***Le PHP :***

Le PHP dans une page HTML est appelé via une balise :

<?php

?>

Pour afficher quelque chose grâce à PHP, on utilisera le mot clef ***echo .***

<?php

echo "Bienvenue dans le monde du PHP";

?>

Le console.log qu’on utilise sur JS pour vérifier une variable est différente sur PHP.

En effet, il faut utiliser ***var-dump(contenu à tester) ;***

Contrairement à JS, il n’y a pas de console et le résultat apparaîtra directement sur la page.

Dès lors qu’il y a du PHP dans une page, alors cette page devra avoir l’extension .php et non .html .

Pour concaténer, en JS on va utiliser +.

***En PHP on va utiliser un point.***

echo "bonjour "." monsieur";

On peut afficher avec des guillemets et des apostrophes.

Toutefois, les apostrophes vont rendre un texte littéral alors que les guillemets vont interpréter les variables ou autres qui sont à l’intérieur :

$var1='Hello DWWM';

echo '$var1';

echo '<br>';

echo "$var1";

// Ici, le premier $var1 va être affiché comme écrit alors que le deuxième entre guillemets va appliquer

// la variable et lire Hello DWWM.

On peut mettre un nombre entier (integer) en variable. Pour un float (nombre à virgule), il faut séparer les chiffres par un point :

$x = 147;

$y = 147.8;

Pour les entiers, on peut aller de -2 000 000 000 à + 2 000 000 000. Pour les float, on peut aller bien plus loin.

Les variables :

Pour une variable, il suffit simplement de mettre le signe $ :

$x=10;

Les variables booléenes (true or false) :

Le booléen que l’on écrit dans JS marche de la même manière dans PHP.

// Ici, on vient dire que 5 est supérieur à 2.

$x = (5>2);

// On vient tester si 5 est supérieur à 2 alors affiche vrai, sinon faux. Comme 5 est

// effectivement supérieur à 2 alors il va afficher true.

if($x){

echo "true";

}

else {

echo "false";

}

Les incrémentations :

Comme sur JS, les incrémentations seront différentes selon l’ordre dans lequel on met les signes.

$x=10;

// Ici, serveur va afficher 10 et ensuite lui ajouter 1.

echo $x++ . "<br>";

$t=10;

// Ici, il va d'abord ajouter 1 et ensuite afficher le résultat, ce qui donnera 11.

echo ++$t;

Pour les tests en if/else ou autre (les for, …), il n’y a aucune différence avec javascript.

Les fonctions :

Elles s’écrivent comme dans JS :

function bienvenue(){

echo 'bienvenue chez nous!';

}

bienvenue();

function bienvenue($x){

echo 'bienvenue '.$x.' !'.'<br>';

}

bienvenue('Marc');

bienvenue('Marie');

Les variables externes dans les fonctions :

Ici, contrairement à JS, il y a une commande pour utiliser une variable extérieure à la fonction :

$e=["marc","sophie","marie","paul"];

function tableau($x){

// Ici, le global va préciser que la variable e extérieure à la fonction va être utilisée.

// Si on ne le précise pas, il ne connaitra pas la variable e.

global $e;

echo 'bienvenue '.$e[$x].'<br>';

}

tableau(0);

tableau(1);

tableau(2);

tableau(3);

Il est donc possible d’utiliser un nom de variable propre à une fonction qui n’existera que dans cette fonction. Bien évidemment, il faut toujours avoir des noms différents pour éviter les erreurs.

Une variable appelée en global, si elle est modifiée dans une fonction, restera modifiée après être passée dans la fonction. Il faut prendre ça en considération pour éviter les erreurs.

Les modifications de variables :

Quand on appelle une variable, on ne fait qu’afficher son résultat ou s’en servir pour arriver à un résultat. La variable en elle-même ne change pas. Il est possible de changer l’adresse mémoire d’une variable et donc de modifier son résultat  avec le signe & :

function foo(&$var) {

$var++;

}

$a=5;

foo ($a);

// $a vaut 6 maintenant

Les paramètres par défaut :

On peut rentrer des variables qui ont un paramètre par défaut dans une fonction. Cependant,

Les paramètres par défaut doivent toujours être les derniers de la liste.

// Ici, l'âge par défaut est donc 11 ans.

function agg($nom,$age=11){

echo "nom: ".$nom." , âge =".$age. " ans. <br>";

}

agg("marie");

agg("max");

agg("solange");

// Ici, on veut modifier ce qui va être affiché après âge = donc on vient le préciser.

agg("pierre",47);

Les tableaux :

Il existe deux types de tableaux : les tableaux indéxés et les tableaux associatifs.

