Fiche PHP

Table des matières

I. Introduction

Une fois que PHP est installé, on peut ouvrir un serveur local PHP avec la commande :
php -S localhost:8000

Une fois que le serveur PHP est lancé, on peut se rendre à l'adresse suivante :
http://localhost:8000

```
<?php echo "Ceci est du texte !" ?>
<?php echo "Ceci est du <strong>texte</strong>"; ?>
<p>Aujourd'hui nous sommes le <?php echo date('d/m/Y h:i:s'); ?>.</p>
```

Pour commencer à écrire du code en PHP, on peut commencer par créer une page index.php dans laquelle on commencera par mettre le contenu d'une page HTML. Voici le type de code PHP que l'on peut y mettre :

Le code PHP s'écrit donc entre des balises <?php ?>. Il peut s'écrire dans la bage HTML mais aussi à l'intérieur d'une balise. Sur une seule ligne ou plusieurs.

Ici la fonction *echo* sert donc à afficher du texte sur la page HTML.

Pour ne pas se répéter on peut mettre des parties de pages amenées à être réutilisées comme les header et les footer dans d'autres fichiers. Ex : le header dans un fichier _header.php ; le footer dans un fichier _footer.php et on l'injecte dans le fichier index.php avec :

```
<?php include("_header.php"); ?>
<?php include("_footer.php"); ?>
```

Par défaut le PHP n'affiche pas ses erreurs (pour éviter de donner des informations aux utilisateurs) et affiche une page blanche en cas d'erreur. Sur un serveur personnel, il est conseiller de les afficher pour débugguer son code. Pour cela il faut :

- Insérer <?php phpinfo(); ?> rfDans la page HTML et chercher le fichier de configuration "Loaded configuration file"
- Ouvrir le fichier php.ini
- Mettre à la propriété **error_reporting** la valeur **E_ALL**
- Mettre à la propriété **display_errors** la valeur **On**

II. Les variables en PHP

Les variables vont permettre de stocker des informations dans différents types de données :

4 types scalaire :	4 types composés :	2 types spéciaux :
- Les chaînes de caractères : string - Les nombres entiers : int - Les nombres décimaux : float - Les booléens : bool	- Array - Object - Callable(fct de rappel) - Iterable (pseudo type)	- resource - NULL

La fonction *gettype($var)* permet de récupérer le type de variable, principalement à des fin de débogage.

Les fonctions *is_int($var), is_string($var) etc… de manière générale is_type($var) permettent de vérifier le type et retourne un booleen.*

https://www.php.net/manual/fr/types.comparisons.php

1. Affecter une valeur à une variable

```php
<?php
$userAge = 17;
$price = 57.3;
$isAuthor = true;
$isAdministrator = false;
$noValue = NULL;
?>
```

```php
<?php
$variable = "Mon \"nom\" est Florent";
$variable = 'Je m\'appelle Florent';
$variable = 'Mon "nom" est Florent';
$variable = "Je m'appelle Florent";
?>
```

Il faut faire attention avec les guillemets dans les chaînes de caractères.

Les fonctions *isset($var), is_null($var), empty($var)* permettent de vérifier l'état d'une variable :

```php
<?php
print isset($a); // $a is not set. Prints false. (Or more accurately prints ''.)
$b = 0; // isset($b) returns true (or more accurately '1')
$c = array(); // isset($c) returns true
$b = null; // Now isset($b) returns false;
unset($c); // Now isset($c) returns false;
empty($vessel); // returns true. Also $vessel is unset.
?>
```

La fonction *unset($var)*permet de désacfecter une varibale.

2. Convertir le type d'une vartiable :

Pour les booléens : "", "0", 0 sont considérés comme *false*. Tous les nombres et objets non vides sont considérés comme true.

```php
<?php
$foo = 10;   // $foo is an integer
$bar = (bool) $foo;   // $bar is a boolean
$foo = 10;            // $foo is an integer
$str = "$foo";        // $str is a string
$fst = (string) $foo; // $fst string (same)
var_dump((int)8.1); // 8 dans les deux cas
var_dump(intval(8.1)); // 8 dans les deux
cas
?>
```

```php
<?php
$foo = "5bar"; // chaîne
$bar = true;   // booléen

settype($foo, "integer"); // $foo vaut
maintenant 5   (integer)
settype($bar, "string");  // $bar vaut
maintenant "1" (string)
?>
```

3. Afficher le contenu d'une variable

```php
<?php
$fullname = 'Florent Vasseur';
echo $fullname;
?>
```

Pour les objetcts, tableaux etc… on utilise plutôt :

```php
var_dump($var);
print_r($array);
```

III. Les opérateurs

```php
<?php
$a = 1      // assignation / affectation   => $a vaut 1
$a == 1     // égalité (valeur uniquement)  => $a est-il égal à 1 ?
$a === 1    // égalité (valeur ET type)     => $a est-il égal à 1 et est-il un integer ?
$a + $b // addition
$a % $b // modulo (reste de la division)    => 2%1 = 0 car il reste 0. 3%2 reste 1.
$a++    // incrémentation                   => ici $a est incrémenté de 1 par rapport à sa
valeur précédente.
$a || $b    // ou
$a && $b    // et
$a != $b    // différent
$a . $b     // concaténation
!$a         // Not (inversion)
$a & $b     // et (comparaison de bits, à ne pas confondre avec &&)
?>
```

