Atelier 3 : Technologie Web: PHP

Série 3

---

1 En-tête HTTP et authentification

Le web repose sur le protocole *HTTP*, HyperText Transfer Protocol, qui permet à un navigateur de communiquer avec un serveur. Cet échange est bien évidemment normé et codifié. Il est utilisé pour échanger toute sorte de données entre client HTTP et serveur HTTP. Une bonne connaissance de ce protocole permet de tirer partie de fonctionnalités avancées du couple navigateur/serveur. Pour plus d'informations, voir `https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol`.

1.1 Requêtes et réponses

La demande faite par un navigateur à un serveur web est appelée une requête (HTTP Request). Une requête typique consiste à demander le contenu d'une page ou d'une image. Suite à cette requête, le serveur retourne une réponse (HTTP Response). Les requêtes vont toujours par paires : la demande du client et la réponse du serveur. Requêtes et réponses sont composées d'une multitude d'instructions formant une en-tête.

1.1.1 Extension LiveHTTPHeaders

L'extension de Firefox LiveHTTPHeaders, télécharger gratuitement sur le site `http://livehttpheaders.mozdev.org`., permet de visualiser précisément les échanges HTTP entre un client et un serveur.

- Installer cette extension
- Le navigateur doit ensuite être fermé puis réouvert pour prendre en compte cette nouvelle extension.
- L'extension peut être lancée en passant par le menu Outils/En-têtes HTTP en direct.

1.1.2 Les requêtes clients

Voici la syntaxe générale d'une requête HTTP :

```
METHODE URL VERSION
EN-TETE : Valeur
.
.
.
EN-TETE : Valeur
Ligne vide
CORPS DE LA REQUETE
```

Trois catégories de données sont incluses dans l'en-tête de la requête :

- des informations sur la page cible et la version du protocole
  `GET /test.php HTTP/1.1`
  `Host: localhost`

- des précisions sur le navigateur

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.8.0.1)
Gecko/20060111 Firefox/1.5.0.1
Accept: text/xml,application/xml,application/xhtml+xml,
text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

- le type de la connexion

```
Keep-Alive: 300
Connection: keep-alive
```

Ces informations peuvent être récupérées au sein du script à l'aide de la fonction *getallheaders()* qui retourne un *array* associatif contenant l'ensemble des informations présentes dans l'en-tête.

Rappel :

```
getallheaders()
- Récupère tous les en-têtes de la requête HTTP
- Retourne un tableau associatif avec tous les en-têtes HTTP de la requête courante
  ou FALSE en cas d'échec.
- Cette fonction est un alias de la fonction apache_request_headers().
Description
array getallheaders ( void )
```

Ces informations permettent au script d'ajuster son comportement en fonction :

- du type du navigateur (User–Agent),
- des formats des fichiers acceptés (Accept),
- de la langue (Accept–Language).

Rappel :

```
La fonction strpos():
- Cherche la position numérique de la première occurrence dans une chaîne.
- Retourne FALSE si l'occurrence n'a pas été trouvée.
Description
strpos(string,findMe, start)
string:La chaîne dans laquelle on doit chercher.
findMe:Indique la chaîne à trouver
start:Optionnel. Indique où commencer la recherche
```

Voici un exemple d'utilisation de la fonction *getallheaders()* :

```
$header = getallheaders();
if (strpos($header["User-Agent"], "Firefox") > 0) {
echo("Vous utilisez Firefox");
}
elseif (strpos($header["User-Agent"],"Chrome") > 0) {
echo("Vous utilisez Google Chrome");
}
else {
echo("Vous utilisez un autre Navigateur");
}
echo("<br/><br/><i>".$header["User-Agent"]."</i>");
```

Exercice 1
En utilisant la boucle foreach, afficher tous les en-têtes de la requête HTTP.

1.1.3 Les réponses

L'en-tête de la réponse donne des informations sur :
- le code réponse,
  `HTTP/1.1 200 OK`
- le serveur distant,
  `Date: Sun, 13 Dec 2016 23:43:24 GMT`
  `Server: Apache/2.2.22 (Win64) PHP/5.3.13`
  `X-Powered-By: PHP/5.3.13`
- le contenu,
  `Content-Length: 1539`
  `Content-Type: text/html`
- la connexion,
  `Keep-Alive: timeout=15, max=100`
  `Connection: Keep-Alive`

La fonction *apache_ response_ headers()* retourne un tableau associatif contenant tous les éléments de l'en-tête de réponse HTTP en cas de réussite ou FALSE si une erreur survient.
Rappel :

```
apache_response_headers - Récupère tous les en-têtes de réponse HTTP
Desscription:
array apache_response_headers ( void )
```

Exercice 2
Exécutez le script suivant :

```
<?php
echo "<pre>";
print_r(apache_response_headers());
echo "</pre>";
?>
```

1.2 Fonction header()

PHP permet de modifier l'en-tête de la réponse à l'aide de la fonction header(). L'emplacement de cette fonction au sein d'un script est extrêmement important : la fonction *header()* doit absolument être placée avant le premier affichage de la page.
Rappel :

```
header - Envoie un en-tête HTTP
Description
void header(string, replace,http_response_code)
string:l'en-tête.
replace:optionnel. il indique si la fonction header() doit remplacer
        un en-tête précédemment émis, ou bien ajouter un autre en-tête du même type.
http_response_code:force le code réponse HTTP à la valeur spécifiée.
```