Les tableaux indexés sont pareil que sur JS. On va appeler tel élément grâce à [x].

Ici, au lieu de .length on utilisera le mot count.

Les tableaux associatifs sont différents. On va associer une valeur à un mot, afin que l’on puisse appeler la valeur en mettant le mot entre crochets.

// Créer un tableau:

$x= array();

// ou

$x=[];

// Pour le créer.

// LES TABLEAUX INDEXES:

// Ils se créent comme dans JS, il faut ajouter les valeurs dans un ordre précis et appeler un

// numéro précis pour avoir la valeur correspondante. Comme partout, la première valeur du tableau est 0.

// Pour ajouter une valeur à un tableau à la suite:

$x[count($x)]="toto";

// Ici, le count remplace le .length et vient directement compter le tableau et ajouter à la suite.

// Pour ajouter à une place en particulier ou remplacer une valeur:

$x[2]="Jean";

// Ici, on cible la donnée que l'on veut remplacer.

// LES TABLEAUX ASSOCIATIFS:

// Ici, on va associer une valeur à un mot afin de ressortir la valeur en appellant le mot:

// La flèche => va définir que telle mot renvoie à telle valeur.

$z=["nom" =>"Maureau", "prenom" => "Maurice", "age" => 38];

// Si on veut sortir le prénom:

echo $z["prenom"];

Les tableaux associatifs renvoient à une personne en particulier. Il est possible de stocker plusieurs tableaux associatifs dans un tableau indexé par exemple.

Les implode, explode et str\_split :

Il esiste des variables pour agir sur les tableaux :

// IMPLODE:

// Implode permet de créer un string à partir d'un tableau et les séparer par le premier paramètre

// que l'on rentre:

$x=["marc","sophie","jean","louise","hector"];

// var\_dump est l'équivalent php du console.log.

var\_dump(implode(",",$x));

echo "<br>";

// EXPLODE:

// Explode permet de créer un array à partir d'une string. Le premier paramètre va permettre de dire quel caractère va permettre

// de séparer la string:

$str="un,deux,trois,quatre";

// Ici, il va chercher les virgules dans la string et saparer ce qu'il y a entre les virgules.

var\_dump(explode(",",$str));

echo "<br>";

// STR\_SPLIT:

// str\_split va prendre une string et la diviser tout les x éléments, définis en second paramètre pour la transformer en array:

$str2="abcdefghijklmnopqrstuvwxyz";

// ici, l'alphabet sera divisé toutes les 5 lettres et chaque portion équivaudra à une valeur du tableau.

var\_dump(str\_split($str2,5));

Parcourir un tableau associatif :

$x=["nom"=>"marc","prenom"=>"pierre","age"=>21];

// ici, on vient dire que pour chaque valeur du tableau $x

// as $k(nom de valeur au pif qui fait référence au mot key) => $v(nom de valeur au pif qui fait référence au mot value)

// on renvoie le nom de la clé et la valeur qu'elle représente.

foreach ($x as $k=>$v){

echo"key=".$k." valeur =".$v."<br>";

}

On utilisera le mot clef foreach pour aller chercher chaque key et la value à laquelle elle rapporte.

Les constantes :

Ce sont des valeurs que l’on utilisera dans tout le programme et que l’on ne pourra jamais modifier.

Ce sont les seules variables qui ne prennent pas de $. Par convention, on les écrit en majuscules pour les différencier.

// Pour definir une constante:

define("SOCIETE","Microsoft");

// ici, à chaque fois qu'on appellera SOCIETE alors Microsoft ressortira.Il faut impérativement respecter les majuscules.

// Il est possible de créer une variable qui n'est pas case sensitive:

define("SOCIETE","Microsoft",true);

// avec le true, même si on écrit societe sans majuscules alors ça marchera.

Afficher un tableau lui-même contenu dans un tableau :

// Ici, on a un tableau. Dans ce tableau est stocké un autre tableau qui lui-même contient un autre tableau.

$x=array(1,2,3,array(10,20,30,array(100,200,300)));

// Pour afficher les données des tableaux internes:

// On va créer une fonction qui va lire les tableaux et les afficher: On met un paramètre pour ensuite l'utiliser avec un

// tableau en paramètre.

function returnArray($tab){

// Pour chaque élément ($v) contenu dans le tableau ($tab):

foreach($tab as $v){

// Si l'élément v est un tableau:

if(is\_array($v)){

// Ici, la fonction s'appelle elle-même pour dire: "affiche les éléments contenus dans le tableau". On met $v en

// paramètre pour en afficher les éléments si c'est un tableau.

