CC BY NC SA

SUPPORT DE COURS

PHP

date	révision
Janvier 2019	Amélioration (v.83)
Juin 2019	Ajout instructions de débogages
Décembre 2019	Mise à jour serveur PHP et comparaison des différents langages (initiation rapide via autre lang.)
Février 2020	Correction quote/guillemet + $num manquant
Mars 2020	Corrections copier/coller p29 et guillemet p27 + déplacement chapitre POO à la fin
Février 2021	Corrections boucles et mise à la norme du nouveau référentiel + ajout Laragon
Février 2022	Corrections fonctions deprecated, simplifications diverses, ajout declare(strict_types=1);
Janvier 2023	Corrections affichage dans chaque langage (print, WriteLine, console.log()...)
Février 2024	Gestion des erreurs dans un formulaire

TABLE DES MATIÈRES

1 INTRODUCTION

PHP est un langage ouvert, créé par Rasmus LERDORF en 1994, avec pour objectif de garder une trace des visiteurs qui consultaient son CV.

PHP Signifie **Personal Home Page** mais certains insinuent que c'est l'acronyme de People Hate Perl car Perl était un autre langage à la mode. Finalement, c'est **PHP Hypertext Preprocessor** qui reste la définition actuelle.

PHP est initialement un langage de script, et non un langage de programmation : il est censé réagir aux événements lorsque l'utilisateur soumet une requête.

Il y a eu de nombreuses versions de PHP mais actuellement, deux versions cohabitent :

- La version 7 qui est la plus moderne et la plus performante, mais aussi la plus incompatible avec les anciens scripts (sortie en décembre 2015).
- La version 5.6 qui devient obsolète mais dont le support existe pour faciliter la maintenance des anciens scripts.

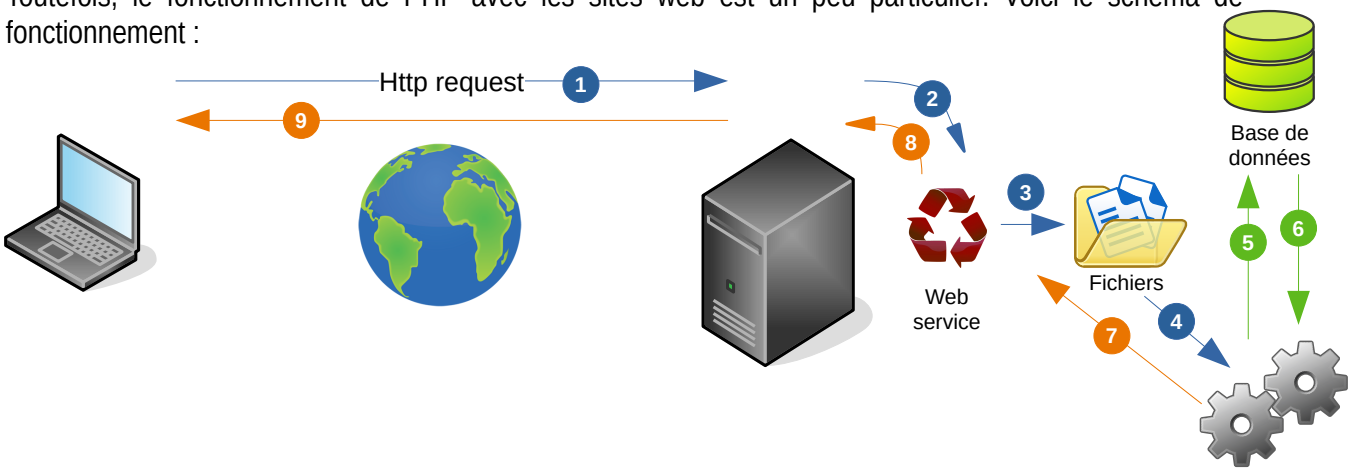
Ce cours abordera PHP dans sa généralité, la notion de version ne devrait pas avoir d'impact (sinon, ce sera précisé). La première partie sera générale tandis que les parties suivantes donneront des exemples pratiques.

2 FONCTIONNEMENT

2.1 TRANSACTIONS HTTP

PHP peut fonctionner comme langage de script (au même titre que PowerShell, Ruby, Perl).

Toutefois, le fonctionnement de PHP avec les sites web est un peu particulier. Voici le schéma de fonctionnement :

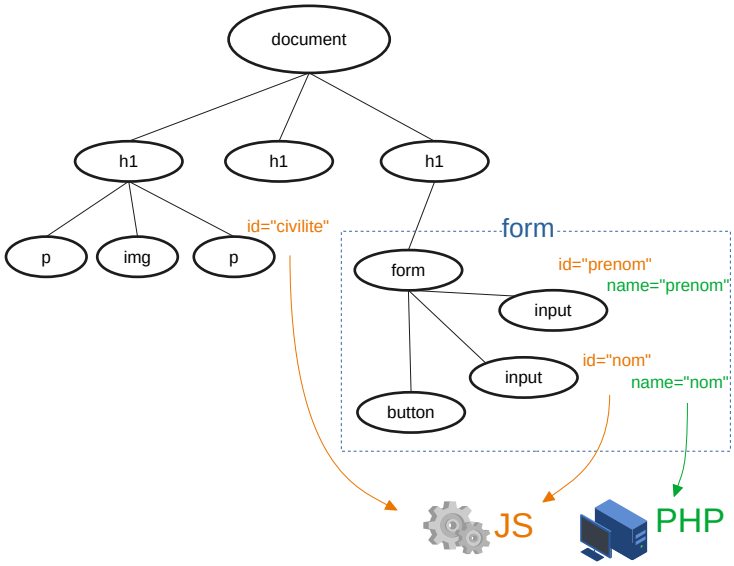


- ❶ **Le client ouvre une connexion** sur le serveur web, sur le port 80 par défaut. Le service web ❷ répond (sur le port TCP 80 ou 43). Il vérifie que le fichier demandé dans le lien existe ❸. Si c'est un fichier PHP, il l'envoie au moteur PHP ❹.
- Le moteur PHP traite éventuellement des données dans la base (❺ **et** ❻).Il retourne le résultat au service web ❼ **qui renvoie une page web HTML au client (❽ et ❾).**

2.2 RÉCUPÉRATION DE FORMULAIRES

Pour envoyer des données provenant d'une page HTML à PHP, il est nécessaire d'utiliser les formulaires et une méthode (GET ou POST, sachant que les valeurs transmises avec GET sont visibles dans l'URL).

Pour que PHP puisse reconnaître les valeurs, on doit leur donner un nom, avec la propriété `name` (qui agit différemment de la propriété `id`).



2.3 INTÉRÊT DE CETTE ARCHITECTURE

Le traitement est effectué du côté serveur (qui est censé être sécurisé).

En effet, JavaScript (côté client) peut vérifier des formulaires, modifier des données… mais il est toujours possible de changer le résultat en modifiant la page localement avec le débogueur intégré aux navigateurs.

En PHP, il faut pouvoir accéder aux fichiers sur le serveur, ce qui nécessite de compromettre le serveur : c'est beaucoup plus difficile.

2.4 INCONVÉNIENT DE CETTE ARCHITECTURE

Cet aspect implique que pour tester du code PHP, il faut **impérativement** avoir un serveur web actif : cela peut être Microsoft IIS, Apache de l'Apache Software Foundation, NGINX d'Igor Sysoev ou d'autres serveurs.

Microsoft IIS	APACHE HTTP SERVER	NGINX	

Tableau 2.1 : les serveurs web les plus connus

Pour faciliter l'installation d'un ensemble de fonctions et de services, il existe plusieurs solutions gratuites

EasyPHP	**EasyPHP** intègre les services Apache, PHP et MySQL. Il propose deux versions : Devserver et Webserver (censé permettre la mise en production de site). Cette version est complètement configurable, et permet d'arrêter ou démarrer facilement les services (web ou base de données) et est portable. 61 Mo
WAMP	**WAMP** est similaire, avec les services Apache, PHP et MySQL. 431 Mo
UwAmp	**UwAmp** intègre lui aussi Apache, PHP et MySQL. Il propose des raccourcis graphiques (pratique pour les débutants) et est portable (peut fonctionner depuis une clé USB, sans installation préalable) 36 Mo (sans Visual C++ redistribuable)
XAMPP	**XAMPP** se veut totalement libre, et ne propose que des logiciels libres et open-source. Apache, MariaDB, PHP et Perl. 123 Mo
Winginx	**Winginx** utilise NGINX comme moteur web. Pour le reste, on retrouve PHP et MySQL (mais aussi NodeJS, MongoDB, Redis). 45 Mo
Laragon	**Laragon** est l'un des derniers nés et est portable. Il ne fonctionne que sous Windows mais propose NodeJS, Redis et quelques autres outils de développements. 135 Mo

Tableau 2.2 : choisissez votre solution (cliquez sur le logo)…

PHP et MySQL ne sont pas les seuls langages pour créer et animer des sites web : **ASP.net** de Microsoft, **Ruby on Rails**, **Django** (Python), **Java EE** (JSP) permettent de rendre un site web dynamique. De même **SQL server** de Microsoft, **MariaDB** (clone de **MySQL**), **Oracle** et **PostGreSQL** sont d'autres bases de données reconnues.