Exemples : Que remarquez-vous après l'exécution de ces 3 scripts ?
script 1 :

```
Bonjour
<?php
header("Text: coucou");
?>
```

script 2 :

```
<?php
print("Bonjour");
header("Text: coucou");
?>
```

script 3 :

```
<?php
header("Text: Salam");
echo("Bonjour");
?>
```

Il est également possible de modifier le contenu des instructions en les écrasant :

```
<?php
print("Bonjour");
header("X-Powered-By: PHP/8 !!!");//Modification de l'instruction X-Powered-By
?>
```

La fonction `header()` permet également de donner des instructions telle qu'une demande de redirection avec l'instruction Location.

```
<?php
//Demande de redirection vers une autre page:page de l'EMI ici
header("Location: http://www.emi.ac.ma");
//Assurez-vous que la suite du code ne soit pas exécutée une fois la redirection effectuée.
exit();
?>
```

1.2.1 Boîte de téléchargement

Si vous voulez que vos utilisateurs reçoivent une alerte pour sauver les fichiers générés, comme si vous génériez un fichier PDF, vous pouvez utiliser l'en-tête `Content-Disposition` pour fournir un nom de fichier par défaut, à afficher dans le dialogue de sauvegarde.
Utilisation de `header()` pour générer un fichier de type PDF ou d'un autre type.

```
<?php
// Vous voulez afficher un pdf
header('Content-type: application/pdf');

// Il sera nommé downloaded.pdf
header('Content-Disposition: attachment; filename="downloaded.pdf"');

// Le source du PDF original.pdf
readfile('original.pdf');//Par exemple readfile('C:\Users\hp\Desktop\tonFichier.pdf');
?>
```

1.2.2 Directives concernant la mise en cache

Les scripts PHP génèrent souvent du HTML dynamiquement, qui ne doit pas être mis en cache, ni par le client, ni par les proxy intermédiaires. On peut forcer la désactivation du cache de nombreux clients et proxy avec :

```
<?php
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT"); // Date dans le passé
?>
```

1.3 Page d'erreur

Le code retour 200 présent dans l'en-tête de réponse *(HTTP/1.1 200 OK)* signifie que le navigateur n'a rencontré aucun problème pour traiter la requête. Si vous lancez une page n'existant pas sur le serveur, le code d'erreur passe à 404.
Le protocole HTTP prévoit un certain nombre de codes d'erreur tous commençant par 4xx. Voici les différents codes d'erreur client :

Code d'erreur	Signification
400	Erreur de syntaxe dans l'adresse du document
401	Pas d'autorisation d'accès au document
402	Accès au document soumis au paiement
403	Pas d'autorisation d'accès au serveur
404	La page demandée n'existe pas
405	Méthode de requête du formulaire non autorisée
406	Requête non acceptée par le serveur
407	Autorisation du proxy nécessaire
408	Temps d'accès à la page demandée expiré
409	L'utilisateur doit soumettre à nouveau avec plus d'infos
410	Cette ressource n'est plus disponible
411	Le serveur a refusé la requête car elle n'a pas de longueur
412	La pré condition donnée dans la requête a échoué
413	L'entité de la requête était trop grande
414	L'URI de la requête était trop longue
415	Type de média non géré

L'exemple suivant indique que la page n'existe pas uniquement aux visiteurs utilisant Firefox (utilisez trois navigateurs Firefox, Google Chrome et IE) :

```
<?php
$header = getallheaders();
if (strpos($header["User-Agent"], "Firefox") > 0) {
header("HTTP/1.1 404 Not Found");
echo("La page n'existe pas");
}
elseif (strpos($header["User-Agent"], "Chrome") > 0) {
echo("Coucou les utilisateurs Google Chrome");
}
else
{
echo("Coucou les utilisateurs IE");
}
?>
```

1.4 Authentification

Le protocole HTTP inclut une gestion d'authentification. Cette authentification met en oeuvre le code retour 401. En recevant ce code retour le navigateur sait qu'il doit ouvrir une petite fenêtre afin de permettre à l'utilisateur de renseigner son identifiant et son mot de passe. En plus du code d'erreur, le serveur doit également préciser le nom de l'espace (realm) à protéger avec l'instruction

WWW–Authenticate.
Voici un exemple d'une demande d'authentification, à exécuter, qui peut donc être codée de la manière suivante :

```
<?php
header('WWW-Authenticate: Basic realm="Zone protégée"');
header("HTTP/1.1 401 Unauthorized");
?>
```

Une fois le bouton OK validé, Firefox transmet une requête contenant vos informations d'identification. Cela se traduit par l'ajout au sein de l'en-tête d'une instruction Authorization correspondant l'encodage en *base 64*, cliquez ici pour plus de détails `http://fr.wikipedia.org/wiki/Base64`, de votre identifiant suivi de votre mot de passe :

```
identifiant:mot de passe
```

Vous pouvez vérifier qu'il s'agit bien d'un encodage en base 64 en utilisant la fonction *base64_ decode()* sur la valeur associée à *Authorization.*
Rappel :

```
base64_decode - Décode une chaîne en MIME base64
Description
string base64_decode (data,strict)

data: Les données à décoder.
strict: Retourne FALSE si l'entrée contient des caractères hors de l'alphabet base64.
```

Pour cela, essayez d'exécuter le script suivant :

```
<?php
echo base64_decode("ZGFyb3VpY2g6YXppeg==");
//il affichera votre identifiant:votre mot de passe
?>
```

Dans le cas d'une authentification, *$_ SERVER* dispose de deux éléments en plus, *$_ SERVER["PHP_ AUTH_ USER"]* et *$_ SERVER["PHP_ AUTH _ PW"]* qui contiennent respectivement les valeurs que vous avez précisées pour l'identifiant et le mot de passe.
Le principe consiste maintenant à utiliser ces deux valeurs pour vérifier si elles sont correctes. Deux cas apparaissent alors :

- la vérification a échoué et vous proposez à nouveau la demande d'authentification,
- la vérification a réussi et la suite du script est proposée.