// Comme on est rentré dans le tableau interne (grâce au if),alors tout ce qui est contenu dans le tableau

// va s'afficher.

returnArray($v);

}

// Si l'élément v n'est pas un tableau, alors on va l'afficher.

else{

echo $v."<br>";

}

}

}

// Ici, on va appeler la fonction et l'utiliser sur le tableau $x. Les éléments du tableau qui sont eux-même des tableaux

// vont s'afficher à la suite puisque la fonction est une boucle qui va traiter le cas où les valeurs sont un tableau.

returnArray($x);

La fonction sans les commentaires pour la réutiliser :

function returnArray($tab){

if(is\_array($v)){

returnArray($v);

}

else{

echo $v."<br>";

}

}

}

returnArray($x);

Tester des tableaux associatifs stockés dans un tableau indexé :

function returnArray($tab)

{

// Ici, pour chaque tableau construit de manière associative (donc construit de manière $k=>$v):

foreach ($tab as $k => $v) {

// Si $v est un tableau:

if (is\_array($v)) {

// On utilise la fonction avec $v en paramètre pour afficher ce qu'il y a dans ce tableau:

returnArray($v);

} else {

// Montre de quelle manière afficher les éléments qui ne sont pas des tableaux:

echo $k . " " . $v;

}

}

}

returnArray($x);

Sans les commentaires :

function returnArray($tab)

{

foreach ($tab as $k => $v) {

if (is\_array($v)) {

returnArray($v);

} else {

echo $k . " " . $v;

}

}

}

returnArray($x);

Le return :

Comme dans JS, on utilise return pour retourner un résultat après un test de fonction. Le return permet de sortir de la fonction. Il est donc inutile d’écrire quelque chose dans une fonction après un return, puisque s’il est appliqué alors on sort de la fonction et ce qui vient ensuite ne sera jamais lu par l’ordinateur.

// Ici, on vient dire que si le premier paramètre est plus grand que le deuxième alors on retourne le premier,

// sinon le deuxième. Si le premier est plus grand alors le return fait directement sortir de la fonction.

function maxab($a,$b){

if($a>$b){

return $a;

}

else{

return $b;

}

}

$c= maxab(2,7);

echo $c;

// Ici, le viewport va donc afficher 7.

Les formulaires :

***La différence entre post et get :***

Le get va transférer les infos via l’URL (donc les mdp également aussi, il faut faire attention). Alors que le post va envoyer les informations via un header caché.

Pour récupérer les infos d’un formulaire :

<form action="test\_get.php" method="get">

<label >Nom</label> <input type="text" name="nom"> <br>

<label >Prenom</label> <input type="text" name="prenom"> <br>

<label >Age</label> <input type="text" name="age"> <br>

<input type="submit" value="envoyer"><br>

</form>

***Si on veut reprendre les infos en get :***

// Le $\_GET est une variable super globale. Elle est toujours la même et générée via PHP. Il faut donc l'écrire

// telle qu'elle.

$nom=$\_GET['nom'];

$prenom=$\_GET['prenom'];

$age=$\_GET['age'];

echo "votre nom est: <b><u>".$nom."</u></b> <br>";

echo "votre prenom est: <b><u>".$prenom."</u></b> <br>";

echo "votre âge est: <b><u>".$age."</u></b> <br>";

?>

***Si on veut reprendre les infos en post :***

<!-- On met évidemment le method="post" en HTML. Le action = « » fait référence au fichier PHP qui va créer les variables. -->

<form action="bienvenue.php" method="post">

<label >Nom</label> <input type="text" name="nom"> <br>

<label >Prenom</label> <input type="text" name="prenom"> <br>

<label >Age</label> <input type="text" name="age"> <br>

<input type="submit" value="envoyer"><br>

</form>

***$\_POST est également une super globale.***

$nom=$\_POST['nom'];

$prenom=$\_POST['prenom'];

$age=$\_POST['age'];

echo "votre nom est: <b><u>".$nom."</u></b> <br>";

echo "votre prenom est: <b><u>".$prenom."</u></b> <br>";

echo "votre âge est: <b><u>".$age."</u></b> <br>";

Un autre exemple avec un text-area ;

<form action="bienvenue.php" method="get">

<textarea name="text" id="" cols="30" rows="10"></textarea><br>

<input type="submit" value="envoyer"><br>

</form>

Le fichier de sortie « bienvenue.php » va afficher le texte entré dans le text area. On le verra également dans l’URL parce qu’on utilise la méthode GET.