Pour Apache, il faut configurer "`AddHandler php5-script .php .html`" si vous voulez que PHP puisse lire les scripts PHP à l'intérieur des pages HTML.

3 La syntaxe PHP

PHP étant devenu structuré et orienté objet, de nombreux concepts sont déjà connus.

3.1 Syntaxe

Comme la plupart des langages, PHP utilise une syntaxe souvent similaire mais a ses propres conventions et particularités :

Commentaires	Bloc	Fin de ligne	Variables
// comment. /* comment. */	{ }	;	$xxx

D'autre part, une page PHP est un code web (HTML) qui sera généré et renvoyé au navigateur par le serveur. Le serveur analyse la page, et s'il rencontre les balises PHP, exécute le script entre celles-ci.

Balise ouvrante	Balise fermante
<?php	?>

Voici un premier exemple de script PHP. Ce script affiche automatique une liste à puce contenant 5 lignes :

index.php

Code HTML

Variable $i

Ouverture bloc

```
<html>
  <head></head>
  <body>
    <ul>
      <?php for($i=1; $i<=5; $i++){ ?>
        <li>Liste à puce N° <?php echo $i; ?></li>
      <?php } ?>
    </ul>
  </body>
</html>
```

Résultat dans le navigateur :

- Liste à puce N° 1
- Liste à puce N° 2
- Liste à puce N° 3
- Liste à puce N° 4
- Liste à puce N° 5

Attention : les pages PHP doivent impérativement se trouver sur le serveur Apache pour que le script PHP s'exécute. Si vous double-cliquez directement sur une page PHP, elle s'ouvrira dans le navigateur mais n'affichera que les parties HTML.

PHP conserve certains défauts d'origine pour des raisons de compatibilité. Notamment, comme en JavaScript, il n'est pas nécessaire de typer les variables. Toutefois, il est possible d'utiliser une instruction dédiée pour respecter un typage fort, comme dans les langages typés comme C#, C++ ou Java...

```
declare(strict_types=1);
```

Dans ce cas, il est possible de décrire les types attendus dans les fonctions.

Code sans erreur	Code indiquant une erreur

`function outputAnInteger(int $num)`
`{`
`    echo $num;`
`}`
`outputAnInteger("3");`
	`declare(strict_types=1);`
	
	`function outputAnInteger(int $num)`
	`{`
	`    echo $num;`
	`}`
	`outputAnInteger("3");`

3.2 POINTS COMMUNS (C#, JAVA, JAVASCRIPT)

3.2.1 Itérations et conditions

La mise en place d'une condition ou d'une boucle est identique :

```
<?php
      $num = 4;
      if ($num == 0) {
            echo "La variable est nulle";
      } else {
            for ($t=0 ; $t<$num; $t++) {
                  echo "compte ".$t."/".$num." fois";      // la variable est
affichée
            }
      }
      // echo ecrire($nom);
?>
```

condition

boucle

Vous noterez que la **concaténation** de chaîne ne se fait plus avec le symbole '**+**' mais avec le point '**.**' car ne pas avoir de type sur les variables implique que l'addition soit uniquement numérique.

3.2.2 Gestion des chaînes et variables

Notez, que PHP est capable de lire une variable à l'intérieur d'une chaîne avec des guillemets. En revanche, il n'interprète pas les chaînes qui sont encadrées par de simples quotes.

Guillemet "	Simple quote '
`$nom = "NORRIS";` `echo "votre nom est $nom.";`	`$nom = "NORRIS";` `echo 'votre nom est $nom.';`
Votre nom est NORRIS.	Votre nom est $nom.

PHP	`echo "votre nom est $nom.";` `print "votre nom est $nom.";`
C#	`Console.WriteLine("Votre nom est {0}", nom);`
Java	`System.out.println("Votre nom est %s", nom);`
JavaScript	`console.log(`Votre nom est ${nom}`)`

3.3 Différences (C#, Java, JavaScript)

Si vous avez déjà appris C#, les différences sont faibles sur l'aspect procédural. La déclaration d'une classe et l'usage de méthodes est similaire dans l'écriture mais la syntaxe est différente.

3.3.1 Synchronisme

PHP est un langage synchrone, dans lequel les actions sont faites les unes après les autres. C'est également le cas de C# et Java, mais pas celui de JavaScript qui se veut asynchrone.

3.3.2 Boucle foreach

Les boucles for, while et do/while sont similaires dans les 4 langages. La boucle foreach reste propre à chaque langage.

Le tableau ci-après résume les écarts entre les différents langages :

PHP	`foreach ($unTableau as $laValeur) {`
	`  echo $laValeur;`
	`}`

C#	`foreach (Type laValeur in unTableau) {`
	`  System.Console.Writeln(laValeur);`
	`}`

Java	`foreach (Type laValeur : unTableau) {`
	`  System.out.println(laValeur);`
	`}`

JavaScript	`UnTableau.forEach(maFonction);`
	`function maFonction(laValeur, index) {`
	`  console.log(laValeur)`
	`}`

3.3.3 Syntaxe particulière des classes

Les classes étant initialement réservées aux langages Java et C#, PHP a dû s'adapter. En PHP, le caractère '.' étant réservé à la concaténation, on utilise le symbole '->' (tiret et plus grand) pour l'affectation.

```
$obj->methode() ;
```

Lorsqu'une classe hérite d'une autre classe, en C# le symbole est "classe:classeMère". En PHP, on utilise le mot "extends" (comme en Java) ou bien " :: ".

```
class Saisie extends Control {
   ...
}
```

Enfin, la création d'un constructeur ne porte pas le nom de la classe mais "__construct" :

```
public function __construct($util, $get=null) {
```

Remarquez que ce constructeur peut ne recevoir qu'un paramètre, l'autre sera alors mis à zéro. En revanche, si le constructeur reçoit deux paramètres, $get prendra la valeur du deuxième.

Le tableau ci-après résume les écarts entre les différents langages :

PHP	C#	Java	Javascript
`$obj->methode();`	`Obj.methode();`	`Obj.methode();`	`Obj.methode()`
`Class AI extends B {}`	`Class AI : B {}`	`Class AI extends B {}`	`Class AI extends B {}`
`__construct() {}`	`Public AI() {}`	`Public AI() {}`	`Constructor() {}`
`$D = new AI()`	`AI D = new AI()`	`AI D = new AI()`	`Let D = new AI()`

3.3.4 Supervariables

PHP utilise quelques variables très particulières, que l'on ne retrouve pas toujours dans d'autres langages. Ce sont des variables globales, dont la **portée** est accessible de partout (et par tout le monde) dans le programme :

- **$_SERVER** Tableau contenant les entêtes, dossiers et chemins du script
- **$_GET** Tableau contenant les couples nom_de_variable et valeur méthode GET
- **$_POST** Tableau contenant les couples nom_de_variable et valeur méthode POST
- **$_FILES** Tableau concernant les téléchargements de fichier via HTTP
- **$_COOKIE** Les données du cookie (stockée coté client)
- **$_SESSION** Les données de sessions (stockée coté serveur)
- **$_REQUEST**
- **$_ENV**
- **$GLOBALS** Contient l'ensemble des variables précédentes

Leur utilisation sera décrite plus loin.

3.3.5 Fonctionnement

Puisque le script PHP est appelé par le service HTTP, le fonctionnement correct dépend de cet appel via l'URL, qui doit contenir l'entête HTTP:// ou HTTPS:// et l'adresse du serveur.

l'URL dans le navigateur doit impérativement comporter l'adresse IP (et éventuellement le port) du serveur web.
En effet, si le navigateur peut afficher le fichier, il ne peut pas exécuter les scripts PHP.
Exemple d'erreur classique : file:///E:/uWamp_3/www/monFichierHTML.html

3.4 DÉBOGAGE

Enfin, il ne faut pas oublier que PHP s'exécute du côté serveur : les développeurs ne souhaitent pas que les utilisateurs voient les codes d'erreurs en phase de production.

En revanche, ils souhaitent avoir des messages clairs pendant les phases de développement et de tests.

Pour cela, ils peuvent ajouter les instructions suivantes qui indiquent à l'interpréteur PHP que le script est en débogage.

```
<?php
ini_set('display_errors', 'On');
error_reporting(E_ALL | E_STRICT);
```

3.4.1 ini_set()

L'instruction ini_set() permet de modifier les variables de débogages dynamiquement. Dans un environnement de développement, la modification serait faite dans le fichier php.ini

```
display_errors = on ;
```

3.4.2 error_reporting()

Les erreurs de type "notifications" (notices) ne sont pas forcément affichées en PHP 4 et 5. Cette instruction permet donc de forcer le niveau d'affichage, grâce à '**E_ALL**' ;

L'attribut '**E_STRICT**' permet de remonter les erreurs de type "fonctions dépréciées"...

3.4.3 documentation PHP

Pour écrire du code PHP, trouver des fonctions ou bien utiliser correctement celles-ci, la documentation PHP est très bien faite.

