_champ"` sera disponible dans `$_GET['nom_champ']` (ou `$_POST['nom_champ']` , selon la méthode)

```
//form.html
```

```
<form action="tst.php" method="get">
```

```
Prénom : <input type="text" name="prenom" size="30" value="Votre nom">  
<br><br>
```

```
Mot de passe : <input type="password" name="passwd" size="16"><br><br>
```

```
...
```

```
//tst.php
```

```
$_GET['prenom']
```

```
$_GET['passwd']
```

\$_REQUEST

Remarque. \$_REQUEST regroupe le contenu des tableaux \$_GET, \$_POST et \$_COOKIES

il peut donc être utilisé toujours au lieu de ces trois tableaux

//form.html

```
<form action="tst.php" method="get">
```

```
Prénom : <input type="text" name="prenom" size="30" value="Votre nom">
```

```
<br><br>
```

```
Mot de passe : <input type="password" name="passwd" size="16"><br><br>
```

...

//tst.php

```
$_REQUEST['prenom']
```

```
$_REQUEST['passwd']
```

\$_REQUEST

Remarque. \$_REQUEST regroupe le contenu des tableaux \$_GET, \$_POST et \$_COOKIES

il peut donc être utilisé toujours au lieu de ces trois tableaux

//form.html

```
<form action="tst.php" method="post">
```

```
Prénom : <input type="text" name="prenom" size="30" value="Votre nom">
```

```
<br><br>
```

```
Mot de passe : <input type="password" name="passwd" size="16"><br><br>
```

...

//tst.php

```
$_REQUEST['prenom']
```

```
$_REQUEST['passwd']
```

Traitement de formulaires : tâches de base

Un exemple très simple de traitement de formulaire

04-form-simple.html

04-form-simple.php

Dans une situation plus réaliste les actions associées au traitement d'un formulaire sont plusieurs, et comprennent certaines tâches récurrentes telles que :

- récupérer et “préparer” les données saisies par l'utilisateur
- valider l'input de l'utilisateur (champs requis, bien formés, ...)
- interroger/modifier une base de données (e.g. recherche de vols, enregistrement d'un utilisateur....)

Récupérer et “préparer” les données du formulaire

- Récupérer les valeurs d'un champ de texte est simple (déjà vu)
- Récupérer les valeurs de certains autres éléments du formulaire (checkbox, radio buttons, listes à sélection multiple) peut être un peu différent
- Dans tous les cas les données récupérées doivent être préparées avant le traitement

Récupérer la valeur d'un checkbox individuel

```
//04-check.html
```

```
<form action="04-check.php" method="post">
```

```
...
```

```
<input type="checkbox" name="accepter" value="OK" checked> J'accepte les  
conditions d'utilisation
```

```
...
```

```
//04-check.php
```

```
// si la checkbox a été cochée, la valeur de son attribut value ("OK" en  
l'occurrence) a été affectée à la variable $_POST['accepter']
```

```
//sinon $_POST['accepter'] est une variable non-définie
```

```
if (isset ($_POST['accepter'])) $acc = true;
```

```
else $acc = false;
```


Récupérer la valeur de plusieurs *checkboxes* reliées

//04-check.html

```
<form action="04-check.php" method="post">
```

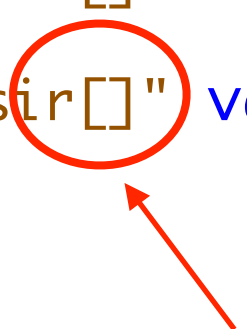
...

```
Cinéma <input type="checkbox" name="loisir[]" value="C">
```

```
Théâtre <input type="checkbox" name="loisir[]" value="T">
```

```
Musique <input type="checkbox" name="loisir[]" value="M">
```

...



rappel : le nom de la *checkbox* doit terminer par [] pour que toutes les valeurs cochées soient accessibles avec le même nom

Récupérer la valeur de plusieurs *checkboxes* reliées

```
//04-check.html
```

```
<form action="04-check.php" method="post">
```

```
...
```

```
Cinéma <input type="checkbox" name="loisir[]" value="C">
```

```
Théâtre <input type="checkbox" name="loisir[]" value="T">
```

```
Musique <input type="checkbox" name="loisir[]" value="M">
```

```
...
```

```
//04-check.php
```

```
// la variable $_POST['loisir'] est un tableau contenant dans l'ordre, les  
valeurs (attribut value) de toutes les checkboxes de nom loisir qui ont été  
cochées par l'utilisateur
```

```
//$_POST['loisir'] est non-défini si aucune de ces checkbox n'a été cochée
```

```
$tab = $_POST['loisir'];
```

```
if (isset ($tab)) { foreach ($tab as $val) echo $val; }
```