IV. Les chaînes de caractères

```php
<?php
$firstname = 'Indiana';
$lastname = 'Jones';
echo '$firstname'; // affiche : $firstname
echo "$firstname"; // affiche : Indiana
echo $firstname . $lastname;  // affiche : IndianaJones
$fullname = $firstname . ' ' . $lastname; // le résultat d'une concaténation peut aussi
être enregistré dans une variable.
echo $fullname; // affiche : Indiana Jones
$presentation = "$firstname $lastname est un célèbre archéologue"; // les variables sont
interprétées
echo $presentation; // affiche : Indiana Jones est un célèbre archéologue
$name = 'Indiana';
$name .= ' Jones'; // équivalent à $name = $name . ' Jones';
?>
```

L'opérateur de concaténation demande moins de ressources que l'interprétation avec les doubles guillemets.

Accéder à un caractère :

```php
<?php
$actor = 'Harrison Ford';
echo $actor[1] . PHP_EOL; // affiche 'a', le second caractère.
echo $actor[-2] . PHP_EOL; // affiche 'r'
echo $actor[25] . PHP_EOL; // affiche une erreur car ce caractère n'existe pas dans la chaîne !
?>
```

Méthode et fonction avec les chaînes de caractères (liste non exhaustive) :

- strlen($string) permet de connaître la longueur de la chaîne
- trim($string), ltrim($string), rtrim($string) permet de tronquer les caractères blancs au début, à la fin ou aux deux. Possibilité d'indiquer un caractère optionnel à trim.
- strtoupper($string), strlolower($string) renvoie respectivement une chaîne en majuscule et en minuscule.
- ucfirst($string), ucwords($string) convertie la 1ère lettre d'une chaîne / de chaque mot d'une phrase en majuscule.
- strreplace() permet de remplacer tout ou partie d'une chaîne par une autre chaîne
- $explode() et $implode() permettent de convertir une chaîne en tableau / un tableau en chaîne.

https://www.php.net/manual/fr/ref.strings.php
https://www.w3schools.com/php/php_ref_string.asp

V. Les tableaux

1. Tableaux indexés numériquement

```php
$weapons[0] = 'whip';
$weapons[1] = 'gun';
$weapons[2] = 'saber';
```

Ou encore :

```php
$weapons = ['whip', 'gun', 'saber'];
```

```php
$weapons[] = 'nuclear'; // pour ajouter un élement à la suite
```

Un tableau peut stocker différents éléments de différents types. Même d'autres tableaux.

On accède aux différentes valeurs du tableaux grâce aux index :

```php
  echo $weapons[0]; // affiche : whip
```

2. Tableaux associatifs

Tableaux sous forme key => value

```php
$weapons['weapon_one'] = 'whip';
$weapons['weapon_two'] = 'gun';
$weapons['weapon_three'] = 'saber';
```

```php
$weapons = [
    'weapon_one' => 'whip',
    'weapon_two'=>'gun',
    'weapon_three'=>'saber'
];
```

On accède aux éléments du tableaux à l'aide de la clé :

```php
echo $weapons['weapon_one']; // affiche : whip
```

3. Les tableaux multi-dimensionnels

Ce sont des tableaux qui vont contenir des tableaux en valeur. Eux même pouvant contenir des tableaux etc…

```php
$weapons = [
    'Indiana Jones' => ['whip', 'gun', 'saber'],
    'Marion Ravenwood' => ['knife', 'shield'],
    'Helen Seymour' => ['belt', 'dagger', 'gun', 'shield']
];
```

Pour accéder aux valeurs :

```php
print_r($weapons['Indiana Jones']);
// Array([0] => whip [1] => gun [2] => saber)
echo $weapons['Indiana Jones'][2] // saber
```

Dans cet exemple, les tableaux imbriqués sont indexés numériquement, mais cela pourrait très bien être des tableaux associatifs.

Il n'y a pas de limite dans l'imbrication des tableaux.

4. Méthodes et fonctions

Liste non exhaustive des fonctions utiles pour les tableaux :

- count() : permet de compter les éléments du tableaux
- sort() : trie un tableau par ordre croissant ; rsort() dans l'odre décroissant. A ne pas utiliser avec les tableaux associatifs car ça casse l'association clé => valeur
- Trie avec les tableaux associatifs (qui préserve l'association clé => valeurs
    - asort() trie croissant sur les valeurs
    - ksort() trie croissant sur les clés
    - arsrt() trie déccroissant sur les valeurs
    - krsort() trie décroissant sur les clés.

Les fonctions de trie modifient directement le tableau d'origine. Pas besoin de stocker dans une autre variable.