On la trouve sur https://www.php.net/manual/fr/

4 LES INSTRUCTIONS, FONCTIONS ET VARIABLES PHP

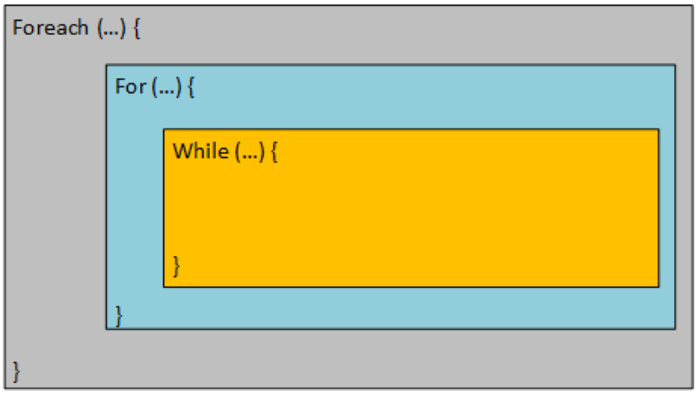
Nous allons maintenant utiliser des instructions et des fonctions disponibles dans PHP.

4.1 LES INSTRUCTIONS

Comme les autres langages étudiés, PHP dispose de boucles (répétitions) et de conditions.

4.1.1 Les boucles

La notion de répétition avec sortie prédéfinie ou conditionnelle est implémentée en PHP. Les instructions contenues dans le bloc seront répétées un certain nombre de fois. Il est bien sûr possible d'imbriquer les conditions.



4.1.1.1 For

La boucle de comptage la plus simple : on initialise la variable, on donne la condition de sortie et enfin on précise le pas d'incrémentation ou de décrémentation :

```
for ($t = 0 ; $t < 10 ; $t++) {
    echo '$t contient '.$t.'<br>' ;
}
```

4.1.1.2 Foreach

Similaire dans le fonctionnement interne, c'est PHP qui détermine le nombre d'élément d'un tableau ou d'une liste à parcourir. Le programmeur sait seulement que la liste sera parcourue dans l'ordre.

```
$arr = array(1, 5, 3, 4, 5, 1, 2, 2, 3, 4);
foreach ($arr as $value) {
    echo "Note $value /5 = ".$value*4." /20";
}
```

Ici, le programme affiche la table des notes d'une classe, multipliée par 4 pour afficher une note sur 20.

4.1.1.3 while et do… while

Il s'agit de deux boucles dont la sortie se fera lorsque la condition sera fausse. Dans le premier cas, le test est fait au début de la boucle, ce qui signifie que le bloc peut n'être jamais exécuté.

Dans le deuxième cas, on exécute les instructions du bloc puis on teste la condition à la fin.

Comparez

```
$coffre = 10 ;
$ration = 2 ;
while ($verre < 10) {
    echo "le coffre n'est pas encore plein… <br>" ;
    $coffre = $coffre + $ration ;
}
echo "le coffre est maintenant a $coffre<br>" ;
```

et

```
$coffre = 10 ;
$ration = 2 ;
do {
    echo "le coffre n'est pas encore plein… <br>" ;
    $coffre = $coffre + $ration ;
} while ($verre < 10);
echo "le coffre est maintenant a $coffre<br>" ;
```

4.1.2 Les conditions

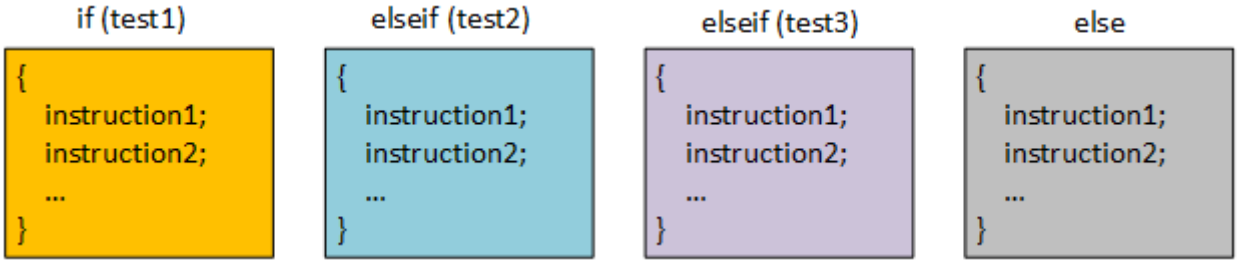
Les conditions permettent de faire des tests, de contrôler des hypothèses.

Il existe un ensemble de conditions : if, elseif et else.

Mais il y a aussi un ensemble pour les décisions nombreuses : switch, case, et default.

4.1.2.1 If elseif et else

Ces instructions permettent de créer des blocs qui ne seront exécutés que si la condition est vraie.



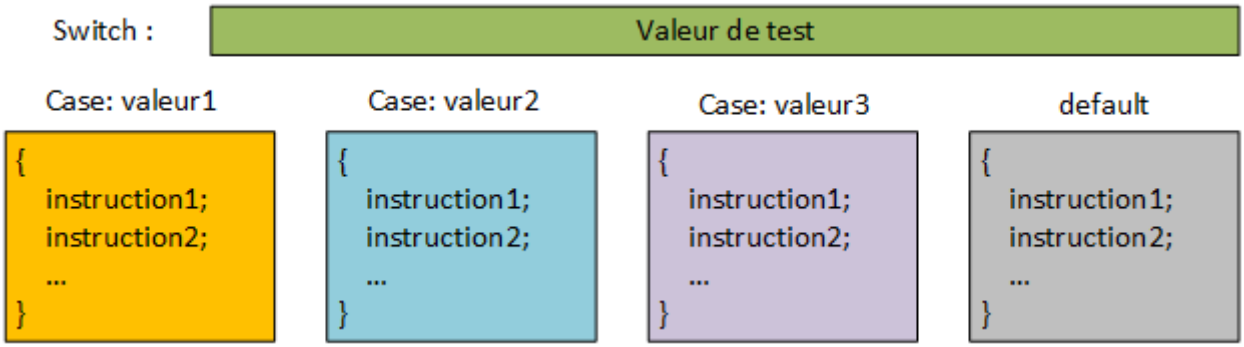
Le premier test utilise le mot-clé 'if' tandis que les suivants utilisent 'elseif'. Enfin, 'else' permet d'exécuter un dernier bloc si les autres blocs n'ont pas été exécutés.

En code PHP, cela donne les instructions suivantes :

```
if ($role == "administrateur") {
    echo "connecté en tant qu'administrateur<br>" ;
} elseif ($role == "rédacteur") {
   echo "connecté en tant que rédacteur<br>" ;
} else {
    echo "vous êtes un utilisateur<br>" ;
}
```

4.1.2.2 switch, case et default

Dans ce mode de condition, une seule valeur est comparée avec un ensemble de références.



En code PHP, cela donne les instructions suivantes :

```
switch ($avisFiabilite) {
    case 0:
        echo "0/5 : ce produit est inutilisable !";
        break;
    case 1:
        echo "1/5 : ce produit n'est pas fiable !";
        break;
    case 2:
        echo "2/5 : ce produit fonctionne parfois !";
        break;
    case 3:
        echo "3/5 : le produit est moyen !";
        break;
    default:
        echo "> 3/5 : un produit correct !";
        break;
}
```

4.2 LES FONCTIONS

Elles s'écrivent avec des parenthèses à la fin. La richesse de PHP est d'avoir de très (très) nombreuses fonctions. En voici quelques-unes…

4.2.1 appeler une fonction : date

Dans l'exemple suivant, nous utilisons la fonction date()

```
<p>Aujourd'hui nous sommes le <?php echo date('d/m/Y h:i:s'); ?></p>
```

affichera

Aujourd'hui nous somme le 28/01/2018 23:30:00

Parfois les scripts auront un message d'avertissement indiquant que la variable TIMEZONE n'a pas été définie. Il y a deux solutions possibles :

- Dans le fichier php.ini, recherchez ou ajoutez la variable date.timezone avec comme valeur le fuseau horaire du serveur[1].

```
[Date]
; Defines the default timezone used by the date functions
; http://php.net/date.timezone
date.timezone = America/New_York
```

- Dans le script php, vous pouvez forcer la valeur de timezone.

```
date_default_timezone_set('Europe/Paris');
```

Notez que la date affichée n'est pas forcément celle de votre ordinateur : le script PHP est en effet exécuté sur le serveur. Ce dernier n'est pas toujours dans le pays du visiteur ; pour afficher la date du client, on préférera utiliser une fonction Javascript.

```
<SCRIPT LANGUAGE="JavaScript">
     var maintenant=new Date();
     document.write(maintenant);
</SCRIPT>
```

4.2.2 Créer une fonction

Pour créer sa propre fonction en PHP, il suffit d'encadrer le code entre deux accolades et de préciser le nom de la fonction. Il n'est pas nécessaire d'indiquer le type de retour :

```
function monMax($a, $b) {
    // le code
    return $max ;
}
```

1 http://php.net/manual/en/timezones.europe.php

4.3 LES VARIABLES PRÉDÉFINIES

PHP propose plusieurs variables prédéfinies (portée globale), qui permettent de récupérer des informations sur l'environnement, sur les cookies, les sessions et bien sûr les données transmises via un formulaire.