04-check.html
04-check.php

Récupérer la valeur d'une liste déroulante

```
//04-select.html
```

```
<form action="04-select.php" method="post">
```

```
<select name="age" size="1">
```

```
  <option value="1">0-3   ans</option>
```

```
  <option value="2">3-12  ans</option>
```

```
  <option value="3">12-18 ans</option>
```

```
  ...
```

```
</select>
```

idem pour les radio buttons

```
...
```

```
//04-select.php
```

```
// la variable $_POST['age'] contient la valeur de l'attribut value de  
l'option sélectionnée par l'utilisateur
```

```
//$_POST['age'] est non-défini si aucune option n'a été sélectionnée
```

```
if (isset($_POST['age'])) echo $_POST['age'];
```

Récupérer la valeur d'une liste déroulante à sélection multiple

//04-select.html

```
<form action="04-select.php" method="post">
<select name="loisir[]" size="3" multiple>
  <option value="C"> Cinema </option>
  <option value="M"> Musique </option>
  <option value="T"> Theatre </option>
</select>
```

...

//04-select.php

// Comme dans le cas des checkbox multiples :

// la variable \$_POST['loisir'] est un tableau contenant, dans l'ordre, les valeurs (attribut value) de toutes les options de la liste déroulante loisir qui ont été sélectionnées par l'utilisateur

```
$tab = $_POST['loisir'];
```

```
if (isset ($tab)) { foreach ($tab as $val) echo $val; }
```

04-select.html
04-select.php

Préparer les données du formulaire pour le traitement

- Les données saisies par l'utilisateur peuvent contenir
 - ▶ des espaces inutiles
 - ▶ des caractères spéciaux HTML
 - ▶ des parties dangereuses (injection de code)
- Il est nécessaire de les nettoyer avant le traitement

Préparer les données du formulaire pour le traitement

- ▶ Si les données saisies sont susceptibles d'être insérées dans une page HTML
 - la présence de caractères spéciaux HTML, ou de portions de HTML mal formé, peut rendre le HTML généré par le serveur non valide
 - problème encore plus grave : les données saisies peuvent contenir une balise `<script>` avec du code Javascript possiblement "méchant" qui sera inséré dans la page générée par le serveur !
 - protéger avec `htmlspecialchars` , ou `htmlentities`
- ▶ Si les données sont susceptibles d'être insérées dans une requête pour la BD
 - protéger avec les fonctions d'échappement spécifiques au système de BD utilisé (voir plus loin pour MySQL)

Préparer les données du formulaire pour le traitement

Exemple de fonction de pre-traitement des données (sans protection pour la BD) :

```
function preTraiterChamp($champ) {  
    if (!empty($champ)) {  
        $champ = trim($champ);  
        $champ = htmlspecialchars($champ);  
    }  
    return $champ;  
}
```

Préparer les données du formulaire pour le traitement

Exemple

//04-form.html

```
<form action="04-form.php" method="post"> ...
```

//contient les champs : 'prenom', 'passwd', 'rpasswd', 'comment',
'pays'(radio), 'loisir[]'(checkbox), 'age'(select), 'accepter'(checkbox)

//04-form.php

```
$donnees = $_POST;
```

```
preTraiter ($donnees);...
```

//fonc_valid.php

```
function preTraiter(&$donnees) {
```

```
    $donnees["prenom"] = preTraiterChamp($donnees["prenom"]);
```

```
    $donnees["comment"] = preTraiterChamp($donnees["comment"]); }
```


Valider les données du formulaire

- L'application s'attend à ce que les données saisies soient “valides” :
 - ▶ bien formées (e.g. adresse mails au bon format, pwd avec au moins un chiffre, deux pwds identiques etc.)
 - ▶ non vides : certaines données peuvent être obligatoires

Valider les données du formulaire : champs requis

Exemple. On introduit une variable globale qui indique les champs requis

```
//fonc_valid.php
```

```
$REQUIS["prenom"] = true;
```

```
$REQUIS["passwd"] = true;
```

```
$REQUIS["rpasswd"] = true;
```

```
$REQUIS["accepter"] = true;
```