1.5 Script d'authentification

```
if ($_SERVER["PHP_AUTH_USER"]!="este" || $_SERVER["PHP_AUTH_PW"]!="licenceProISIL15")
{
header('WWW-Authenticate: Basic realm="Zone protégée"');
header("HTTP/1.1 401 Unauthorized");
exit(0);
}
echo("Bienvenue dans la zone sécurisée");
```

Dans le cas où un internaute ne parvenant pas à s'authentifier peut à tout moment cliquer sur le bouton Annuler et obtenir un message d'aide. Essayez d'éxecuter le script suivant (*test.php*) :

```
<?php
header('WWW-Authenticate: Basic realm="Zone protégée"');
header("HTTP/1.1 401 Unauthorized");
print("Veuillez contacter votre administrateur système ");
print("pour obtenir vos identifiants.");
exit(0);
echo("Bienvenue dans la zone sécurisée");
?>
```

L'exemple suivant propose un lien vers un script *quitter.php* chargé de permettre à un internaute de se déconnecter, en renvoyant un code d'erreur 401.
test.php

```
<?php
if ($_SERVER["PHP_AUTH_USER"]!="test" || $_SERVER["PHP_AUTH_PW"]!="test")
{
header('WWW-Authenticate: Basic realm="Zone protégée"');
header("HTTP/1.1 401 Unauthorized");
exit (0);
}
echo("Bienvenue dans la zone sécurisée<br/><br/>");
echo("<a href='quitter.php'>quitter l'application</a>");
?>
```

quitter.php

```
<?php
header('WWW-Authenticate: Basic realm="Zone protégée"');
header("HTTP/1.1 401 Unauthorized");
?>
```

Exercice 3

1. Une fois identifié, cliquez sur le lien pour basculer sur quitter.php et constater que la fenêtre d'authentification est bien proposée.
2. Modifiez maintenant l'url dans la barre de navigation afin d'indiquer `http://localhost/test.php`. Vous constatez à nouveau que la déconnexion a bien fonctionné.

2 Vérifications simples en PHP

Les tests *JavaScript* ne doivent en aucun cas être considérés comme un gage de sécurité absolu. Vous savez en effet qu'il est possible de composer soi-même un lien permettant de transmettre des données. En utilisant cette technique, l'utilisateur passe à travers les tests *JavaScript* et force la transmission des informations qui lui conviennent. Il est également important de savoir que la technologie *JavaScript* peut être désactivée au niveau du navigateur. Au bilan, vos vérifications *JavaScript* doivent plutôt être envisagées comme des avertissements. L'utilisateur est mis au courant

que ses données risquent d'être refusées au niveau du script et qu'il lui faudra alors revenir en arrière afin de modifier les données du formulaire. Nous sommes donc ici plus dans une optique de confort, d'ergonomie et de gain de temps.

2.0.1 Fonction empty()

La fonction empty() est la plus importante. Elle retourne *true* si la variable que vous lui avez adressée en argument est vide ou inexistante.
Cette fonction retourne *FALSE* si le paramètre passé existe et est non-vide, et dont la valeur n'est pas zéro. Ce qui suit est considéré comme étant vide :

- "" (une chaîne vide)
- 0 (0 en tant qu'entier)
- 0.0 (0 en tant que nombre à virgule flottante)
- "0" (0 en tant que chaîne de caractères)
- *NULL*
- *FALSE*
- *array()* (un tableau vide)
- $*var* ; (une variable déclarée, mais sans valeur).

Par exemple pour les champs titre, année, description et pays.

```
if (empty($_REQUEST['titre'])==true)
//ou la version simplifiée if (empty($_REQUEST['titre']))
{
print("ERREUR : le champ titre n'a pas été rempli");
exit();
}
```

Remarque
$_REQUEST est une superglobale, disponible dans tous les contextes du script. Il n'est pas nécessaire de faire global $variable ; pour y accéder dans les fonctions ou les méthodes.
Rappel

```
$_REQUEST - Variables de requête HTTP
Description
Un tableau associatif qui contient par défaut le contenu
des variables $_GET, $_POST et $_COOKIE.
```

2.0.2 Fonction exit()

La fonction exit() est utilisée pour interrompre et quitter définitivement le *script*. Vous pouvez aussi faire appel à la fonction die() qui quitte également le programme tout en affichant à l'écran la chaîne de caractères qui lui est adressée en paramètre.

```
if (empty($_REQUEST['titre']))
{
die("ERREUR : le champ titre n'a pas été rempli");
}
```

2.0.3 Fonction return()

Remarque
Vous pourrez trouver sur le Web des scripts qui utilisent la fonction return() plutôt que les fonctions

exit() ou die(). Cette fonction est pourtant différente dans le sens où return() ne quitte que le *script* en cours. Vous verrez en effet par la suite qu'un *script* peut en inclure d'autres. Pour être sûr de quitter définitivement le *script*, il vaut donc mieux utiliser exit() ou die().