4.3.1 Liste de variables superglobales

variable	utilité
$_GET $_POST	Variables contenant le retour d'un formulaire. Pour retrouver la valeur d'un champ, il faut le préciser entre les crochets : $_POST['chamNom']
$_SERVER	Variables sur l'environnement d'exécution. • $_SERVER['SERVER_ADDR'] donne l'adresse du serveur • $_SERVER['SERVER_NAME'] donne le nom DNS du serveur • $_SERVER['REMOTE_ADDR'] donne l'adresse du client • $_SERVER['REQUEST_METHOD'] indique la requête utilisée pour l'accès à la page (GET, POST, HEAD, PUT) • $_SERVER['HTTP_USER_AGENT'] indique le navigateur • $_SERVER['HTTPS'] indique si le script a été appelé via HTTPS • $_SERVER['REMOTE_USER'] donne l'utilisateur authentifié • $_SERVER['DOCUMENT_ROOT'] indique la racine du script PHP http://php.net/manual/fr/reserved.variables.server.php
$COOKIE	Même principe que précédemment, il faut indiquer quels champs est ciblé entre les crochets. $COOKIE['name']
$_SESSION	Variable utilisée dans la gestion des sessions sur le serveur (session_start()). Des exemples seront fournis plus loin.

Tableau 4.1 : exemples de variables prédéfinies

D'autres variables existent, elles sont décrites dans la documentation PHP :

http://php.net/manual/fr/reserved.variables.php

4.3.2 lecture de variables $_GET ou $_POST

Une fonction permet de lire les variables, sous la forme d'un dictionnaire de clé/valeur.

```
print_r($_POST);      // cette fonction est équivalente au code ci-après...
foreach($_GET as $key=>$val) {
  echo $key." : ".$val." <br>" ;
}
```

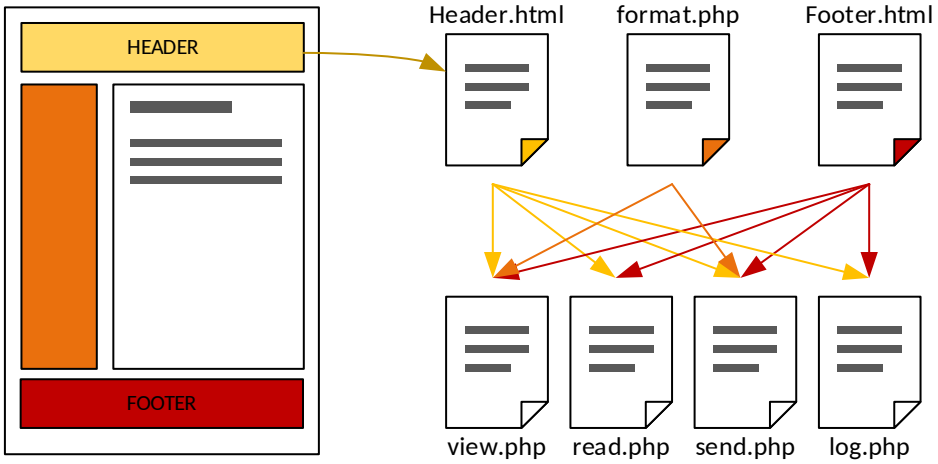
En effet, ces variables contiennent souvent plusieurs données, dont la clé est le nom du champ dans le formulaire. Voici une méthode fournie par PHP pour rapidement déboguer une application :

```
var_dump($_POST);
```

4.4 LES INCLUSIONS

PHP permet d'utiliser plusieurs fichiers lors du développement d'une application web.

C'est intéressant pour ne pas recopier plusieurs fois le même code. Le principe est simple : un fichier contient un ensemble de fonctions utiles pour plusieurs autres pages web. Il suffit de modifier ou corriger ce fichier pour que l'ensemble des autres pages bénéficient des améliorations.



Il n'est pas obligatoire que les fichiers soient dans le même répertoire : cependant, il est préférable d'indiquer le chemin d'accès en mode absolu (qui commence par **/** , ex : */images/dossier/fichier.jpg*)

Dans l'exemple ci-dessus, l'entête et le pied de page sont deux fichiers toujours identiques sur toutes les pages. Plutôt que de recopier plusieurs fois les contenus de ses fichiers, il suffit de les inclure au bon endroit de nos fichiers.

Si le site doit changer son entête (changement de marque, de logo, de couleurs), il n'y a qu'un seul fichier à modifier pour toutes les pages.

Il est possible de comparer cette inclusion comme les fonctions similaires en HTML et JavaScript :

```
<script src="js/genadresse.js" type="text/javascript"></script>
<link rel="stylesheet" type="text/css" href="source/page.css">
```

Pour utiliser les fonctions d'un fichier dans un autre fichier PHP, il y a 4 possibilités :

- require
- require_once
- include
- include_once

4.4.1 require et require_one

En utilisant les mots clés **require** ou **require_once**, on indique à PHP d'abandonner le traitement si le fichier à inclure est inexistant.

```
<?php
    require "entete.html";
?>
```

La subtile différence entre **require** et **require_once** est que dans le deuxième cas, PHP ne chargera pas le fichier s'il l'a déjà chargé une fois.

4.4.2 include et include_once

En utilisant les mots clés **include** ou **include_once**, PHP ne provoquera pas d'erreur bloquante si le fichier à inclure est inexistant.

```
<?php
    include "entete.html";
?>
```

La subtile différence entre **include** et **include_once** est la même que pour **require** et **require_once**.

4.4.3 Exemple

L'exemple ci-dessous, montre l'usage d'include avec la programmation objet (que nous verrons au prochain chapitre).

```
<?php
     include_once 'control.class.php';
     class Accueil extends Control {

          public function __construct($util, $get=null) {
               parent::__construct($util);

               $data["auth"]=$this->setViewMenu();

               $this->view->init('corpsaccueil.php',$data);
               $this->setViewBas();
          }
     }
?>
```

5 LES ACCÈS AUX BASES DE DONNÉES

Il existe deux moyens d'écrire dans une base de données MySQL (ou MariaDB) mais un seul et suffisamment sécurisé et permet d'écrire sans se soucier du type de la base.

- Si la base est MySQL, il est possible d'utiliser les fonctions mysqli (en écriture procédurale ou bien en objet) et PDO (plus sûr contre les injections SQL)
- Si la base n'est pas compatible avec MySQL, il faut envisager PDO (PHP Data Object)

Le tableau[2] ci-dessous énumère quelques avantages et inconvénients des deux méthodes.

	PDO	MySQLi
Database support	12 different drivers	MySQL only
API	OOP	OOP + procedural
Connection	Easy	Easy
Named parameters	Yes	No
Object mapping	Yes	Yes
Prepared statements (client side)	Yes	No
Performance	Fast	Fast
Stored procedures	Yes	Yes

Empêche les injections SQL

Les différences restent suffisamment lisibles pour comprendre les fonctions employées.

```
// mysqli, mode procédural
$bdd = mysqli_connect('localhost','username','password','database');

// mysqli, mode objet
$bdd = new mysqli('localhost','username','password','database');

// PDO
$bdd = new PDO("mysql:host=localhost;dbname=database", 'username',
'password');
```

Le modèle retenu pour accéder à une base de données sera **PDO**. Ce modèle permet d'accéder, lire et écrire des données depuis et vers des variables PHP.

PDO étant une extension de PHP, assurez-vous qu'elle est **disponible et active** dans le fichier `php.ini`.

```
153  ;Database Extensions
154  extension = php_mysqli.dll
155  extension = php_pdo_mysql.dll
156  ;extension = php_odbc.dll
```

Note : Vous pouvez également activer "`display_errors = On`", lors du débogage.

2 https://code.tutsplus.com/tutorials/pdo-vs-mysqli-which-should-you-use--net-24059

5.1 OUVERTURE D'UNE BASE

L'ouverture d'une base est très simple. Il faut indiquer le nom DNS du serveur de base de données ('localhost'), le nom de la table ('exemptetable'), le compte de l'utilisateur (ici 'root' mais il vaut mieux créer un utilisateur ayant les droits corrects) et le mot de passe (dans l'exemple, le champ est vide puisque par défaut, l'utilisateur 'root' n'a pas de mot de passe dans MySQL).

```
<?php
      try {
            $bdd = new PDO('mysql:host=localhost ;
dbname=exempletable','root','') ;
      }
      catch (exception $e) {
            die('Erreur de type '.$e->getMessage()) ;
      }
?>
```

Voici le même exemple utilisant des variables PHP :

```
<?php
      $server ="localhost";  // peut-être remplacé par l'adresse IP
      $base   ="exempletable";
      $userdb ="root";
      $userpwd="";
      try {
            $bdd = new PDO("mysql:host=$server; dbname=$base", $userdb,
$userpwd);
      }
      catch (exception $e) {
            die('Erreur de type '.$e->getMessage());
      }
?>
```

Peu de choses semblent avoir changées toutefois, elles sont importantes :

- On a déclaré chaque variable avec sa valeur ;
- La ligne new PDO utilise des doubles cotes à la place de simples cotes. Cela permet à PHP d'intervenir sur le texte chaque fois qu'il rencontre le symbole $ et de remplacer la variable par sa valeur.

Pour l'accès à la table, il est utile de préciser l'encodage choisi :

```
$bdd = new PDO("mysql:host=$server; dbname=$table; charset=utf8", $userdb, $userpwd) ;
```

Une autre solution consiste à utiliser les propriétés de concaténation, en créant une variable de connexion.

```
$cnx = "mysql:host=".$server.";dbname=".$table.";charset=utf8";
$bdd = new PDO($cnx, $userdb, $userpwd);
```

La variable $bdd contient l'étiquette de l'objet PDO. Il faudra désormais utiliser "$bdd-> " pour exploiter les méthodes permettant d'accéder à la base.