Valider les données du formulaire : champs requis

Exemple de fonction de vérification des champs requis :

```
//fonc_valid.php
```

```
function verifierRequis (&$donnees, &$erreur) {  
    global $REQUIS;  
    $ok = true;  
    foreach ($REQUIS as $champ => $valeur) {  
        if (empty($donnees[$champ])) {  
            $erreur[$champ] = true;  
            $ok = false;  
        }  
    }  
    return $ok;  
}
```

```
//04-form.php
```

```
$erreurs_requis = array();  
$ok = verifierRequis($donnees, $erreurs_requis);  
if ($ok) page_succes($donnees);  
else page_erreur($erreurs_requis);
```

04-form/04-form2/
localhost/../04-form.html

Valider les données du formulaire : format des données

- Les **expressions régulières** permettent de vérifier qu'une chaîne de caractères a un format donné (plus d'infos en TP)
- L'extension "**Filtres PHP**" fournit plusieurs fonctionnalités de validation (email, url, IP, ...)
- Certains **éléments HTML5** (url, email, etc.) permettent une vérification du format préalable à l'envoi du formulaire (pas encore supportés par tous les navigateurs)
- D'autres vérifications sont à faire le cas par cas (e.g deux mdp égaux, ...)

Valider les données du formulaire : format des données

Exemple

```
//fonc_valid.php
```

```
function pwd_valide ($pwd) {  
    return preg_match('/[0-9]/', $pwd);  
    //ou quelque chose de plus compliqué  
}
```

```
function verifierFormat(&$donnees, &$erreur) {  
    $ok = true;  
    if (!pwd_valide($donnees["passwd"])) {  
        $erreur["passwd"] = true;  
        $ok = false;  
    }  
    if ($donnees["passwd"] != $donnees["rpasswd"]) {  
        $erreur["rpasswd"] = true;  
        $ok = false;  
    }  
    return $ok;  
}
```

Valider les données du formulaire : format des données

//04_form.php

...

```
$erreurs_requis = array();
```

```
$erreurs_format = array();
```

```
$ok1 = verifierRequis($donnees, $erreurs_requis);
```

```
$ok2 = verifierFormat($donnees, $erreurs_format);
```

```
if ($ok1 && $ok2) page_succes($donnees);
```

```
else page_erreur($erreurs_requis, $erreurs_format);
```

Re-affichage du formulaire

Au lieu d'afficher les erreurs dans une nouvelle page vide, en général on préfère re-afficher le formulaire, avec les erreurs indiqués.

On peut introduire par exemple une fonction qui re-affiche la page d'enregistrement; le formulaire sera pré-rempli avec les valeurs saisies, et annoté avec les erreurs :

```
// form.php
```

```
...
```

```
if ($ok1 && $ok2) page_succes($donnees);
```

```
else page_enreg($donnees, $erreurs_requis, $erreurs_format);
```

```
...
```

```
//fonc_affichage.php
```

```
function page_enreg(&$donnees, &$erreurs_requis, &$erreurs_format) {
```

```
    afficher_entete("Enregistrement");
```

```
    $erreurs = erreurs($erreurs_requis, $erreurs_format);
```

```
    afficher_formulaire($donnees, $erreurs);
```

```
    afficher_pied_page();
```

```
}
```

Re-affichage du formulaire

La fonction erreurs calcule l'erreur à afficher à côté de chaque champ du formulaire; l'erreur requis est prioritaire (écrase l'erreur de format)

```
// fonc_affichage.php
```

```
function erreurs (&$erreurs_requis, &$erreurs_format) {  
    $erreurs = array();  
    if ($erreurs_format["passwd"])  
        $erreurs["passwd"] = " * le mot de passe doit contenir au moins  
                                un chiffre ! * ";  
    ...  
    foreach ($erreurs_requis as $champ => $val) {  
        $erreurs[$champ] = " * ce champ est requis ! * ";  
    }  
    return $erreurs;  
}
```


Re-affichage du formulaire

```
// fonc_affichage.php
```

```
function afficher_formulaire(&$donnees, &$erreurs) { ?>
```

```
    <form action="04-form.php" method="post">
```

```
        ... // formulaire prerenpli avec $donnees et annoté avec $erreurs
```

```
    </form>
```

```
<?php
```

```
}
```

```
...
```