- in_array($valeur, $array, type(opt)) permet de vérifier la présence
    - true number 5 not the same as string "5"
    - false – default
- array_sum() calcule la somme des valeurs d'un tableau
- array_key_exists($key , $array) pour vérifier si une clé existe dans le tableau

```
$tab1 + $tab2     // union
```

https://www.php.net/manual/fr/ref.array.php
https://www.w3schools.com/php/php_ref_array.asp
https://www.progmatique.fr/article-33-Php-fonctions-manipuler-tableaux.html

Note :

Pour afficher des tableaux (débogage) on utilise les fonctions var_dump($array), print_r($array),

VI. Les boucles

1. Boucle For

```php
for ($i = 0; $i < 10; $i = $i+1) {
    // Do something
}
```

2. Boucle While

```php
$i = 0;
while ($i < 10) {
    // Do something
    $i++;
}
```

Pour itérer sur un tableau :

3. Boucle Foreach

Cette fonction permet de boucler sur les tableaux :

```php
$weapons = [
    'weapon_one' => 'whip',
    'weapon_two'=>'gun',
    'weapon_three'=>'saber'
];
```

Si on n'a pas besoin des clés :

```php
foreach ($weapons as $weapon) {
    echo $weapon . PHP_EOL;
}
```

Si on a besoin des clés :

```php
foreach($weapons as $key => $weapon) {
    echo $key . ' ' . $weapon . PHP_EOL;
}
```

4. Boucle sur les tableaux multidimensionnels

```php
<?php
$films = [
    'Indiana Jones et les aventuriers de l\'arche perdue' => ['Harrison Ford', 'Karen Allen', 'Paul Freeman', 'Ronald Lacey'],
    'Indiana Jones et le temple maudit' => ['Harrison Ford', 'Kate Capshaw', 'Jonathan Ke Quan', 'Amrish Puri'],
    'Indiana Jones et la dernière croisafe' => ['Harrison Ford', 'Sean Connery', 'Denholm Elliott', 'Alison Doody']
];
foreach ($films as $film => $actors) {
    echo 'Dans le film ' . $film . ', les principaux acteurs sont : '. '<br>';
    foreach ($actors as $actor) {
        echo '- ' . $actor . '<br>';
    }
};
```

```php
?>
```

Et dans une page HTML :

```php
<ul>
    <?php foreach ($films as $film => $actors) : ?>
        <li>
            <?php echo 'Dans le film ';?> <strong><?php echo $film; ?></strong><?php echo
', les principaux acteurs sont : '. '<br>'; ?>
            <ul>
                <?php foreach ($actors as $actor) : ?>
                    <li><?php echo '- ' . $actor . '<br>'; ?></li>
                <?php endforeach; ?>
            </ul>
        </li>
    <?php endforeach; ?>
</ul>
```

NB : voir mot clé *break* et *continue* dans les boucles for et while.

VII. Les conditions en PHP

1. Structure if

```php
<?php
$isAllowedToEnter = "Oui";
if ($isAllowedToEnter == "Oui") {
    // instructions
}
elseif ($isAllowedToEnter == "Non") {
    // instructions à exécuter quand on n'est pas autorisé à entrer
}
else {
    echo "Euh, je ne comprends pas ton choix, tu peux me le rappeler s'il te plaît ?";
}
?>
```

Avec des booléens :

```php
<?php
$isAllowedToEnter = true;

if ($isAllowedToEnter) {
    echo "Bienvenue petit nouveau. :o)";
}
else {
    echo "T'as pas le droit d'entrer !";
}
?>
```

On peut multiplier les conditions avec les opérateurs && (et) et || (ou).

2. Structure Switch

```php
switch (expression) {
    case 0:
        instructions
        break; // optionnel
    case 1:
        instructions
        break; // optionnel
    case x:
        instructions
        break; // optionnel
    default:
        instructions
        break; // optionnel
}
```

3. Structure match

```php
<?php
match(expression) {
    0 => expression,
    1 => expression,
    x => expression,
    default => expression,
};
?>
```

```php
$result = match ($x) {
    // This match arm:
    $a, $b, $c => 5,
    // Is equivalent to these three match arms:
    $a => 5,
    $b => 5,
    $c => 5,
};
```

L'avantage de la structure *match* est que, alors que *switch* compare les valeurs de façon "lâche" (==), *match* utilise un contrôle d'identité (===). L'expression de match peut également contenir plusieurs expressions séparées par des virgules, ce qui permet une syntaxe plus concise.

4. L'opérateur ternaire

Ces deux écritures sont équivalentes :

```php
<?php
$userAge = 24;

if ($userAge >= 18) {
    $isAdult = true;
}
else {
    $isAdult = false;
}
?>
```

```php
<?php
$userAge = 24;

$isAdult = ($userAge >= 18) ? true : false;

// Ou mieux, dans ce cas précis
$isAdult = ($userAge >= 18);
?>
```

5. Pour aller plus loin

```php
$year = {saisie utilisateur}

if (1981 == $year) {
    echo 'Oui';
} elseif (1984 == $year) {
    echo 'Oui';
} elseif (1989 == $year) {
    echo 'Oui';
} elseif (2008 == $year) {
    echo 'Oui';
} else {
    echo 'Non';
}
```

```php
$year = {saisie utilisateur}
$results = [1981, 1984, 1989, 2008];
$message = 'Non';

if (in_array($year, $results)) { // la fonction in_array vérifie l'existence d'une valeur dans un tableau.
    $message = 'Oui';
}

echo $message
```

A l'aide des objets, on diminue le nombre de conditions, et on simplifie donc la relecture. On permet aussi une évolution plus aisée du code.