5.2 ÉCRITURE DANS LA BASE

Voici maintenant comment écrire dans la base.

Tout d'abord, on prépare la requête et pour éviter toute erreur si les colonnes de la table sont modifiées, on précise les attributs qui seront modifiés : ce sont donc les attributs entre parenthèses qui définissent les champs modifiés.

```
$sql = "INSERT INTO exempletable (nom, prenom, email)
    VALUES ('ROUMANET', 'David', 'david.roumanet@ac-grenoble.fr')";
$bdd->exec($sql);       // utilisation d'exec car aucune valeur de retour
```

Dans cet exemple, si la base venait à être modifiée (par l'ajout d'une colonne 'civilité' avant la colonne 'nom') les données écrites restent affectées aux bonnes colonnes.

Bien entendu, les champs autoincrémentés n'ont pas à être spécifié lors de l'insertion.

5.2.1 Écriture inexacte

La bonne exécution d'une requête n'est pas toujours garantie : il peut y avoir des erreurs et SQL bloquera la requête "silencieusement".

Voici un exemple de requête qui ne fonctionne pas :

```
$nom = "ROUMANET";
$requete = "INSERT INTO exempleTable (nom) VALUES ($nom)";
echo $requete;
$bdd->exec($requete);
```

En effet, $nom est une chaîne de caractères (et dans la base, ce doit être un champ 'TEXTE'). Or, $requete devient :

INSERT INTO exempleTable (nom) VALUES (ROUMANET)

alors qu'il faudrait avoir une forme indiquant que c'est une chaîne :

INSERT INTO exempleTable (nom) VALUES ("ROUMANET")

5.2.2 écriture correcte (2 méthodes)

Il faut donc ajouter des cotes ou des guillemets, par l'usage du caractère d'échappement \ :

```
$nom = "ROUMANET";
$requete = "INSERT INTO exempleTable (nom) VALUES (\"$nom\")";
echo $requete;
$bdd->exec($requete);
```

ou bien, insérer des cotes en utilisant le principe de la concaténation

```
$nom = "ROUMANET";
$requete = "INSERT INTO exempleTable (nom) VALUES ('".$nom."') ;
echo $requete;
$bdd->exec($requete) ;
```

Dans le doute, vous pouvez toujours copier le résultat affiché par la commande echo puis le copier dans le champ SQL de la page d'administration 'phpmyadmin'.

5.2.3 sécurité des requêtes (injection SQL)

C'est désormais ici que se trouve la principale faiblesse des sites web : l'injection SQL !

Reprenons la requête précédente mais utilisons le retour d'un formulaire :

```
$nom = $_POST['nom'];
$requete = "INSERT INTO exempleTable (nom) VALUES ('".$nom."') ;
echo $requete;
$bdd->exec($requete) ;
```

Si l'utilisateur tape ROUMANET dans le champ 'nom' du formulaire, tout va bien… mais que se passe-t-il s'il écrit : ROUMANET' OR '1' = '1' --

La requête SQL deviendra :

INSERT INTO exempleTable (nom) VALUES ('ROUMANET' OR '1' = '1' --)

Ici, il n'y a pas de risque.

En revanche, si la requête utilisait ce champ pour une recherche, nous aurions un véritable risque :

SELECT id FROM users WHERE username='username' AND password='ROUMANET' OR 1=1'

Reprenons la première requête et modifions la réponse de l'utilisateur :[3]

INSERT INTO maTable (nom) VALUES ('ROUMANET') ; DROP TABLE matable; --)

Pour rappel, le double – signifie que le reste de la ligne est commenté. La requête DROP TABLE effacera irrémédiablement la table.

Désormais, cette requête affichera tous les utilisateurs de la table users, alors qu'elle ne devait afficher que l'utilisateur saisit.

Heureusement, il existe plusieurs moyens pour limiter et éviter cela, mais l'usage de PDO est fortement recommandé, car les méthodes incluent une protection de premier niveau sur le contenu des champs.

https://www.univ-orleans.fr/iut-orleans/informatique/intra/tuto/php/bd-php-pdo-securite.html

3http://bobby-tables.com/

5.2.3.1 Méthode 1 : prepare()

En PDO uniquement, cette méthode apporte un deuxième niveau de protection. Le fait de préparer la requête revient à créer une requête à trou :

```
$req = $bdd->prepare("SELECT * FROM maTable WHERE login= ? AND password= ?");
$req->execute(array($nom, $password));
```

revient à écrire

SELECT * FROM maTable WHERE login= [ ] AND password= [ ] ;

Cette méthode bloque les attaques de niveau 1 (mais ne bloque pas toutes les attaques).

5.2.3.2 Methode 2 : mysqli_real_escape_string()

Il s'agit de la fonction que PDO utilise en interne. Ainsi, une requête mysqli_ pourra utiliser explicitement pour obtenir le même niveau de sécurité.

```
$formNom = $_POST['nom'];
$nom = mysqli_real_escape_string($formNom); //$nom sans les caractères
d'échappement
```

https://stackoverflow.com/questions/134099/are-pdo-prepared-statements-sufficient-to-prevent-sql-injection

https://www.supinfo.com/articles/single/2550-php-mysql

5.3 LECTURE DANS LA BASE

Sauf à écrire une requête spécifique, la plupart du temps vous recevrez plusieurs lignes de réponse.

Prenons une requête simple et utilisons la fonction **query()** :

```
$reponse = $bdd->query('SELECT * FROM maTable');
```

Le résultat peut-être décomposé par la fonction **fetch()** : cette fonction renvoie les données lignes par ligne, ou bien rien s'il n'y a plus de données. On peut donc créer la boucle suivante :

```
while ($ligne = $reponse->fetch()) {
     echo "nom : $ligne['nom'] \t prenom : $ligne['prenom'] \t Email :
$ligne['mail'] <br>\n" ;
}
$reponse->closeCursor();     // libère le curseur de position de la fonction
fetch()
```

6 GESTION DE SESSION

Une session est valide tant que le navigateur reste ouvert. Voici comment gérer une session.

6.1 CRÉATION D'UNE SESSION

La création d'une session doit se faire au tout **début** de la page et utilise les variables globales :

```
<?php
  session_start();
  /* début de session. Écriture d'une valeur dans la clé "newsession" */
  $_SESSION["newsession"]=$value;
  $_SESSION["age"]=$age;
?>
```

6.2 UTILISATION DE LA SESSION

Vérification des paramètres d'une session :

```
<?php
  echo $_SESSION["newsession"];
  echo $_SESSION["age"];
?>
```

Il est possible d'utiliser plusieurs variables, elles seront lisibles avec une boucle :

```
foreach($_SESSION as $cle => $valeur)
{
      echo " $cle." : ".$valeur." <br>";
}
```

6.3 MISE À JOUR D'UNE SESSION

La mise à jour reste aussi simple, et la suppression d'une valeur se fait avec unset() :

```
<?php
  $_SESSION["age"] = $nouvelAge;       // modification de la valeur de la clé 'age'
  unset($_SESSION["newsession"]);            // suppression de la clé 'newsession'
?>
```

6.4 CLÔTURE D'UNE SESSION

Pour terminer la session, il suffit de la détruire avec session_destroy() **ou** session_unset() ;

```
<?php
  session_destroy();                        // oui, c'est un peu violent ;)
  session_unset();                     // vide toutes les variables mais conserve la
```

```
session
?>
```

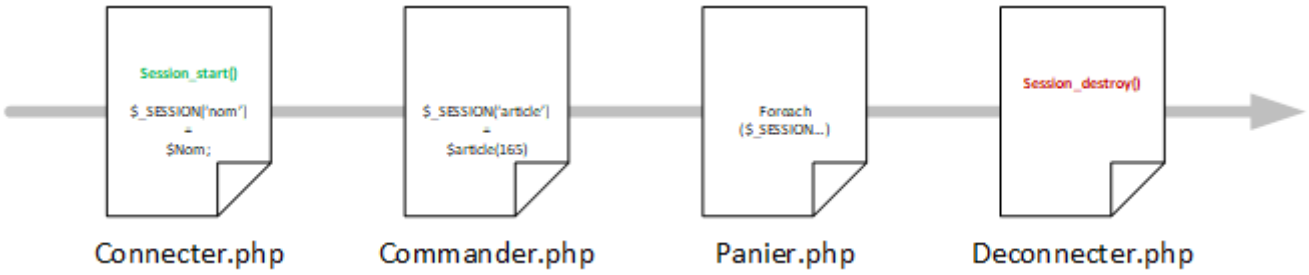
6.5 EXEMPLE D'UTILISATION

6.5.1 Exemple simplifié

Voici un exemple simple d'utilisation[4] (après avoir créé une session avec session_start()) :

```
<?php
	if ((isset($_SESSION['login'])) && (!empty($_SESSION['login'])))
	{
		// le login existe et n'est pas vide. Tentative d'accès à la page
profil
		echo '<a href="profil.php" title="Accès profil">Mon profil</a>';
	}
	else
	{
		// pas de login en session : proposer la connexion
		echo '<a href="connexion.php" title="Accès
connexion">Connexion</a>';
	}
?>
```

Le site propose également des exemples plus intéressants, l'idée principale restant la suivante :



L'utilisateur créé une session, puis les différentes pages vérifient qu'il existe bien une variable $_SESSION(['nom']) et qu'elle contient le bon utilisateur.