Re-affichage du formulaire

Afficher un champs du formulaire pré-rempli et annoté

- champs de texte :

```
<form action="04-form4.php" method="post">
```

...

```
Prénom : <input type="text" name="prenom" size="30" value="
               <?php echo $donnees['prenom']; ?>">
```

```
<span class="error"> <?php echo $erreurs['prenom']; ?> </span>
```

...

```
</form>
```

Re-affichage du formulaire

Afficher un champs du formulaire pré-rempli et annoté

- checkbox, radio, liste déroulante, ... :

```
<input type="checkbox" name="accepter" value="OK"
<?php echo checkedif($donnees['accepter'], "OK");?> >
```

J'accepte les conditions d'utilisation

```
<span class="error"> <?php echo $erreurs['accepter']; ?> </span>
```

avec

```
function checkedif ($var, $val) {
    if (isset($var) && $var == $val) return "checked" ;
    else return "";
}
```

Logique de contrôle du formulaire d'enregistrement

Avec la fonction d'affichage du formulaire, le script form.php de traitement du formulaire peut gérer également la première connexion (formulaire vide)

Pas besoin de la page initiale .html ! form.php peut contrôler complètement la fonctionnalité d'enregistrement

Exemple

Pour remplir le formulaire “vide” on utilise des données par défaut, calculées par cette fonction

```
//fonc_valid.php
```

```
function remplir_valeurs_defaut (&$donnees){  
    $donnees["accepter"] = "OK";  
    $donnees["age"] = 2;  
    $donnees["pays"] = "fr";  
}
```

Logique de contrôle du formulaire d'enregistrement

Exemple

...ensuite on teste si la page a été demandée pour le traitement des données saisies ou pas :

```
//form.php
```

```
$erreurs_requis = array(); $erreurs_format = array();
```

```
$donnees = array();
```

```
remplir_valeurs_defaut($donnees);
```

```
if ($_SERVER ["REQUEST_METHOD"]=="POST") {
```

```
    $donnees = $_POST; preTraiter ($donnees);
```

```
    $ok1 = verifierRequis($donnees, $erreurs_requis);
```

```
    $ok2 = verifierFormat($donnees, $erreurs_format);
```

```
    if ($ok1 && $ok2) {
```

```
        page_succes($donnees);
```

```
        exit;
```

```
    }
```

```
}
```

```
page_enreg($donnees,$erreurs_requis,$erreurs_format); //vide ou pré-remplie
```

Logique de contrôle du formulaire d'enregistrement

Le script précédent est une façon typique de contrôler la soumission de formulaires

Il a un seul inconvénient : la page de succès est envoyée comme réponse à la requête HTTP qui a envoyé les données du formulaire

conséquence : recharger la page de succès implique la re-soumission des données du formulaire

toute action faite par le script form.php (e.g. enregistrement dans la BD) sera répétée !

<localhost/.../04-form5/04-form.php>

Logique de contrôle du formulaire d'enregistrement

Le script précédent est une façon typique de contrôler la soumission de formulaires

Il a un seul inconvénient : la page de succès est envoyée comme réponse à la requête HTTP qui a envoyé les données du formulaire

Solution :

- un nouveau URL pour la page de succès
- redirection vers cette page en cas de succès

Redirection et modification des en-têtes HTTP

header(\$str)

Permet de modifier les entêtes des messages HTTP envoyés par le serveur au navigateur

Doit être appelé avant d'afficher quoi que ce soit (y compris des espaces)

Exemples de \$str

"Content-type: application force-download"

Demande au navigateur de considérer le contenu du message comme un fichier à sauvegarder, non pas à afficher.

"Content-disposition: filename=xxx"

Propose un nom par défaut au navigateur

Attention aux espaces dans "xxx"!

"Location: http://host/new_homepage.html"

Redirection HTTP : Demande au navigateur de charger une nouvelle page à la place de celle demandée

Redirection et modification des en-têtes HTTP

`header("Location : ...")` est à utiliser avec parcimonie!