VIII. Les PSR : PHP Standards Recommandations

```php
<?php
// Exemples de règles de PSR
// PSR OK
function hello(int $i)
{
    if ($i > 10) {
        echo $i . 'est supérieur à 10';
    }
}
// PSR KO
// L'accolade de la fonction n'est pas à la ligne : KO
// L'accolade du if est à la ligne : KO
// Pas d'espace entre le if et la parenthèse : KO
function hello() {
    if($i>10)
    {
        echo $i . 'est supérieur à 10';
    }
}
?>
```

IX. Les fonctions en PHP

1. Syntaxe de base

Définition :

```php
function myFunction()
{
  //Some code here...
}
```

Appel :

```php
myFunction();
```

2. Avec des paramètres

```php
function myFunction($param1, $param2)
{
  //Some code here using $param1 and
$param2
}
myFunction($varA, $varB);
```

```php
myFunction($param2, $param1);
```

Lors de cet appel *$param2* va jouer le rôle de *$param1* dans la fonction car c'est le 1er argument.
Lors de l'appel de la fonction, on peut nommer les arguments pour les appeler dans l'ordre de son choix :

```php
myFunction(param2: $varB, param1: $varA);
```

Fonction avec paramètre par défaut :

```php
function myFunction($param1, $param2 =
true)
{
  //Some code here using $param1 and
$param2
}
```

Les 3 appels donneront le même résultats :

```php
myFunction($var, true);
myFunction($var);
myFunction($var, null);
```

Si l'argument n'est pas fourni c'est celui par défaut qui sera utilisé.

Lors de la définition d'une fonction, on doit mettre les paramètres avec valeur par défaut après ceux qui n'en possèdent pas.

3. Les arguments nommés

A partir de php8, il est possible d'utiliser les arguments nommés qui permettent d'enlever la contraite de l'ordre des paramètres. Cela consiste à utiliser l'argument dans le signe $ suivi de deux points ":" puis de la variable passée en paramètre :

```php
function myFunction($param1, $param2)
{
  //Some code here using $param1 and
$param2
}
```

```php
myFunction($param2, $param1);
```

Ici *$param2* prendra le rôle de *$param1* dans la fonction car il est appelé en 1er.

```php
myFunction(param2: $varB, param1: $varA);
```

Ici *$varB* bien qu'appelé en 1er jouera le rôle de *$param2*.

4. Typage des paramètres et de la valeur de retour

Typage des paramètres :

```php
function myFunction(array $myList, string
$myText) : array

{
  //Some code here...
  return //some $var
}
```

Si les arguments passés à la fonction ne correspondent pas à ce qui est attendu, PHP renverra une erreur.

Typage des valeurs de retour :

```php
function myFunction( array $myList, string
$myText): array
{
  //Some code here...
  return //some $var
}
```

Il ne faut pas oublier ":" avant de préciser le type de retour. Erreur si la fonction renvoie autre chose qu'un tableau ici.

Une fonction ne doit que très rarement afficher directement un résultat. En revanche, elle devra presque toujours retourner une valeur. Il faut donc utiliser le mot clé *return*.

Quand n'y a pas de type de sortie on utilise *: void*.

On peut préciser deux types différents en séparant avec la barre verticale(pipe) int |float ; null | float ⇔ ?float

On ne peut retourner qu'une seule valeur à la fois.

Par défaut tous les arguments sont passés en valeur et non en référence. Pour passer une valeur en référence il faut rajouter l'esperluette devant : &.

5. Portée de variable : SCOPE :

```php
$trap = true;
function toggle ()
{
  $trap = false;
}
toggle();
echo $trap; // true
```

La fonction *toggle()* n'a pas accès à la variable *$trap.* Elle crée sa propre variable dans son scope.

6. Quelques fonctions natives en PHP

Quelques fonctions natives de PHP

- move_upload_file pour envoyer un fichier sur le serveur
- imagecreate : pour créer des images miniatures (thumbnails)
- mail pour envoyer un mail avec PHP
- crypt pour chiffrer des mots de passe
- isset() permet de vérifier si une variable est défnie
- date pour renvoyer l'heure, la date

```php
<?php
// Enregistrons les informations de date dans des variables
$day = date('d');
$month = date('m');
$year = date('Y');
$hour = date('H');
$minut = date('i');
// Maintenant on peut afficher ce qu'on a recueilli
echo 'Bonjour ! Nous sommes le ' . $day . '/' . $month . '/' . $year . 'et il est ' .
$hour. ' h ' . $minut;
?>
```

Fonction avec le texte :

- sprintf pour formater une chaîne de caractère

```php
$recipe = [
    'title' => 'Salade Romaine',
    'recipe' => 'Etape 1 : Lavez la salade
; Etape 2 : euh ...',
    'author' =>
'laurene.castor@exemple.com',
];
```

```php
echo sprintf(
    '%s par "%s" : %s',
    $recipe['title'],
    $recipe['author'],
    $recipe['recipe']
);
```

Donnera : Salade Romaine par "laurene.castor@exemple.com" : Etape 1 : Lavez la salade ; Etape 2 : euh ...

X. Intégrer HTML et PHP

C'est deux écritures ont le même efft :

```php
<?php
$chickenRecipesEnabled = true;

if ($chickenRecipesEnabled) {
    echo '<h1>Liste des recettes à base de poulet</h1>';
}
?>
```

```php
<?php $chickenRecipesEnabled = true; ?>
<?php if ($chickenRecipesEnabled): ?> <!-- Ne pas oublier le ":" -->
    <h1>Liste des recettes à base de poulet</h1>
<?php endif; ?><!-- Ni le ";" après le endif -->
```

Il n'y a pas d'accolade il ne faut pas oublier d'ajouter les ":" après la parenthèse fermante de l'instruction *if*. Il faut rajouter l'instruction *endif* à la fin.