Exercice 4

1. Exécuter les deux scripts suivants concernant la fonction return()
   a.php

   ```
   <?php
   include("b.php");
   echo "a";
   ?>
   ```

   b.php

   ```
   <?php
   echo "b";
   return;
   ?>
   ```

2. Exécuter les deux scripts suivants concernant la fonction exit()
   a.php

   ```
   <?php
   include("b.php");
   echo "a";
   ?>
   ```

   b.php

   ```
   <?php
   echo "b";
   exit;
   ?>
   ```

3. Que remarquez-vous ?

2.0.4 Fonction trim()

Rappel

La fonction trim() supprime les espaces (ou d'autres caractères) en début et fin de chaîne. Voici sa sytanxe :

```
string trim (str, character_mask)
```

Par exemple, pour la vérification du champ titre

```
$_REQUEST['titre'] = trim($_REQUEST['titre']);
if (empty($_REQUEST['titre']))
{
die("ERREUR : le champ titre n'a pas été rempli");
}
```

2.0.5 Fonction isset()

Les paramètres, à savoir le genre, la couleur et le sous-titre, ont la particularité de ne pas être propagés par le formulaire si aucune valeur n'est cochée ou sélectionnée. Votre test ne porte donc plus sur le caractère vide ou non de la variable, mais sur son existence. La fonction isset() retourne *true* si une variable existe (et est différente de NULL) et *false* dans le cas contraire. Par exemple pour tester ces trois paramètres : le genre, la couleur et le sous-titre.

```
if (isset($_REQUEST['genre']))
{
die("ERREUR : aucun genre n'a été sélectionné");
}
```

Remarque :
Même si la fonction isset() peut la plupart du temps être remplacée par empty(), il est préférable d'être le plus précis possible dans le choix et l'utilisation de vos fonctions.

Exercice 5
Vous allez donc aller plus loin dans la validation des paramètres et contrôler les points suivants :

1. $_REQUEST['titre'] contient plus de deux caractères ;
2. $_REQUEST['annee'] comprise entre 2000 et 2014 ;
3. $_REQUEST['genre'] contient au moins une des valeurs suivantes : policier, sf, com ;
4. $_REQUEST['description'] contient entre 10 et 500 caractères ;
5. $_REQUEST['couleur'] vaut 0 ou 1 ;
6. $_REQUEST['pays'] contient une des valeurs suivantes : ma, tun, maur, fr, usa ;
7. $_REQUEST['soustitre'] contient au moins une des valeurs suivantes : fr, gb, ar.

2.0.6 Sketch de la réponse :

Validation du titre

```
if (strlen($_REQUEST['titre'])>2)
```

Validation de la description

```
if( strlen($_REQUEST['description'])>10 && strlen($_REQUEST['description'])<500)
```

Validation du champ année

```
if ($_REQUEST['annee']>=2000 && $_REQUEST['annee']<=2014)
```

Validation de la couleur

```
if ($_REQUEST['couleur']==0 || $_REQUEST['couleur']==1)
```

Validation du pays

```
if ($_REQUEST['pays']=='ma' || $_REQUEST['pays']=='tun' || $_REQUEST['pays']=='maur')
|| $_REQUEST['pays']=='fr')|| $_REQUEST['pays']=='usa')
```

Version plus élégante de la vérification du pays

```
$tab_pays = array('ma','tun','maur','fr','usa');
if (in_array($_REQUEST['pays'],$tab_pays)==false)
$erreur .= "- le champ pays est mal rempli<br/>";
```

Validation du genre

```
$tab_genre = array('policier','sf','com');
foreach ($_REQUEST['genre'] as $tmp) {
if (in_array($tmp,$tab_genre)==false)
$erreur .= "- le genre $tmp n'est pas correct<br/>";
}
```

Validation du sous–titre

```
$tab_soustitre = array('fr','gb','ar');
foreach ($_REQUEST['soustitre'] as $tmp) {
if (in_array($tmp,$tab_soustitre)==false)
$erreur .= "- le sous-titre $tmp n'est pas correct<br/>";
}
```

Ces deux tests sont valides mais insuffisants car vous ne testez pas le fait que $_REQUEST['genre'] et $_REQUEST['soustitre'] sont bien des tableaux et qu'ils contiennent au moins un élément. Cette vérification est importante dans la mesure où foreach() génère une erreur lorsque la variable qui lui est adressée en paramètre n'est pas un tableau. Renforcez la vérification en utilisant la fonction is_array(), qui indique si la variable qui lui est transmise en paramètre est bien un tableau, et la fonction count(), ou son alias sizeof(), qui retourne le nombre d'éléments contenus dans un tableau.

Version plus complète de la validation du genre

```
if (is_array($_REQUEST['genre'])==false || count($_REQUEST['genre'])<=1) {
$erreur .= "- le genre n'est pas correct<br/>";
}
else {
$tab_genre = array('policier','sf','com');
foreach ($_REQUEST['genre'] as $tmp) {
if (in_array($tmp,$tab_genre)==false)
$erreur .= "- le genre $tmp n'est pas correct<br/>";
}
}
```