- ▶ Utilisation recommandée :
 - ▶ redirection vers un site externe (e.g. homepage qui a changé d'hébergeur, ...)
 - ▶ redirection interne : pour terminer un script dont le rôle n'est pas d'afficher une page, mais d'effectuer une opération (i.e. enregistrer les données d'un formulaire dans la BD, ...)
 - à la fin de l'opération : `header("Location : success_page.php")`
 - cela évite le problème du "refresh" de la page du formulaire
- **Attention** : ne pas utiliser les redirections internes (i.e. vers des pages du même serveur) comme un moyen de réutiliser le code! on a d'autres moyens pour ce faire :
 - les fonctions, les classes, éventuellement les inclusions, ...

Traitement de formulaire : une page protégée

- Une autre application typique des formulaires est la gestion d'une page protégée par login
- Structure typique du code similaire au formulaire d'enregistrement (avec re-affichage)
- Attention : cette solution ne protège qu'une page
 - ▶ voir plus loin (les sessions) pour protéger l'accès à un ensemble de pages

04-login-page / 04-login-page.php
/ fonctions.php
/fonc_affichage.php

localhost/.../04-login-page.php

Maintenir l'état de l'interaction client-serveur:
cookies et gestion de sessions

Maintenir l'état de l'interaction client-serveur

- Le protocole HTTP fonctionne par cycles de demande - réponse
 - ▶ un client demande une page au serveur, le serveur prépare et renvoie la réponse
 - ▶ Chaque cycle est indépendant de l'autre
 - ▶ le protocole lui même ne fournit aucun moyen de base pour transférer de l'information d'un cycle à l'autre (protocole “sans état”)
- Parfois on a besoin de transfer de l'information entre cycles. Exemples :
 - dans un premier cycle un utilisateur s'est authentifié
 - quand le même utilisateur demande une autre page du site, le serveur a besoin de savoir qu'il s'est authentifié au préalable
 - ▶ Un exemple moins critique :
 - dans une premiere connexion l'utilisateur spécifie la langue dans laquelle il veut que les pages soient affichées
 - quand le même utilisateur demande une autre page du site, le serveur a besoin de connaître cette information

Maintenir l'état de l'interaction client-serveur

- Pas de mécanisme tout prêt dans HTTP, mais tous les moyens pour l'implémenter !
 - ▶ une requête HTTP peut transférer des paramètres / données venant du client qui l'a émise
 - trois mécanismes pour transférer des données :
 - dans l'url (GET)
 - dans le corps de la requête HTTP (POST)
 - à travers les COOKIES
- Plusieurs solutions possibles pour maintenir l'état de l'interaction client-serveur en utilisant ces mécanismes...(cf. prochain slide)

Maintenir l'état de l'interaction client-serveur

- Solutions bas-niveau :
 - ▶ 1) Transférer l'information d'un cycle à l'autre en l'incluant à chaque fois dans les paramètres de GET ou POST
 - ▶ 2) Stocker l'information sur la machine client (dans des COOKIES) de façon à ce qu'elle soit re-transférée à chaque nouvelle connexion
- Solution haut niveau :
 - ▶ 3) Contrôle des SESSIONS
 - implémentée en général par une des solutions bas niveau

Maintenir l'état de l'interaction client-serveur

Solution I : transfert d'information par les paramètres GET ou POST

- Le client demande une page et fournit une certaine information au serveur (e.g. la langue souhaitée)
- À chaque fois que le serveur prépare une page de réponse : il inclut l'information récupérée comme paramètre additionnel dans tous les liens ou formulaires de la page
 - ▶ l'information sera donc envoyée au serveur avec chaque nouvelle requête du même client
 - ▶ \Rightarrow information disponible au serveur pendant toute l'interaction du client avec le site

Maintenir l'état de l'interaction client-serveur

Solution I : transfert d'information par les paramètres GET ou POST

Exemple

demander la langue dans laquelle les pages doivent être affichées

- Chaque page commence par appeler une fonction qui teste si la langue a été passée dans les paramètres de GET ou POST, et qui demande de choisir la langue si ce n'est pas le cas
- Si la langue est disponible, la page est affichée dans la langue demandée **et la langue est ajoutée comme paramètre de chaque lien et formulaire de la page**
- Pour inclure dans un lien :

```
<a href="infos.php?langue=<?php echo $langue?>">...</a>
```

- Pour inclure dans un formulaire :

```
<input type = "hidden" name="langue" value ="<?php echo $langue ?>">
```