La même chose avec une boucle *foreach* :

```php
<h1>Affichage des recettes</h1>
<!-- Boucle sur les recettes -->
<?php foreach($recipes as $recipe) : ?>
    <!-- si la clé existe et a pour valeur "vrai", on affiche -->
    <?php if (array_key_exists('is_enabled', $recipe) && $recipe['is_enabled'] == true): ?>
        <article>
            <h3><?php echo $recipe['title']; ?></h3>
            <div><?php echo $recipe['recipe']; ?></div>
            <i><?php echo $recipe['author']; ?></i>
        </article>
    <?php endif; ?>
<?php endforeach ?>
```

Pour ne pas se répéter, il peut être intéressant de mettre le header et le footer dans un fichier particulier et de les inclure dans les fichiers de chaque page avec *include :*

```php
<?php include('header.php'); ?>
```

On peut aussi utilise *require* mais dans ce cas, si le fichier est manque le reste de la page ne se charge pas.

XI. Les erreurs en PHP

1. Parse error

Si on formule mal une instruction comme

- oublier un point virgule à la fin d'une instruction
- oublier de ferme des guillemet
- se tromper dans la concaténation
- mal ferme une accolade

En fonction de l'erreur la ligne indiqué lors de l'erreur n'est pas forcément la ligne où se situe le problème,

2. Undefined function

Si on utilise une fonction non reconnue :

- soit la fonction n'existe vraiment pas
- soit elle existe mais PHP ne la reconnaît car elle se trouve dans une extension que l'on n'a pas activé.

3. Wrong parameter count

Si on entre un nombre incorrect de paramètre dans une fonction.

Il y a une liste de 3946 erreurs en PHP,,,

- Headers already sent by : si on écrit du code au mauvais endroit,
- Maximum execution time exceeded : si on a fait une boucle infinie

PHP limite le temps d'exécution à 30secondes par défaut.

XII. Transmettre les données de page en page

1. Ecouter la requête des utilisateurs

On va rendre le formulaire de contact dynamique en retrouvant une page de prise en compte de la demande à chaque personne qui soumettra le formulaire.

Lorsque l'on fait une recherche sur Google, la barre d'adresse contient une URL un peu longue qui ressemble à ceci :

http://www.google.fr/search?rlz=1C1GFR343&q=motCherche

Les informations après le "?" sont en réalité des données que l'on fait transiter d'une page à l'autre.

Envoyer des paramètres dans l'URL :

Si sur notre site il y a un formulaire de contact, lorsque l'on validera le formulaire les informations vont figurer à la fin de l'url :

http://www.monsite.com/contact.php?nom=Dupont&prenom=Jean

Le "?" sépare me nom de la page des paramètres qui se trouvent sous la forme *nom=valeur* et sont séparés par "&".

La seule limite est la longueur de l'URL qui ne doit pas dépasser 256 caractères.

Pour envoyer des paramètres dans l'url on peut donc faire de deux méthodes :

Via un lien

```
<a
href="bonjour.php?nom=Dupont&amp;prenom=Jean">Dis-
moi bonjour !</a>
```

Via un formulaire avec la méthode GET

```
<form action="contact.php"
method="GET">
    <!-- données à faire passer à
l'aide d'inputs -->
    <input name="nom">
    <input name="prenom">
</form>
```

Récupérer les paramètres en PHP

Supposons que l'on a un formulaire sur une page contact.php et que l'on va soumettre sur une autre page submit_contact.php, un message qui accusera bonne réception. Formulaire simplifié :

```
<form action="submit_contact.php" method="GET">
    <div>
        <label for="email">Email</label>
        <input type="email" name="email">
    </div>
    <div>
        <label for="message">Votre message</label>
        <textarea placeholder="Exprimez vous" name="message"></textarea>
    </div>
    <button type="submit">Envoyer</button>
</form>
```

Si l'utilisateur complète le formulaire avec utilisateur@example.com et comme message "Bonjour", le formulaire va alors être converti en lien vers :

```
submit_contact.php?email=utilisateur%40exemple.com&message=Bonjour
```

On va récupérer les informations par PHP dans le fichier *sbmit_contact.php*.

Lors de la soumission, une variable *superglobale* appelée $_GET va contenir les données envoyées. Il s'agit d'un tableau associatif dont les clés correspondent aux noms des paramètres envoyés dans l'URL :

```
$_GET['email'] = utilisateur@exemple.com
$_GET['message'] = Bonjour
```

On peut donc récupérer les informations, les traiter, les afficher :

```
<h1>Message bien reçu !</h1>
<div class="card">
    <div class="card-body">
        <h5 class="card-title">Rappel de vos informations</h5>
        <p class="card-text"><b>Email</b> : <?php echo $_GET['email']; ?></p>
        <p class="card-text"><b>Message</b> : <?php echo $_GET['textarea']; ?></p>
    </div>
</div>
```

Ne jamais faire confiance aux données reçues

- Contrôler la présence d'un paramètre

Comme les données sont transmises dans l'URL, tous les utilisateurs peuvent modifier les URL et mettre ce qu'ils veulent, dont du code malicieux.