Lorsque l'utilisateur se déconnecte, il faut détruire la session.

Attention : certains sites proposent l'usage de unset($_SESSION), mais cela n'est pas recommandé : il s'agit de tenter de détruire une variable superglobale. Bien que ce soit possible, cela peut entraîner des bugs ou des problèmes dont vous vous passeriez bien.

4 http://www.php-astux.info/sessions-php.php

6.5.2 Message d'erreur pour formulaires

Imaginons maintenant un formulaire que l'utilisateur doit remplir, avec de nombreux champs. Il y a de fortes chances qu'il se trompe et oubli un champ. Voici comment utiliser la session :

formulaire.php

```php
<?php
// Démarrer une session
session_start();

// Vérifier si un message d'erreur est présent dans la session
if (isset($_SESSION['error'])) {
    // Afficher le message d'erreur
    echo '<p>' . $_SESSION['error'] . '</p>';

    // Supprimer l'affichage du message d'erreur pour la prochaine fois
    unset($_SESSION['error']);
}
?>
<form ....>
</form>
```

verifFormulaire.php

```php
<?php
// Démarrer une session
session_start();
if (!isPOST(['nom', 'prenom', 'age', 'ville',...]) {
     // Stocker un message d'erreur dans la session
     $_SESSION['error'] = 'Une erreur s\'est produite. Veuillez réessayer.';

     // Rediriger l'utilisateur vers une autre page
     header('Location: formulaire.php');
     exit; // Terminer l'exécution du script après la redirection
}
?>
```

6.6 ERREUR TYPE

Si vous rencontrez l'erreur "Cannot modify header information - headers already sent by ..." c'est probablement parce que les fonctions session_start() et setcookie() envoient des headers au serveur. Le header étant une portion d'information échangée avec le navigateur de l'utilisateur.

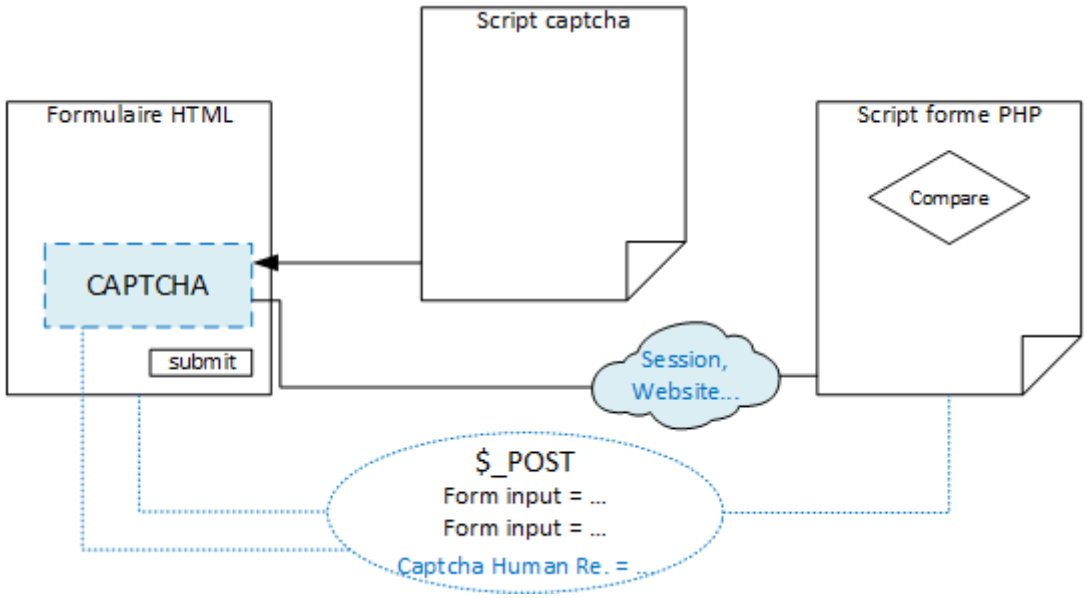
La page https://openclassrooms.com/courses/concevez-votre-site-web-avec-php-et-mysql/au-secours-mon-script-plante-1 donne également quelques conseils.

7 GESTION DES CAPTCHAS

Les "captchas" sont des systèmes pour éviter que les bots ne remplissent vos formulaires avec des données inutiles.

7.1 FONCTIONNEMENT

Il existe de nombreux systèmes de captchas, mais le principe est toujours le même :



Un script (ou une librairie) génère le captcha à l'endroit où il est appelé dans le formulaire HTML.

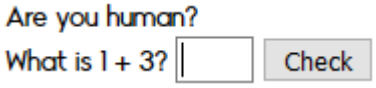
Lors de la soumission du formulaire, la réponse saisie par l'utilisateur est transmise dans la variable super globale $_POST, mais la valeur calculée par le captcha transite par un autre moyen (reCaptcha[5] de Google passe par le site de Google, Secureimage[6] utilise une session). Le script récupère la réponse et peut effectuer le traitement ou non.

7.2 MISE EN ŒUVRE

7.2.1 Solution 1 : JavaScript

Il s'agit ici de faire un système simplifié : un bot de base sera bloqué, tandis qu'un bot évolué trouvera le bouton et le validera sans utiliser le captcha. L'intérêt est principalement de montrer la collaboration entre PHP et JavaScript. N'oubliez pas que JavaScript peut ouvrir une session si besoin.

Le script propose de calculer l'addition de 2 nombres choisis au hasard.



5 https://www.google.com/recaptcha/
6 https://www.phpcaptcha.org/

7.2.1.1 fichier JavaScript

Voici le script à écrire dans un fichier captcha.js :

```
var a = Math.ceil(Math.random() * 10);
var b = Math.ceil(Math.random() * 10);
var c = a + b;

function DrawBotBoot()
{
	document.write("What is "+ a + " + " + b +"? ");
	document.write("<input id='BotBootInput' type='text' maxlength='2'
size='2'/>");
}

function ValidBotBoot(){
	var d = document.getElementById('BotBootInput').value;
	if (d == c) {
		return true;
	} else {
		alert("Réponse incorrecte. Veuillez saisir une réponse correcte.");
		return false;
	}
}
```

7.2.1.2 intégration formulaire

Voici maintenant comment le réutiliser dans votre page de formulaire :

```
<form method="post" action="depot.php" onSubmit="return ValidBotBoot()">
	<input type="hidden" name="MAX_FILE_SIZE" value="2097152">
	<input type="file" name="nom_du_fichier"><br>
	<p>moins de 2 Mo.</p>
	<script>DrawBotBoot();</script>
	<input type="submit" value="Valider">
</form>
```

L'instruction onSubmit permet en effet de soumettre le formulaire, seulement si le retour d'une fonction JavaScript est vrai (true).

7.2.2 Solution 2 : reCaptcha

La solution de Google nécessite seulement le téléchargement d'une petite librairie (90 Ko environ). La version de PHP5 est nécessaire.

7.2.2.1 Enregistrer son site chez Google

Il faut remplir le formulaire d'enregistrement d'un nouveau site : https://www.google.com/recaptcha/

7.2.2.2 Formulaire HTML

Entre les balises <head> et </head> placez la ligne suivante :

```
<script src='https://www.google.com/recaptcha/api.js'></script>
```

Puis, à l'intérieur du formulaire, entre les balises <form> et </form> la ligne fournie par Google :

```
<div class="g-recaptcha" data-
sitekey="6LcJIFIUA...clé_publique...o6uAr7"></div>
```

La clé publique est appelée "Site key" par Google

7.2.2.3 Page de récupération PHP

Dans la page de récupération du formulaire, il faut insérer la librairie recaptchalib.php dans l'entête.

```
<?php require_once "recaptchalib.php"; ?>
```

Ensuite, il faut placer le code de vérification suivant :

```
$secret = "6LcJIFIU...clé_secrète...2zWS";
$response = null;
if (class_exists('reCaptcha')) {
     $reCaptcha = new ReCaptcha($secret);

     if ($_POST["g-recaptcha-response"]) {
          $response = $reCaptcha->verifyResponse(
          $_SERVER["REMOTE_ADDR"],
          $_POST["g-recaptcha-response"]
          );
     }
} else {
     echo "classe reCaptcha non-trouvé <br>";
     die();
}

// si le captcha is ok
if ($response != null && $response->success) {
```

```
        //... actions ...
}
```

ça devrait fonctionner. Sinon, vous pouvez tester le résultat manuellement :

Voici le résultat de la fonction _submitHTTPGet($path, $data) dans la bibliothèque recaptchalip.php.

https://www.google.com/recaptcha/api/siteverify?secret=6LcJIFIUAXXXXXXXXXXXXXX2zWS&remoteip=88.170.36.111&v=php_1.0&response=03AJlzXZ6GDgVuP9gtKelaT9ZovhyyxtP-SOfQn4cR-Nr80O680As7RXQMZcF6uK3OjH7hWHRuOvMEUJ0pAoXwhnUW0Uaae1-FSP9Chb3e2T6nLgJa99GwOoUR5QA9iTNRPf7d_X1y3AJjrIOfsOmPWIoKzcmDXtDfVgs3G2gHQetxmkD9oX-YPv1keFRaRIaLaFRuwHXL3BM_Bah9L75FPzy2iwR_a9VGrajcNrCd30QAzAYnHX9vR32KXFBi7--66VCi1GQTDvzxOSqZseX9psOXFjMx8vasItAaa0hDc_Q1aSuvUatQTLwiUmLIH6jQQl0H0HpvS4ARwpmomUEp74SQwEFef1pzJ-YtmP9YotoJQgklSNM6UFhki1d2m_BHljeCn-keEAnkoP9Dor3G4AlPRWalWW9-n9WtWKyVzzZtLs6rpDGL3UUHWnUopakDFmbVz-UyllLvqzXvO0EqJbQfdtlEJtu9RA

Et son résultat dans le navigateur, il s'agit d'un fichier JSON.