Maintenir l'état de l'interaction client-serveur

Solution I : transfert d'information par les paramètres GET ou POST

Exemple

- Pour simplifier on utilise des fonctions qui renvoient les différentes portions de texte dans trois langues différentes
- Toutes les fonctions sont regroupées dans un fichier commun, inclus dans toutes les pages

04-get-post/
> fonctions.php
> home.php
> imc.php
> infos.php
> affichage.php
> texte_langue.php

localhost/./04-get-post/home.php

Maintenir l'état de l'interaction client-serveur

Solution I : transfert d'information par les paramètres GET ou POST

- Inconvenient :
 - ▶ solution manuelle et lourde à implementer
 - ▶ redondante en terme de transfert : toute l'information à transférer fait l'aller-retour à chaque fois entre le client et le serveur
 - ▶ pas du tout adaptée au cas d'informations sensibles (comme les infos d'authentification)
 - elles seront visibles dans chaque url !
 - un autre utilisateur sur la même machine pourra les récupérer dans l'historique du browser par exemple

Maintenir l'état de l'interaction client-serveur

Solution 2 : stockage de cookies

- Le client demande une page et fournit une certaine information au serveur (e.g. la langue souhaitée)
- le serveur prepare une page de réponse dans laquelle il demande de stocker cette information sur la machine client (dans un cookie)
 - ▶ Le cookie sera transféré du client vers le serveur à chaque nouvelle connexion au site
 - ▶ \Rightarrow information disponible au serveur pendant toute l'interaction du client avec le site

Cookie

- Texte envoyé par le serveur et conservé par le navigateur dans un fichier sur la machine du client
- Envoyé dans les en-têtes de la réponse HTTP
- Un cookie a un **nom** et une **valeur**
 - ▶ La valeur est définie par le serveur à la création du cookie
- Une fois stocké chez le client, le cookie est automatiquement transmis par le navigateur au serveur à chaque requête du client
- Chaque cookie a une **date d'expiration**
 - Par défaut : la fin de la session du navigateur (quand le navigateur est fermé)
 - Peut être modifiée par le serveur web
- Un cookie peut-être refusé par le navigateur et peut aussi être effacé par l'utilisateur

Cookies en PHP

- En PHP, pour créer un cookie et l'insérer dans la réponse :

`setcookie(nom, valeur, date_expiration)`

`nom` : le nom du cookie

`valeur` : la chaîne stockée dans le cookie

`date_expiration` : secondes depuis le 01/01/1970

- utiliser la fonction `time()` (heure courante - en nombre de secondes)

`time()+60*60*24*30` fera expirer le cookie dans 30 jours

- ou `mktime()` (retourne un nombre de secondes correspondant à une heure, min, sec, mois, jour et année)

- Exemple : `setcookie("MyCookie", "Some Value", time() + 3600);`

Expiration dans une heure

Temps de référence: heure de la machine de l'utilisateur

- Invoquer la fonction **avant tout code HTML** inséré dans la page (y compris les espaces)

Cookies en PHP

- Les cookies transmis au serveur par le navigateur sont disponibles dans le tableau prédéfini `$_COOKIE`

```
echo $_COOKIE["nom_cookie"];
```

- Pour vérifier que le cookie existe :

```
isset($_COOKIE["nom_cookie"]);
```

il faut ensuite vérifier que sa valeur est non vide et correcte

- Pour détruire un cookie

```
unset($_COOKIE["nom_cookie"]);
```

▶ Attention :

- détruit la variable chez le serveur,
- ne comporte pas une demande d'effacer le cookie chez le client.

Maintenir l'état de l'interaction client-serveur

Solution 2 : stockage de cookies

Exemple : le même exemple de la langue, mais avec des cookies

- Chaque page commence par appeler une fonction qui teste si la langue a été transmise par le navigateur dans un cookie,
 - si ce n'est pas le cas, la fonction demande de choisir la langue
- Le script qui traite le formulaire de choix de la langue ajoute un cookie de nom "choix_langue" dans sa réponse, dont la valeur est la langue choisie
- Ce cookie est reçu par le navigateur et stocké chez le client
- À partir de ce moment le navigateur enverra le cookie à chaque nouvelle connexion, et les autres pages pourront le lire
- Si la langue est disponible dans un cookie, la page est affichée dans la langue demandée
 - **plus besoin d'ajouter la langue dans les liens ou formulaires : elle sera stockée chez le client !**