L'utilisateur peut aussi essayer d'accéder à la page *submit_contact.php* en supprimant les paramètres de l'URL. Dans ce cas cela affichera :

```
Notice: Undefined index: email in submit_contact.php on line 15
OU
Warning: Undefined array key "email" in submit_contact.php on line 15
```

En effet comme on les a supprimées de l'URL, elles n'ont as été créés. Pour résoudre ce problème on peut faire appel à une fonction PHP : *isset()* :

```
<?php
if (!isset($_GET['email']) || !isset($_GET['message']))
{
    echo('Il faut un email et un message pour soumettre le formulaire.');
    // Arrête l'exécution de PHP
    return;
}
?>
```

Avec ce code, s'il nous manque des variables on affiche un message d'erreur et on arrête l'exécution de la page. Il est nécessaire de faire cela car on ne peut pas faire confiance à l'utilisateur !

- Contrôler la valeur d'un paramètre

Une fois que les valeurs sont soumises et correctement récupérées, il faut qu'elles correspondent à ce qui est attendu. Il va donc falloir contrôler :

  - Si l'email passé est bien valide avec la fonction *filter_var()*
  - Si le message n'est pas vide à l'aide de la fonction *empty()*

Voici donc le code final sécurisé :

```php
<?php
if (
    (!isset($_GET['email']) || !filter_var($_GET['email'], FILTER_VALIDATE_EMAIL))
    || (!isset($_GET['message']) || empty($_GET['message']))
    )
{
    echo('Il faut un email et un message valides pour soumettre le formulaire.');
    return;
}
```

Cela fait beaucoup de conditions mais c'est nécessaire. On doit toujours faire attention et prévoir tous les cas les plus tordus.

Il peut être aussi fortement conseillé d'utiliser la fonction *trim()* qui enlève les espaces de part et d'autre des données saisies par l'utilisateur.

Pour ne pas avoir les informations qui transitent dans la barre d'addresse, on peut utiliser la méthode POST et on remplace les GET par les POST. Mais on peut toujours récupérer les informations autrement.

2. Administrer les formulaires de façon sécurisée

Il y a deux attributs très importants à connaitre pour la balise <form> :

- La méthode : method
- La cible : action

Il y a deux méthodes pour envoyer les données des formulaires :

- *get* : les données vont transiter par l'URL. On pourra les récupérer grâce au tableau (array) : $_GET. Cette méthode est peu utilisée car on ne peut pas envoyer beaucoup d'informations dans l'URL (256 caractères maximum).
- *post* : les données ne transiteront pas par l'URL, l'utilisateur ne les verra donc pas dans la barre d'adresse. Cette méthode permet d'envoyer autant de données que l'on veut, ce qui fait qu'on la privilégie le plus souvent. Mais les données ne sont pas plus sécurisées qu'avec la méthode GET et il faudra toujours vérifier si tous les paramètres sont bien présents et valides comme vu précédemment. **On ne doit pas plus faire confiance aux formulaires qu'aux URL.**

La bonne pratique est donc de privilégier la méthode *post* pour les formulaires.

L'attribut *action* sert à définir la page appelée par le formulaire. C'est cette page qui recevra les données du formulaire et qui sera chargée de les traiter. Selon la méthode utilisée, ce ne sera pas la même variable spéciale qui aura accès aux données :

- Pour la méthode GET c'est la supervariable $_GET
- Pour la méthode POST c'est la supervariable $_POST

On travaille donc bien sur deux pages différentes :

- La page qui contient le formulaire : form.php par exemple
- Celle qui reçoit les données du formulaire pour les traiter submit_form.php

Les clés des tableaux des valeurs $_GET et $_POST sont les attribut *"name"* des champs *input* du formulaire. Il y a donc autant de clé dans le tableau $_GET que champ avec un attribut *name* dans le formulaire.

Les champs cachés :

C'est un code dans le formulaire qui n'apparaîtra pas aux yeux du visiteurs mais qui va quand même créer une variable avec une valeur. On peut s'en servir pour transmettre des informations fixes.

Imaginions que l'on veuille "retenir" le pseudo du visiteur on peut alors rajouter le code suivant dans la page :

```
<input type="hidden" name="pseudo" value="Mateo21" />
```

Sur la page web, on ne verra rien mais une variable $_POST['pseudo'] sera créée et elle aura la valeur "Mateo21". C'est apparemment inutile mais on en aura parfois besoin.

Ces champs ne sont pas réellement cachés. N'importe quel visiteur peut afficher le code source de la page pour voir et modifier les champs cachés en lisant le code HTML.

La faille XSS :

Tout comme les paramètres qui transitent par l'URL, il ne faut jamais faire confiance aux données reçues. Il faut faire attention au code HTML que l'on reçoit.

La faille XSS (pour cross-site scripting) est vieille comme le web ! Et on la trouve encore sur de nombreux site web, même professionnels. C'est une technique qui consiste à injecter du code HTML contenant du JavaScript dans vos pages pour le faire exécuter à vos visiteurs.