Finalement, on regroupe l'ensemble de ces tests qui vous informe de la validité des paramètres, au sein d'une fonction : *verif()*. Cette fonction retourne *true* si aucune erreur n'est détectée et *false* dans le cas contraire.

```
<?php
function verif(){
$erreur = "";
if (strlen($_REQUEST['titre'])<=2)
$erreur .= "- le champ titre est mal rempli<br/>";
if( strlen($_REQUEST['description'])<=10 ||strlen($_REQUEST['description'])>=500)
$erreur .= "- le champ description est mal rempli<br/>";
if ($_REQUEST['annee']<2000 || $_REQUEST['annee']>2014)
$erreur .= "- le champ année est mal rempli<br/>";
if ($_REQUEST['couleur']!=0 && $_REQUEST['couleur']!=1)
$erreur .= "- le champ couleur est mal rempli<br/>";
```

```
$tableau_pays = array('ma','tun','maur','fr','usa');
if (in_array($_REQUEST['pays'],$tableau_pays)==false)
$erreur .= "- le champ pays est mal rempli<br/>";
if (is_array($_REQUEST['genre'])==false || count($_REQUEST['genre'])<1) {
$erreur .= "- le genre n'est pas correct<br/>";
}
else {
$tableau_genre = array('policier','sf','com');
foreach ($_REQUEST['genre'] as $tmp) {
if (in_array($tmp,$tableau_genre)==false)
$erreur .= "- le genre $tmp n'est pas correct<br/>";
}
}
if (is_array($_REQUEST['soustitre'])==false || count($_REQUEST['soustitre'])<1) {
$erreur .= "- le sous-titre n'est pas correct<br/>";
}
else {
$tableau_soustitre = array('fr','gb','ar','es');
foreach ($_REQUEST['soustitre'] as $tmp) {
if (in_array($tmp,$tableau_soustitre)==false)
$erreur .= "- le sous-titre $tmp n'est pas correct<br/>";
}
}
if (!empty($erreur)) {
print($erreur);
return false;
}
return true;
}
if (verif()==false) exit(0);
echo("<b>Titre</b> : ".$_REQUEST['titre']."<br/>");
echo("<b>Année</b> : ".$_REQUEST['annee']."<br/>");
$str_genre = join(',',$_REQUEST['genre']);
echo("<b>Genre</b> : ".$str_genre."<br/>");
echo("<b>Description</b> : ".$_REQUEST['description']."<br/>");
echo("<b>Couleur</b> : ".$_REQUEST['couleur']."<br/>");
echo("<b>Pays</b> : ".$_REQUEST['pays']."<br/>");
$str_soustitre = join(',',$_REQUEST['soustitre']);
echo("<b>Sous titres</b> : ".$str_soustitre."<br/>");
?>
```

Exercice 6
Modifier l'ancien script *form.php* pour utiliser cette fonction *verif()*.

3 Les expressions régulières

Les expressions régulières (également appelées regular expressions ou regexp) permettent de réaliser des tests beaucoup plus fins et complexes. L'idée ici est d'utiliser la fonction preg_match()

qui retournera *true* si la variable que vous lui adressez en second paramètre « correspond » à l'expression régulière que vous lui avez transmise en premier paramètre (que vous appellerez désormais *pattern*). En terme technique, vous vérifiez si ces deux arguments se correspondent. L'opération générale de recherche de correspondance est appelée le *pattern matching*.
Un *pattern* est une simple chaîne de caractères entourée par le caractère / qui va vous permettre de préciser très finement le motif que vous recherchez dans la variable à tester.
Par exemple, vous voulez vérifier que la variable $*str* contient quelque part dans son contenu la chaîne "HTTP", écrivez :

```
$str="HTTP a été inventé par Tim Berners-Lee";
if (preg_match("/HTTP/", $str)) {
echo("str contient HTTP");
}
else {
echo("str ne contient pas HTTP");
}
```

- Si $*str* contient "blabla http blabla", la fonction preg_match() retourne *true*.
- Si $*str* contient "bla ht-tp bla" ou "bla hTTp bla", la fonction preg_match() retournera *false*.

Remarques

a. La fonction preg_match() fait en effet partie des fonctions sensibles à la casse.

b. L'option *i* placée derrière le deuxième / permet de réaliser une comparaison de type *case insensitive* (qui n'est pas sensible à la casse).

Voici un exemple d'un test insensible à la casse

```
$str="HTTP a été inventé par Tim Berners-Lee";
if (preg_match("/a/i",$str)) {
echo("str contient a ou A");
}
else {
echo("str ne contient pas a ou A");
}
```

Pour vérifier que "HTTP" se trouve en début de chaîne, vous disposez de l'accent circonflexe ($\hat{}$) qui, dans le cadre d'une *regexp*, correspond à un début de ligne. Ecrivez le test suivant :

```
$str="HTTP a été inventé par Tim Berners-Lee";
if (preg_match("/^HTTP/",$str)) {
echo("str commence HTTP");
}
else {
echo("str ne commence pas HTTP");
}
```

Pour vérifier que "HTTPS" se trouve à la fin de ligne de chaîne, vous disposez du caractère $. Vous l'utlisez comme suit :

```
$str="Un serveur HTTP utilise alors par défaut le port 80, 443 pour HTTPS";
if (preg_match("/HTTPS$/",$str)) {
echo("str contient à la fin de ligne HTTPS");
```

```
}
else {
echo("str ne contient pas à la fin de ligne HTTPS");
}
```

Je signale que le pattern ^http$ permet par conséquent de vérifier que $*str* contient exactement la chaîne "http". Dans un tel cas, le test *if* ($str=="http") reste cependant largement plus rapide et pertinent qu'*if* (*preg_match*("/^http$/",$*str*)).
Le point (.) a aussi un rôle spécial au sein d'un *pattern*, il correspond à un (et un seul) caractère, quel qu'il soit. Testez les deux exemples suivants :

- *preg_match*("/ht.tp/","htatp") retourne *true* ;
- *preg_match*("/ht.tp/","htap") retourne *false*.