8 PHP ORIENTÉ OBJET

Comme beaucoup de langage, PHP supporte une programmation classique et une programmation objet.

Généralement considérée comme plus abstraite, la programmation objet reste pourtant facile à utiliser, la plupart des difficultés provenant du vocabulaire.

- **Objet** ► Classe (bloc comprenant des variables et des procédures)
- **Attributs** ► variables décrivant les propriétés d'un objet
- **Propriétés** ► variables décrivant les attributs d'un objet (!)
- **Méthodes** ► actions permettant d'agir sur les propriétés de l'objet
- **Instanciation** ► action particulière nécessitant l'usage du mot clé "new" pour construire une version d'un objet en mémoire.
- **Constructeurs** ► Méthodes (ou procédures) qui ne renvoient pas de résultats mais qui porte le nom de la classe (ou bien le nom __Construct(arg) en PHP). Il peut y avoir plusieurs constructeurs.
- **Héritage** ► Capacité d'un objet à reprendre l'ensemble des caractéristiques d'une autre objet. La classe qui hérite pourra avoir de nouveaux attributs en plus des attributs (et méthodes) de la classe héritée.

8.1 EXEMPLE D'OBJET

Le codage d'un jeu de rôle offre le meilleur exemple pour montrer comment s'utilise l'objet. L'important est de bien différencier la description de l'objet (l'écriture de la classe) et son instanciation (construction des objets).

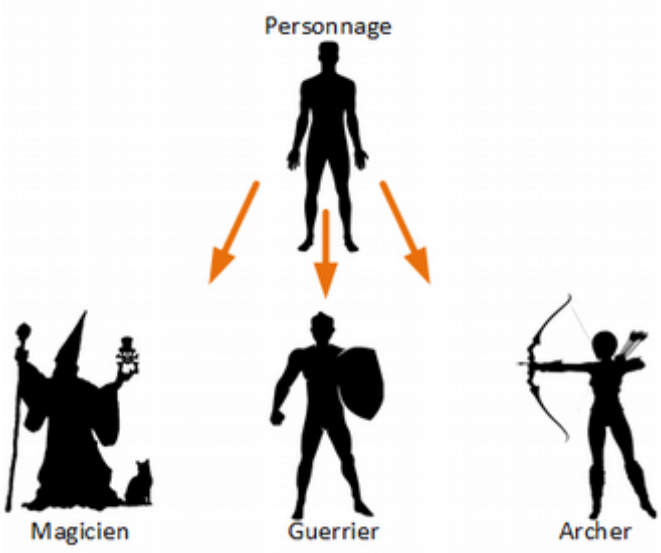
Ainsi, les caractéristiques (les attributs) d'un personnage ont en commun l'intelligence, la force, l'adresse et les points de vie.

La classe **Homme** contient donc ces attributs.

Maintenant, la classe **Magicien** est un Homme qui a de la magie (mana), des soins et de la sagesse. La classe Magicien hérite de la classe Homme et on ajoute les attributs manquants à Homme, c'est-à-dire, magie, soins et sagesse.

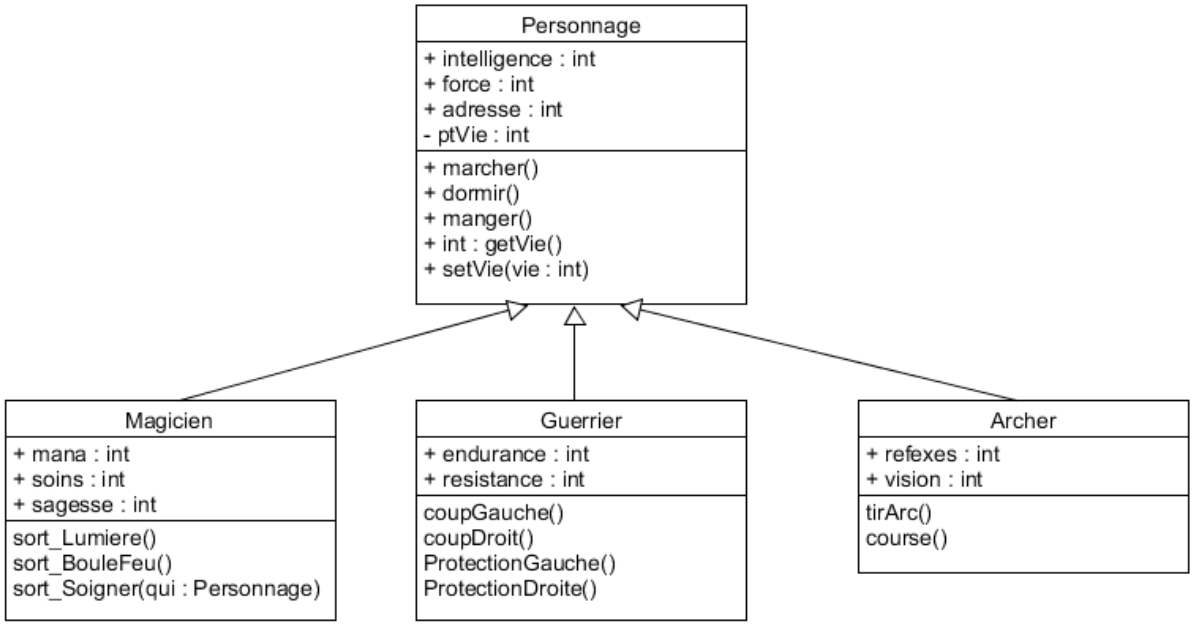
Enfin, le **Guerrier** dispose de capacités (attributs) d'endurance et résistance. Il hérite donc de la classe Homme aussi et on ajoute dans la classe Guerrier deux attributs : endurance et résistance.

La dernière classe sera un **Archer**, avec des attributs réflexes et vision.



8.2 Représentation (UML)

Utilisons d'abord la notation UML pour représenter les 3 classes.



Les symboles **+** représentent un attribut public et **–** représentent un attribut privé.

Notez le sens des flèches pour l'héritage : Magicien hérite de Personnage.

8.3 Rédaction de la classe Personnage

Nous pouvons écrire le code correspondant pour la classe Personnage *(ici en PHP)* :

```
Class Personnage {
	public  $_intelligence;
	public  $_force;
	public  $_adresse;
	private $_ptVie;

	public function marcher() {...}
	public function dormir() {...}
	public function manger() {...}
	public function getVie() {
		return ptVie;
	}
	public function setVie($vie) {
		$_ptVie = $vie;
	}
}
```

Attention, l'attribut *$_ptVie* étant privé, il ne sera pas accessible pour des modifications en dehors de la classe. C'est pourquoi les méthodes *getVie()* et *setVie($vie)* qui sont publiques, permettent de modifier $_ptVie : elles sont à l'intérieur de la classe.

8.4 Rédaction des classes filles de Personnage (héritage)

Nous pouvons maintenant créer les classes "Magicien", "Guerrier" et "Archer" : elles héritent de la classe "Personnage", évitant au développeur de devoir réécrire le code concernant ces attributs.

```
Class Magicien :: Personnage {
      public  $_mana;
      public  $_soins;
      public  $_sagesse;

      public function sort_Lumiere() {...}
      public function sort_BouleFeu() {...}
      public function Soigner($qui) {...}
}

Class Guerrier :: Personnage {
      public  $_endurance;
      public  $_resistance;

      public function coupGauche() {...}
      public function coupDroit() {...}
      public function ProtectionGauche() {...}
      public function ProtectionDroite() {...}
}

Class Archer :: Personnage {
      public  $_reflexe;
      public  $_vision;

      public function tirArc() {...}
      public function course() {...}
      public function approcheDiscrete() {...}
}
```

8.5 Instanciation d'un guerrier

Maintenant, il suffit de créer personnage directement avec son type :

```
$Conan = new Guerrier();
$Conan->_intelligence = 4; //sur 10
$Conan->_force = 8;        //sur 10
$Conan->_adresse = 4;      //toujours sur 10
```

```php
$Conan->setVie(54);          // sur 100... et oui, il faut utiliser la méthode !
```

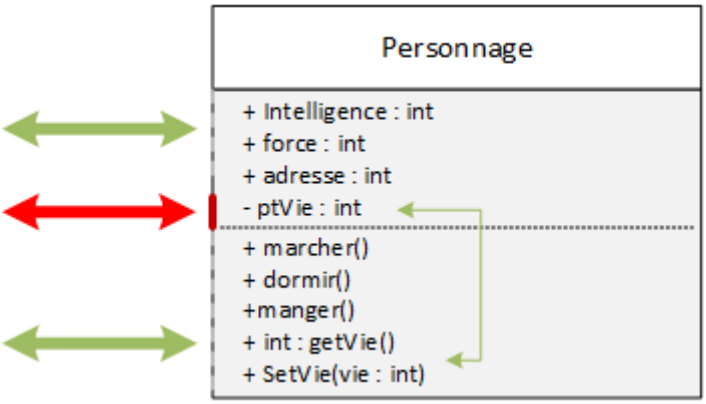
Notez que la variable privée n'étant accessible directement, on utilise une fonction…

8.6 ACCESSEUR, MUTATEUR ET CONSTRUCTEURS

Dans le paragraphe précédent, nous avons vu que les attributs privés nécessitent une méthode personnalisée pour être modifiés ou lus.