On reprend le formulaire précédent qui envoie un message et un email :

```
<h5>Rappel de vos informations</h5>
<p><b>Email</b> : <?php echo $_POST['email']; ?></p>
<p><b>Message</b> : <?php echo $_POST['message']; ?></p>
```

Si le visiteur décide d'écrire du code HTML dans la zone de message cela fonctionnera très bien. Imaginons qu'il écrive cela dans le corps de son message : Badaboom, le code source HTML qui sera généré par PHP sera :

```
<p><b>Message</b> : <strong>Badaboum</strong></p>
```

Outre le fait que l'utilisateur peut insérer n'importe quel code HTML et rendre la page invalide (ce qui n'est pas le plus grave), il peut aussi ouvrir des balises de type <script> pour faire exécuter du code JavaScript au visiteur qui visualisera la page :

```
<p><b>Message</b> : <script>alert('Badaboum')</script></p>
```

Tous les visiteurs qui arriveront sur cette page verront une boîte de dialogue JavaScript s'afficher. Ce qui est plutôt gênant.

Il faut donc sécuriser notre code en bloquant l'exécution de code JavaScript.

Pour ignorer le code HTML, il suffit d'utiliser la fonction *htmlspecialchars*. On dira dans ce cas qu'on "échappe" le code HTML, c'est-à-dire que la fonction JavaScript *alert()* n'en tiendra pas compte. Elle va transformer les chevrons des balises HTML < et > en $lt; et $gt; respectivement. Cela provoquera l'affichage de la balise plutôt que son exécution.

```
<p><b>Message</b> : <?php echo htmlspecialchars($_POST['message']); ?></p>
```

Le code HTML qui en résultera sera propre et protégé car les balises HTML insérées par le visiteur auront été échappées.

```
<p><b>Message</b> : &lt;strong&gt;Badaboum&lt;/strong&gt; ?></p>
```

Tout ce qui est affiché et qui vient, à la base, d'un visiteur, doit être protégé avec *htmlspecialchars*.

Si on préfère retirer les balises HTML que le visiteur a tenté d'envoyer plutôt que les afficher, on peut utiliser la fonction *strip_tags.*

Ici donc une nouvelle version du code de réception du formulaire (avec post) sécurisée, qui prévoit tous les cas pour éviter d'être pris au dépourvu par un utilisateur mal intentionné :

```php
<?php

// submit_contact.php
if (!isset($_POST['email']) || !isset($_POST['message']))
{
    echo('Il faut un email et un message pour soumettre le formulaire.');
    return;
}

$email = $_POST['email'];
$message = $_POST['message'];

?>

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Site de Recettes - Demande de contact reçue</title>
        <link
            href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
            rel="stylesheet"
        >
    </head>
    <body>
        <div class="container">

        <?php include_once('header.php'); ?>
            <h1>Message bien reçu !</h1>

            <div class="card">

                <div class="card-body">
                    <h5 class="card-title">Rappel de vos informations</h5>
                    <p class="card-text"><b>Email</b> : <?php echo($email); ?></p>
                    <p class="card-text"><b>Message</b> : <?php echo strip_tags($message);
?></p>
                </div>
            </div>
        </div>
    </body>
</html>
```

XIII. Le partage de fichier

1. Paramétrer le formulaire d'envoie de fichier

On commence par créer un formulaire. L'utilisateur le remplit en incluant le fichier à envoyer. Puis PHP réceptionne les données du formulaire et, s'il y ades fichiers dedans, il les "enregistre" dans un des dossiers du serveur.

Attention : l'envoie du fichier peut être un peu long s'il est gros. Il faudra prévenir le visiteur de ne pas s'impatienter pendant l'envoie.

Dès l'instant où le formulaire propose aux visiteurs d'envoyer un fichier, il faut ajouter l'attribut *enctype="multipart/form-data"* à la valise <form>.

```
<form action="submit_contact.php" method="POST" enctype="multipart/form-data">
    <!-- champs de formulaire -->
</form>
```

Grâce à cet attribut, le navigateur sait qu'il s'apprête à envoyer des fichiers. Pour envoyer des fichiers, il faut ajouter *<input type="file" />* et donner un nom grâce à l'attribut *name*.

```
<form action="submit_contact.php" method="POST" enctype="multipart/form-data">
    <!-- Ajout des champs email et message -->
    [...]
    <!-- Ajout champ d'upload ! -->
    <div class="mb-3">
        <label for="screenshot" class="form-label">Votre capture d'écran</label>
        <input type="file" class="form-control" id="screenshot" name="screenshot" />
    </div>
    <!-- Fin ajout du champ -->
    <button type="submit" class="btn btn-primary">Envoyer</button>
</form>
```

2. Traitez l'envoi en PHP

Il faut donc maintenant rajouter du code dans la page *submit_contact.php* pour traiter l'envoie du fichier.

Au moment où la page PHP s'exécute, le fichier est envoyé sur le serveur mais il est stocké dans un dossier temporaire. Il faut décider si on accepte le fichier et où on va le mettre.

Il faudra vérifier si le fichier a la bonne extension. Si le fichier est bon, on va l'accepter grâce à la fonction *move_upload_file()*, de manière définitive.

Pour traiter le fichier, on va utiliser la supervariable $_Files. Pour chaque fichier envoyé, une variable $_FILES['nom_du_champ']. Ici cela sera $_FILES['screenshot']. Cette variable est un tableau qui contient plusieurs informations sur le fichier.