Maintenir l'état de l'interaction client-serveur

Solution 2 : stockage de cookies

Exemple : le même exemple de la langue, mais avec des cookies

Adaptation de la solution 1 :

- La fonction pour obtenir la langue, cherche l'information dans un cookie plutôt que dans les paramètres de la requête
- les différentes pages n'insèrent plus la langue dans les liens et formulaires
- la page principale (home.php) peut obtenir la langue par le cookie ou par les paramètres de GET (formulaire de choix)
 - ▶ dans le deuxième cas elle s'occupe de créer le cookie
- On ajoute la possibilité de changer de langue (pour détruire le cookie)
 - ▶ paramètre langue = ' ' envoyé à la page principale

04-cookie/
> fonctions.php
> imc.php
> infos.php
> home.php
> affichage.php

localhost/../04-cookie/home.php

Maintenir l'état de l'interaction client-serveur

- L'utilisation des cookies pour transmettre de l'information d'une page à l'autre :
 - ▶ évite quelques aller-retours des informations transmises
 - ▶ simplifie l'implémentation (pas inclusion "manuelle" dans les liens et formulaires)
- Inconvénients
 - ▶ a) toute l'information à partager est transférée à chaque connexion du client au serveur
 - ▶ b) encore inadaptée au cas d'informations sensibles (e.g. authentication)
 - les cookies sont des fichiers accessibles sur la machine du client à tout moment, aucune information sensible ne devrait y être stockée !
- Les sessions évitent (ou réduisent) les deux problèmes !

Maintenir l'état de l'interaction client-serveur : sessions

Solution 3 (*la* solution) : sessions

- Les sessions sont un mécanisme haut niveau pour transférer de l'information entre différentes requêtes HTTP au même serveur
- Idée :
 - ▶ L'information à transférer est stockée **sur le serveur** et elle est associée à un **identifiant de session** (un nombre aléatoirement généré par le serveur)
 - ▶ Le serveur peut demander la création d'une session, ce qui crée un nouvel identifiant et l'envoie au client
 - ▶ Le client renvoie au serveur cet identifiant à chaque nouvelle connexion
 - ▶ **Seul l'identifiant de session est physiquement transféré entre le client et le serveur**
 - ▶ Quand le client fait une requête portant un certain identifiant de session, le serveur récupère toutes les informations associées à cet identifiant
 - elles pourront être utilisées pour préparer la réponse
 - ▶ Le serveur peut enregistrer à tout moment des nouvelles informations associées à la session

Implementation des sessions

- L'implémentation des sessions est prise en charge par le langage
 - ▶ création de l'identifiant et de l'espace associé
 - ▶ gestion du transfert de l'identifiant entre client et serveur
- et elle est **transparente** à celui qui écrit le code
- Cette implémentation utilise en general un des mécanismes “bas niveau” :
 - ▶ Implementation typique : **identifiant de session stocké dans un cookie** chez le client (solution par défaut en PHP)
 - ▶ Implementation alternative : **identifiant de session inclus dans les paramètres de GET et/ou POST**
- Mais on peut utiliser les sessions sans (trop) se préoccuper de comment elles sont implémentées!
 - ▶ Interface de base :
 - créer une session,
 - accéder aux données associées à une session donnée,
 - détruire une session

Sessions en PHP

Créer une session :

`session_start()` (retourne `FALSE` en cas d'erreur)

- S'il n'y a pas de session en cours, crée une nouvelle session, i.e. si :
 - a) la page a été demandée sans identifiant de session ou
 - b) avec un identifiant de session absent chez le serveur,
 - crée un nouvel identifiant de session et espace de stockage associé
 - inclut cet identifiant dans la page de réponse (typiquement dans un cookie)
 - ▶ peut réutiliser l'identifiant envoyé dans le cas b)
- Si il y a une session en cours i.e. la page a été demandé avec un id de session présent chez le serveur,
 - charge les données de cette session dans le tableau prédéfini `$_SESSION`
- **Attention :** `session_start()` doit être appelé avant que tout code HTML soit produit dans la page

Sessions en PHP

Enregistrer une nouvelle variable de session

```
$_SESSION [nom_var] = valeur
```

Accéder aux variables enregistrées pour la session :

```
if( isset($_SESSION [nom_var]) ) ...
```