Le pattern ^..$ est donc un moyen de tester que $str contient deux caractères. Le test *if* (*strlen*($str)==2) est cependant plus optimisé pour ce type de vérification.
Pour vérifier que $str contient bien le point, vous devez le « protéger » avec la barre oblique inversée (\ ).
Le pattern www\. permet de vérifier que $str contient "www." et non " www#". Nous devons également utiliser la barre oblique inversée pour protéger les caractères suivants :

```
/ ^ [ ] $ ( ) | * { } + ? { \ '
```

Les crochets nous permettent de regrouper un ensemble de caractères qui devront apparaître au moins une fois dans $str. Voici des exemples

- *preg_match*("[aeiouy]",$str) vérifie que $str contient au moins une voyelle ;
- *preg_*match("[aeiouy]r",$str) vérifie que $str contient au moins une fois une voyelle suivie de la lettre r ;

Certains regroupements de caractères peuvent être simplifiés à l'aide des expressions spéciales :

- \w : pour des lettres, des chiffres ou le caractère _ ;
- \d : pour des chiffres ;
- \s : pour des caractères d'espacement : \n, ,\t,.

Testez l'exemple suivant : Le pattern ^\d\d\s\d\d\s\d\d\s\d\d\s\d\d\$ peut être utilisé pour vérifier que $*str* correspond à un numéro de téléphone (ex : 06 61 00 73 40).
Vous pouvez une nouvelle fois simplifier ce pattern en utilisant les expressions de fréquence suivantes :

- "(to) ?" : la chaîne contient une fois au maximum la chaîne "to" ;
- "(to)+" : la chaîne contient une fois au minimum la chaîne "to" ;
- "(to)*" : la chaîne contient zéro ou plusieurs fois la chaîne "to" ;
- "(to){2}" : la chaîne contient deux chaînes "to" qui se suivent ;
- "(to){2,5}" : la chaîne contient entre deux et cinq chaînes "to" qui se suivent.

Voyez quelques exemples :

- Votre pattern « téléphonique » devient donc ^(\d\d\s){4}(\d\d)$.
- "^\w+\$" pour vérifier que $str ne contient que des lettres ;
- "^[\w\s,\.]+$" pour vérifier que $str n'est composée que de lettres, de chiffres, de caractères blancs ou de points et de virgules.

Les crochets permettent également de définir des intervalles :

- [a-z]pour des lettres minuscules ;
- [A-Z]pour des lettres majuscules ;
- [0-9]pour des chiffres de 0 à 9 ;
- [0-5]pour des chiffres de 0 à 5 ;
- [d-g]pour des lettres minuscules de d à g.

Le caractère | au sein d'un *pattern* prend la signification d'un OU :

- "(fr|gb|es)" pour vérifier que la chaîne contient fr ou gb ou es ;
- "^(fr|gb|es)$" pour vérifier qu'elle est égale à fr, gb ou es.

Exercice 7

Ecrire le *pattern* qui vous permettra de tester qu'une chaîne est une adresse de courriel valide. Pour simplifier et ne pas vous perdre dans des détails, considérez qu'une adresse de courriel est construite de la manière suivante : le nom de l'utilisateur (caractères alphanumériques ainsi que les caractères - .), l'arobase (@), le nom de domaine (caractères alphanumériques ainsi que les caractères - .), un point et l'extension du domaine (de deux à trois caractères).

Remarque :

- N'utilisez pas preg_match() si vous voulez uniquement savoir si une chaîne est contenue dans une autre.
- Utilisez dans ce cas les fonctions strpos()(Cherche la position de la première occurrence dans une chaîne) ou strstr()(Trouve la première occurrence dans une chaîne), qui sont beaucoup plus rapides.

4 L'envoi d'un formulaire par courriel

L'envoi par courriel d'informations en provenance d'un formulaire est certainement l'utilisation la plus répandue de *PHP*.

4.1 Configuration requise

Alors que l'envoi de courriel ne nécessite aucune configuration particulière si vos scripts sont exécutés chez un hébergeur, il en va tout autrement s'ils sont placés sur votre machine.
Dans le fichier *php.ini*, vous parcourrez la section suivante :

```
[mail function]
; For Win32 only.
SMTP = localhost
; For Win32 only.
;sendmail_from = you@yourdomain
```

La ligne *SMTP = localhost* indique que *PHP* est paramétré pour utiliser votre propre machine (*localhost*) pour l'envoi des courriels.
La ligne *;sendmail_from = you@yourdomaine* : Le point-virgule initial signifie qu'elle est commentée et qu'elle n'est donc pas prise en compte. Cette directive permet de préciser l'origine des courriels envoyés depuis votre machine. Veillez donc à supprimer le point-virgule et à renseigner votre adresse *e-mail*.

5 Mail Texte

L'envoi de courriels en *PHP* est simple. Il suffit d'utiliser la fonction mail().
Les arguments de cette fonction sont :

- l'adresse de destination ;
- le titre du message ;
- le contenu du message ;
- d'éventuelles options.