Variable	Signification
$_FILES['screenshot']['name']	Contient le nom du fichier envoyé par le visiteur
$_FILES['screenshot']['type']	Indique le type du fichier envoyé. Si c'est une image gif par exemple, le type sera *image/gif*
$_FILES['screenshot']['size']	Indique la taille du fichier envoyé. La taille est en octet. Limitée par PHP, par défaut, à 8Mo.
$_FILES['screenshot']['tmp_name']	Contient l'emplacement temporaire du fichier.
$_FILES['screenshot']['error']	Contient un code d'erreur permettant de savoir si l'envoie s'est bien effectué ou s'il y a eu un problème et si oui lequel. La variable vaut 0 s'il n'y a pas eu d'erreur.

S'il y a un second champ d'envoi de fichier dans le formulaire, il y aura une seconde variable $_FILES['nom_autre_champ'].

Pour décider si on accepte ou non un fichier, il faut faire les vérifications suivantes :

Tester si le fichier a bien été envoyé

On teste la variable $_FILES['screenshot'] avec *isset()* et s'il n'y a pas d'erreur avec $_FILES['screenshot']['error'].

```php
<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['screenshot']) && $_FILES['screenshot']['error'] == 0)
{

}
?>
```

Vérifier la taille du fichier

On vérifie que la taille du fichier ne dépasse pas 1M par exemple avec $_FILES['screenshot']['size'].

```php
<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['screenshot']) && $_FILES['screenshot']['error'] == 0)
{
        // Testons si le fichier n'est pas trop gros
        if ($_FILES['screenshot']['size'] <= 1000000)
        {

        }
}
?>
```

Vérifier l'extension du fichier

Il faut obligatoirement interdire les ficher PHP sinon cela pourrait exécuter du code sur le serveur. On analyse donc la variable $_FILES['screenshot']['type'].

```php
<?php
$fileInfo = pathinfo($_FILES['screenshot']['name']);
$extension = $fileInfo['extension'];
?>
```

La fonction *pathinfo* renvoie un tableau (array) contenant entre autres l'extension du fichier dans $fileInfo['extension']. On stocke ça dans une variable *$extension*.

Une fois l'extension récupérée, on peut la comparer à un tableau d'extension autorisées, et vérifier si l'extension récupérée fait bien partie des extensions autorisées à l'aide de la fonction *in_array()*.

On obtient alors le code suivant :

```php
<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['screenshot']) AND $_FILES['screenshot']['error'] == 0)
{
        // Testons si le fichier n'est pas trop gros
        if ($_FILES['screenshot']['size'] <= 1000000)
        {
                // Testons si l'extension est autorisée
                $fileInfo = pathinfo($_FILES['screenshot']['name']);
                $extension = $fileInfo['extension'];
                $allowedExtensions = ['jpg', 'jpeg', 'gif', 'png'];
                if (in_array($extension, $allowedExtensions))
                {

                }
        }
}
?>
```

Valider l'upload du fichier

Si tout est bon, on accepte le fichier en appelant *move_upload_file()*. Cette fonction prend 2 paramètres :

- Le nom temporaire du fichier $_FILES['screenshot']['tmp_name']
- Le chemin qui est le nom sous lequel sera stocké le fichier de façon définitive. On peut utiliser le nom d'origine du fichier $_FILES['screenshot']['name'] ou générer un nom au hasard.

Supposons que l'on veuille stocker le fichier dans un sous-dossier "Uploads" et que l'on garde le même nom de fichier que celui d'origine.

Comme $_FILES['screenshot']['name'] contient le chemin entier vers le fichier d'origine (C:\dossier\fichier.png par exemple), il nous faudra extraire le nom du fichier. Pour cela on va utiliser la fonction *basename* qui renverra juste *fichier.png*.

```php
<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['screenshot']) && $_FILES['screenshot']['error'] == 0)
{
        // Testons si le fichier n'est pas trop gros
        if ($_FILES['screenshot']['size'] <= 1000000)
        {
                // Testons si l'extension est autorisée
                $fileInfo = pathinfo($_FILES['screenshot']['name']);
                $extension = $fileInfo['extension'];
                $allowedExtensions = ['jpg', 'jpeg', 'gif', 'png'];
                if (in_array($extension, $allowedExtensions))
                {
                        // On peut valider le fichier et le stocker définitivement
                        move_uploaded_file($_FILES['screenshot']['tmp_name'], 'uploads/' .
basename($_FILES['screenshot']['name']));
                        echo "L'envoi a bien été effectué !";
                }
        }
}
?>
```

Il faudra vérifier que le dossier "Uploads" existe sur le serveur et qu'il a les droits d'écriture. Pour ce faire, sous FileZilla par exemple, faire clic droit sur le dossier et choississez "Attributs du fichier". Cela permettra d'éditer les droits du dossier (on parle de CHMOD). Il faut mettre les droits à 733, ainsi PHP pourra placer les fichiers téléversés dans ce dossier.

Ce script est un début et il faudra encore l'améliorer.

Par exemple, si le fichier contient des espaces ou des accents, cela posera un problème une fois envoyé sur le web. D'autre part, si quelqu'un envoie un fichier qui a le même nom que celui d'une autre personne, l'ancien sera écrasé.

La solution consiste en général à choisir nous-mêmes le nom du fichier stocker sur le serveur plutôt que de nous servir du nom d'origine. On peut faire par exemple un compteur qui s'incrémente : 1.png, 2.png, 3.png etc…

Rappel : on doit éviter que quelqu'un puisse envoyer un fichier PHP sur le serveur.