- on peut lire/modifier `$_SESSION` uniquement après `session_start()`

Supprimer une variable de session

```
unset($_SESSION [nom_var])
```

Accéder à l'identifiant de session (pas nécessaire en general si PHP est installé avec support transparent des sessions)

- l'identifiant de session est renvoyé par `session_id()`

Sessions en PHP

Terminer une session

```
$_SESSION = array();  
session_destroy();
```

Attention :

- `session_destroy()` efface les données de session sauvegardées dans un fichier du serveur (y compris l'ID de session) mais ne supprime pas les variables globales `$_SESSION`
- `$_SESSION = array();` les supprime manuellement pour qu'elles ne soient plus accessibles après la fin de la session

Sessions en PHP

Un premier exemple :

```
<?php
session_start();
if (!isset($_SESSION['count'])) {
    $_SESSION['count'] = 0;
} else {
    $_SESSION['count']++;
}
?>
```

Que fait ce script dans une page?

Sessions en PHP

Un premier exemple :

```
<?php
session_start();
if (!isset($_SESSION['count'])) {
    $_SESSION['count'] = 0;
} else {
    $_SESSION['count']++;
}
?>
```

Que fait ce script dans une page? Compte le nombre de visites d'un même client (navigateur)

Jusqu'à quand ?

Sessions en PHP

Un premier exemple :

```
<?php
session_start();
if (!isset($_SESSION['count'])) {
    $_SESSION['count'] = 0;
} else {
    $_SESSION['count']++;
}
?>
```

Deux événements peuvent arrêter le comptage :

- l'utilisateur ferme le navigateur
(id de session détruit chez le client)
- la session est détruite sur une autre page
(id de session détruit chez le serveur)

04-session-count/page.php

04-session-count/oublier.php

localhost:/.../page.php

Sessions et sécurité

L'utilisation des sessions réduit les failles de sécurité du site (les problèmes liés aux deux premières solutions) :

- Les données de session qui peuvent contenir des informations sensibles sont stockées chez le serveur (pas dans l'url , ni dans des cookies)
 - ▶ normalement pas accessibles par d'autres utilisateurs
- La seule information sensible encore présente chez le client est l'identifiant de session (qui donne accès à toutes les données de session)
- Toutefois l'id de session est une information sensible uniquement pendant la session d'un utilisateur (de `session_start()` à `session_destroy()`)
- Après déconnexion de l'utilisateur (`session_destroy()`), cela ne donne plus accès à quoi que ce soit chez le serveur
 - ▶ \Rightarrow à ce moment là, la disponibilité chez le client de l'identifiant de session (dans un cookie pas effacé, ou dans historique du browser) n'est pas critique

Sessions et sécurité

La seule vulnérabilité des sessions : le vol de session

- Le seul problème encore possible : **identifiant de session lu** par un utilisateur malveillant (personne ou programme) **pendant une session en cours**
- Plus difficile que dans les deux premières solutions mais encore possible :
 - ▶ si l'id de session est passé dans l'url il est visible pendant la session
 - ▶ s'il est stocké dans un cookie, un script (Javascript) introduit par un site malveillant peut s'exécuter sur le client et lire ses cookies, pendant la session (XSS)
- Il existe des solutions pour lutter contre le vol de session également...(e.g. cookies stockés en mode httpOnly qui les rendent inaccessibles au Javascript, ...)
- Une précaution habituelle (automatique dans la plus part des serveurs) : **timeout de session**

Sessions et sécurité

Timeout de session

- Avec les sessions la déconnexion de l'utilisateur devient une opération critique
- Si l'utilisateur quitte le site sans déconnexion, l'id de session reste disponible chez le client et encore actif chez le serveur
 - ▶ risque plus élevé de vol de session
- ▶ Solution : en général le serveur appelle automatiquement `session_destroy()` après une certaine période d'inactivité de la session
 - ▶ Inactivité : pas de pages demandées avec l'id de session
- ▶ Automatique pour les sessions PHP (et le délai peut être réglé)

Maintenir l'état de l'interaction client-serveur

Solution 3 : les sessions

- ⇒ On peut considérer les risques liés à l'utilisation des sessions relativement limités
- ⇒ Les sessions sont adaptées au partage d'information sensibles entre plusieurs pages
 - ▶ e.g. protection d'un site par authentification de l'utilisateur
 - ▶ rappel : une solution simpliste pour protéger une seule page ne nécessite pas de sessions