En général, la fonction mail() est appelée de la manière suivante :

```
mail($destinataire,$titre,$message);
```

avec

- $email est l'e-mail de la personne qui va recevoir le courriel, par exemple $destinataire = "este@domaine.ma" ;.
- $titre est le titre de l'e-mail, par exemple $titre = "réponse au formulaire" ;.
- $message est le corps du message qui va contenir toutes les informations.
- La fonction mail() retourne un booléen qui indique si l'envoi s'est bien déroulé. Ce statut ne concerne que l'envoi, il n'indique en aucun cas le fait que le courriel est bien arrivé dans la boîte du destinataire.

Commencez par construire le contenu du courriel :

```
$message = "";
<!--La première ligne initialise la variable (=)-->
$message .= "Titre : ".$_REQUEST['titre']."\n";
<!--le caractère \n à la fin de chaque ligne.
Il s'agit du caractère représentant un saut de ligne.-->
```

```
<!--les lignes suivantes ajoutent des informations à la fin de la variable (.=).-->
$str_genre = join(',',$_REQUEST['genre']);
$message .= "Genre : ".$str_genre."\n");
$message .= "Description : ".$_REQUEST['description']."\n");
$message .= "Couleur : ".$_REQUEST['couleur']."\n");
$message .= "Pays : ".$_REQUEST['pays']."\n");
$str_soustitre = join(',',$_REQUEST['soustitre']);
$message .= "Sous titres : ".$str_soustitre."\n";
```

Votre script prend finalement la forme suivante :

```
<?php
function verif()
{
$erreur = "";
if (strlen($_REQUEST['titre'])<=2)
$erreur .= "- le champ titre est mal rempli<br/>";
if( strlen($_REQUEST['description'])<=10 ||
strlen($_REQUEST['description'])>=500)
$erreur .= "- le champ description est mal rempli<br/>";
if ($_REQUEST['annee']<1930 || $_REQUEST['annee']>2006)
$erreur .= "- le champ année est mal rempli<br/>";
if ($_REQUEST['couleur']!=0 && $_REQUEST['couleur']!=1)
$erreur .= "- le champ couleur est mal rempli<br/>";
$tableau_pays = array('fr','us','gb');
if (in_array($_REQUEST['pays'],$tableau_pays)==false)
$erreur .= "- le champ pays est mal rempli<br/>";
if (is_array($_REQUEST['genre'])==false ||
count($_REQUEST['genre'])<1) {
$erreur .= "- le genre n'est pas correct<br/>";
}
else {
$tableau_genre = array('policier','sf','culte');
foreach ($_REQUEST['genre'] as $tmp) {
if (in_array($tmp,$tableau_genre)==false)
$erreur .= "- le genre $tmp n'est pas correct<br/>";
}
}
if (is_array($_REQUEST['soustitre'])==false ||
count($_REQUEST['soustitre'])<1) {
$erreur .= "- le sous-titre n'est pas correct<br/>";
}
else {
$tableau_soustitre = array('fr','gb','es');
foreach ($_REQUEST['soustitre'] as $tmp) {
if (in_array($tmp,$tableau_soustitre)==false)
$erreur .= "- le sous-titre $tmp n'est pas correct<br/>";
}
}
if (!empty($erreur)) {
```

```
print($erreur);
return false;
}
return true;
}
if (verif()==false) exit(0);
print("<b>Titre</b> : ".$_REQUEST['titre']."<br/>");
print("<b>Année</b> : ".$_REQUEST['annee']."<br/>");
$str_genre = join(',',$_REQUEST['genre']);
print("<b>Genre</b> : ".$str_genre."<br/>");
print("<b>Description</b> : ".$_REQUEST['description']."<br/>");
print("<b>Couleur</b> : ".$_REQUEST['couleur']."<br/>");
print("<b>Pays</b> : ".$_REQUEST['pays']."<br/>");
$str_soustitre = join(',',$_REQUEST['soustitre']);
print("<b>Sous titres</b> : ".$str_soustitre."<br/>");
$destinataire = "fx@kernix.com";
$titre = "réponse au formulaire";
$message = "";
$message .= "Titre : ".$_REQUEST['titre']."\n";
$message .= "Année : ".$_REQUEST['annee']."\n";
$str_genre = join(',',$_REQUEST['genre']);
$message .= "Genre : ".$str_genre."\n";
$message .= "Description : ".$_REQUEST['description']."\n";
$message .= "Couleur : ".$_REQUEST['couleur']."\n";
$message .= "Pays : ".$_REQUEST['pays']."\n";
$str_soustitre = join(',',$_REQUEST['soustitre']);
$message .= "Sous titres : ".$str_soustitre."\n";
if (mail($destinataire,$titre,$message)==true) {
print("<hr/>Les informations ont bien été transmises.");
}
else {
die("<hr/>L'envoi du courriel a échoué.");
}
?>
```

5.1 Mail HTML

L'émission d'un courriel au format *HTML* nécessite la mise en oeuvre du quatrième paramètre de la fonction mail(). Ce paramètre est une chaîne de caractères contenant des informations qui seront ajoutées à l'en-tête (*header*) du message.
Un courriel est composé de deux parties principales : l'en-tête et le corps du message. L'en-tête contient un certain nombre d'informations sur le courriel : son origine, le destinataire, le format, le sujet, l'heure d'envoi, le logiciel d'envoi.
L'organisation des données dans cet en-tête est très simple :

```
champs1: valeur1
champs2: valeur2
etc.
```

Voyez cet exemple d'en-tête de courriel :

```
From: Mohammed@toto.ma
To: Ahmed@titi.ma
Subject: retour de vacances
X-Mailer: Microsoft Outlook Express
```