8.6.1 Assesseur (get) et mutateurs (set)

En effet, le diagramme de classe peut être interprété comme suit :



Le signe **+** indique un membre public. Il est donc accessible directement.

Le signe **–** indique au contraire un membre privé : il n'est accessible qu'aux membres à l'intérieur de la classe et donc, pas aux membres extérieurs. En revanche, il est accessible aux classes filles (qui héritent).

Le signe **#** indique un membre protégé (protected) : seule la classe courante peut y accéder, les classes filles ne peuvent pas.

Les méthodes *getVie()* et *SetVie()* sont des membres publics à l'intérieur de la classe : ils peuvent donc accéder au membre privé *ptVie*.

On appelle donc une méthode qui lit un attribut, un **accesseur** (en anglais, **get**ter).

On appelle une méthode qui modifie un attribut, un **mutateur** (en anglais, **set**ter).

Les notions d'héritage permettent également à une classe fille d'utiliser ses propres méthodes pour lire les attributs privés de la classe mère. L'inverse n'est pas vrai (la classe mère ne peut accéder aux attributs de la classe fille).

L'intérêt d'utiliser une méthode pour lire ou modifier un attribut, est de sécuriser le programme : l'utilisateur ne peut plus écrire une valeur qui ne serait pas autorisé. Par exemple, le contrôle de saisie d'un nombre en pourcentage est compris entre 0 et 100 %.

Sans rendre l'attribut privé, un utilisateur pourrait écrire 1 255 852 dans cet attribut.

Si l'attribut est privé, il doit utiliser la méthode fournie par le développeur. Cette dernière pourra renvoyer une erreur si le nombre n'est pas dans l'intervalle attendu.

8.6.2 Les constructeurs

Par défaut, dès qu'une classe est créée, elle dispose d'un constructeur commis d'office : il créera les tous les attributs en mémoire, mais en les laissant vides (chaînes, tableaux, etc.) ou à zéro (entiers, flottants…)

Dans la plupart des langages, les constructeurs portent le nom de la classe et se différencient par le nombre et le type des arguments, mais en PHP, il ne peut exister qu'un seul constructeur et il porte le nom **__construct()**.

Voici un code d'exemple (tiré du site officiel) :

```
class BaseClass {
    function __construct() {
        print "In BaseClass constructor\n";
    }
}
```

Le fait de créer sa propre méthode de construction de l'objet, permet d'utiliser les arguments pour remplir les attributs.

Voici un autre exemple plus concret :

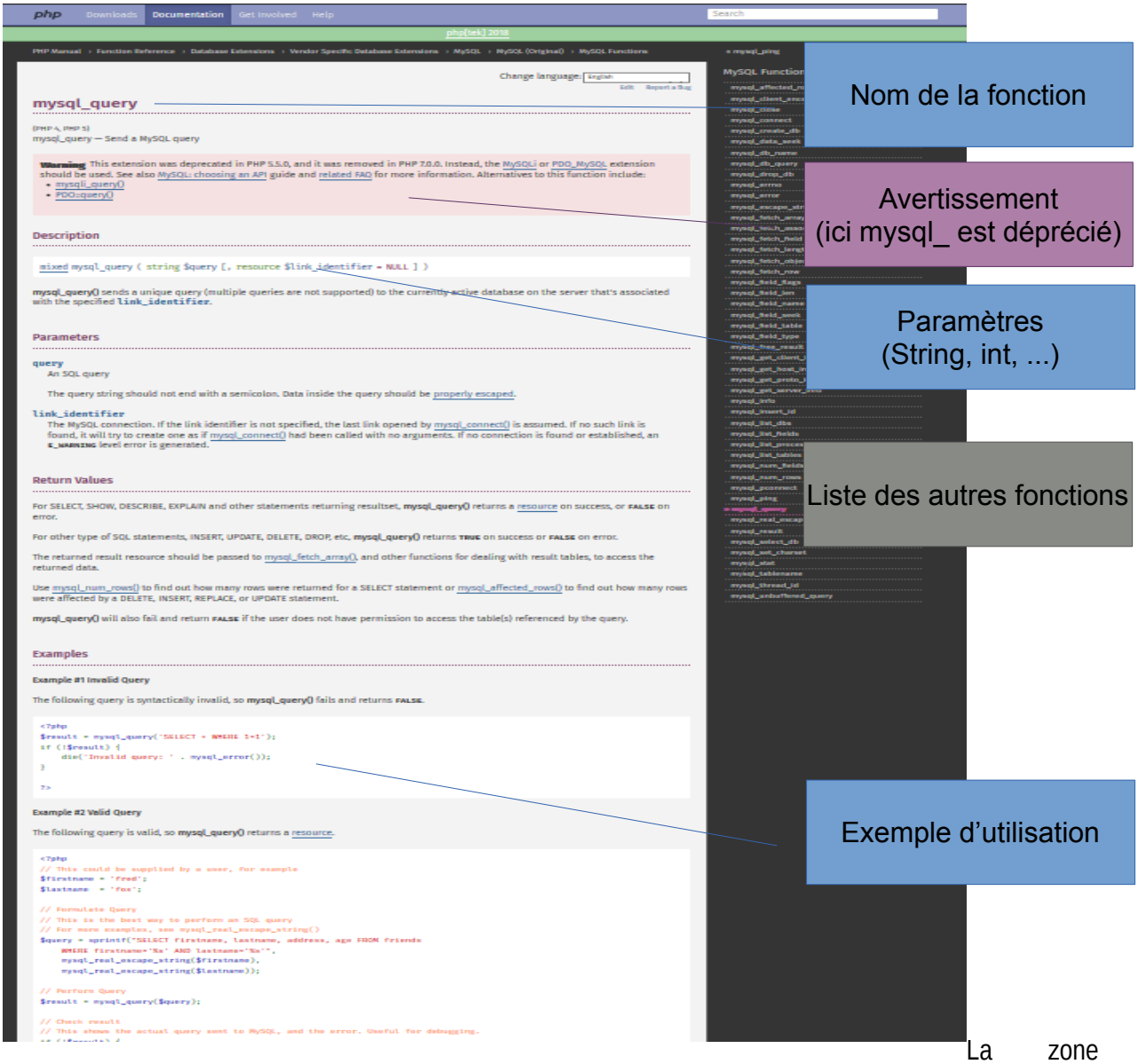
```
class ConnexionBase {
    protected $link;
    private $server, $username, $password, $db;

    public function __construct($server, $username, $password, $db)
    {
        $this->server = $server;
        $this->username = $username;
        $this->password = $password;
        $this->db = $db;
    }
    …
}
```

Le mot-clé **this** permet de distinguer les attributs de la classe et les attributs transmis en arguments.

9 UTILISATION DE LA DOCUMENTATION PHP

Voici la vue générale de l'aide en ligne de PHP, sur l'exemple **mysql_query()** :



La zone de paramètres indique comment utiliser la fonction : comme très souvent dans les documentations, les paramètres entre [ ] sont optionnels.

```
mixed mysql_query ( string $query [, resource $link_identifier = NULL ] )
```

option

Les exemples d'utilisations sont utiles pour voir les fonctions associées et les paramètres réels.

```php
<?php
// This could be supplied by a user, for example
$firstname = 'fred';
$lastname  = 'fox';

// Formulate Query
// This is the best way to perform an SQL query
// For more examples, see mysql_real_escape_string()
$query = sprintf("SELECT firstname, lastname, address, age FROM friends
    WHERE firstname='%s' AND lastname='%s'",
    mysql_real_escape_string($firstname),
    mysql_real_escape_string($lastname));

// Perform Query
$result = mysql_query($query);

// Check result
// This shows the actual query sent to MySQL, and the error. Useful for debugging.
if (!$result) {
    $message  = 'Invalid query: ' . mysql_error() . "\n";
    $message .= 'Whole query: ' . $query;
    die($message);
}

// Use result
// Attempting to print $result won't allow access to information in the resource
// One of the mysql result functions must be used
// See also mysql_result(), mysql_fetch_array(), mysql_fetch_row(), etc.
while ($row = mysql_fetch_assoc($result)) {
    echo $row['firstname'];
    echo $row['lastname'];
    echo $row['address'];
    echo $row['age'];
}

// Free the resources associated with the result set
// This is done automatically at the end of the script
mysql_free_result($result);
?>
```

La génération de la variable qui sera utilisé pour l'appel à la fonction : Utilisation de fonctions inconnues (par exemple : mysql_real_escape_string();

La fonction elle-même

Des commentaires sur la gestion des erreurs

Des conseils sur les fonctions utiles fréquemment utilisées avec la fonction expliquée.

9.1 SOURCES

https://blog.teamtreehouse.com/how-to-debug-in-php