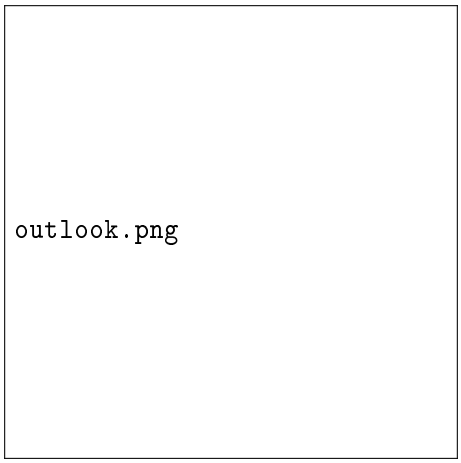

FIGURE 1 – Exemple de l'en-tête d'un courriel, vu avec Outlook Express

La commande mail() compose donc un en-tête par défaut en intégrant les données passées en paramètres : le destinataire (*To :*), le sujet (*Subject :*). Le quatrième paramètre permet d'ajouter certaines informations à cet en-tête.
Il est courant qu'une personne recevant un courriel émanant d'un script PHP ne sache pas qui lui a envoyé.
En ajoutant "*From : Mohammed@toto.ma*" comme quatrième paramètre, le destinataire est en mesure de connaître l'origine du courriel.
Les lignes contenues dans le quatrième paramètre doivent être séparées par des sauts de ligne : \n.
Ajoutez également une adresse de réponse (*Reply-To*) différente de l'adresse de l'émetteur (*From)* :

```
mail("Ahmed@host.com","retour de vacances","excellent", "From: Mohammed@toto.ma\n
Reply-To: khadija@toto.ma");
```

C'est aussi grâce à l'en-tête que vous allez être en mesure de dire au gestionnaire de courriels que le message qu'il a re{ccu doit être considéré comme une page *HTML*. Les deux lignes suivantes dans l'en-tête indiquent que le courriel n'est pas du simple texte.

```
MIME-Version: 1.0
Content-Type: multipart/alternative; boundary=B97C1230
```

Le corps du courriel doit lui aussi être construit de manière spécifique.
Le contenu HTML doit être précédé de :

```
This is a multi-part message in MIME format.
--B97C1230
Content-Type: text/html; charset="iso-8859-1"
... et suivi de :
--B97C1230--
end of the multi-part
```

La valeur "B97C1230", que l'on retrouve en trois endroits, est une valeur à la fois aléatoire et unique. Il est possible de calculer une valeur unique en PHP de la façon suivante :

```
$val_unique = md5(uniqid(rand()));
```

Exercice 8
Affichez une liste de 20 valeurs uniques générées avec cette technique.
Exercice 9

Envoyez votre premier courriel en HTML :

```
<?php
$boundary = md5(uniqid(rand()));
$header = "";
$header .= "From: php <test@mondomaine.com>\n";
$header .= "Reply-To: reply@mondomaine.com\n";
$header .= "Reply-To: reply@mondomaine.com\n";
$header .= "MIME-Version: 1.0\n";
$header .= "Content-Type: multipart/alternative; boundary=$boundary\n";
$sujet = "test d'envoi HTML";
$html = "\n This is a multi-part message in MIME format.";
$html .= "\n--$boundary\nContent-Type: text/html; charset=\"iso-8859-1\"\n\n";
$html .= "<html><body>\n";
$html .= "<br><br><center><h2><font color='red'>premier courriel HTML</font></h2>\n";
$html .= "</body></html>\n";
$html .= "\n--$boundary--\n end of the multi-part";
mail("test@kernix.com",$sujet,$html,$header);
echo("courriel envoyé ...");
?>
```

Afficher les sources du courriel, pour voir comment la fonction a organisé les données.
Exercice 10
Reprenez votre exemple d'envoi de profil en ajoutant la dimension HTML :

```
<?php
$destinataire = "test@kernix.com";
$titre = "réponse au formulaire";
$boundary = md5(uniqid(rand()));
$header = "";
$header .= "From: script php <test@mondomaine.com>\n";
$header .= "Reply-To: reply@mondomaine.com \n";
$header .= "MIME-Version: 1.0\n";
$header .= "Content-Type: multipart/alternative; boundary=$boundary\n";
$message = "";
$message .= "\nThis is a multi-part message in MIME format.";
$message .= "\n--$boundary\nContent-Type: text/html; charset=\"iso-8859-1\"\n\n";
$message .= "<html><body>\n";
$message .= "<b>Titre</b> : <font color='red'>".$_REQUEST ['titre']."</font><br/>\n";
$message .= "<b>Année</b> : ".$_REQUEST['annee']."<br/>\n";
```

```
$str_genre = join(',',$_REQUEST['genre']);
$message .= "<b>Genre</b> : ".$str_genre."<br/>\n";
$message .= "<b>Description</b> : ".$_REQUEST ['description']."<br/>\n";
$message .= "<b>Couleur</b> : ".$_REQUEST['couleur']."<br/>\n";
$message .= "<b>Pays</b> : ".$_REQUEST['pays']."<br/>\n";
$str_soustitre = join(',',$_REQUEST['soustitre']);
$message .= "<b>Sous-titres</b> : <i>".$str_soustitre."</i><br/>\n";
$message .= "</body></html>\n";
$message .= "\n--$boundary--\n end of the multi-part";
if (mail($destinataire,$titre,$message,$header)==true) {
print("<hr/>Les informations ont bien été transmises.");
}
else {
die("<hr/>L'envoi du courriel a échoué.");
}
?>
```