$text=$\_GET["text"];

echo $text;

On peut utiliser le GET en remplissant directement l’URL à la main. Pas besoin de passer par des formulaires pour faire les tests !

Compter la longueur d’une string :

function longueurChaine($a){

// strlen permet de compter la longueur d'une string.

echo strlen( $a);

}

$alphabet="abcdefghijklmnopqrstuvwxyz";

longueurChaine($alphabet);

Remplacer un caractère ou une chaîne de caractères dans une string :

Il est possible de remplacer le contenu d’une string grâce à la fonction str\_replace :

$alphabet="abcdefghijklmnopqrstuvwxyz";

function remplaceCaractere($a){

// 1/ Le premier paramètre va définir ce que l'on veut remplacer.

// 2/ Le second ce par quoi on veut le remplacer.

// 3/ Enfin le dernier va servir à définir dans quelle string ou array on veut le remplacer.

echo str\_replace("e","\_",$a);

}

remplaceCaractere($alphabet);

Les strings en majuscule ou minuscule :

function listVal($tab)

{

foreach ($tab as $k => $v) {

// strtoupper va afficher le paramètre en majuscules. strtolower les affichera en minuscules.

echo strtoupper($k) ." ". strtolower($v) . "(" . strlen($v) . ") <br>";

}

}

listVal($tableau);

Obtenir l’adresse ip :

Pour obtenir une adresse IP, il faut utiliser la super globale $\_SERVER :

echo $\_SERVER['REMOTE\_ADDR'] ;

Obtenir le navigateur :

Il faut également utiliser la super globale $\_SERVER :

// Permet de voir le navigateur utilisé par la personne.

$nav = $\_SERVER['HTTP\_USER\_AGENT'];

echo $nav."<br>";

Pour voir le nom du fichier courant :

// Permet de voir le nom du fichier courant et son dossier parent.

$path = $\_SERVER['PHP\_SELF'];

// Cette commande permet de n'afficher que le nom du fichier courant.

$file = basename ($path);

echo $path;

Créer un nouveau fichier via PHP :

// Cette variable renvoie au nom de fichier et le type (ici .txt):

$file = 'php\_ex2.txt';

// Cette variable définit le contenu à ajouter.

$content = "Vive le PHP";

// Ecrit le contenu dans le fichier, en utilisant le drapeau

// FILE\_APPEND pour rajouter à la suite du fichier et

// LOCK\_EX pour empêcher quiconque d'autre d'écrire dans le fichier

// en même temps

file\_put\_contents($file, $content, FILE\_APPEND | LOCK\_EX);

file\_put\_contents permet de créer n’importe quel type de fichier de manière concise et efficace. Cette fonction est l’ensemble de plusieurs fonctions qui permettent à elles toutes de faire la même chose.

Compter les lignes d’une page :

// la variable homepage cible la page du fichier courant:

$homepage = basename($\_SERVER['PHP\_SELF']);

// Cette variable permet de compter le nombre de lignes de la page.

$noOfLines = count(file($homepage));

echo "Il y a <b>".$noOfLines."</b> lignes dans la page ".$homepage;

Enlever les doublons d’un tableau :

$tabex7 = [1, 1, 1, 2, 3, 5, 5, 6, 7, 9];

// Pour enlever les doublons d'un array, on utilise array\_unique. Cette commande ne fonctionne pas sur

// Les tableaux multi-directionnels.

$tabSansDoublon=array\_unique($tabex7);

print\_r($tabSansDoublon);

Opérer une redirection automatique :

// Le header(location:lien) permet de rediriger automatiquement vers le lien ciblé. Au lancement du fichier, on sera

// directement sur la page du lien.

header("location:http://www.jeuxvideo.com/forums/0-51-0-1-0-1-0-blabla-18-25-ans.htm");

La fonction isset :

Pour éviter les erreurs en cas de formulaire non rempli où la valeur est nécessaire pour qu’un site fonctionne par exemple on peut mettre une valeur par défaut.

On utilise la fonction ***isset*** pour définir que si la valeur a été rentrée, alors on va l’utiliser, sinon on aura une valeur par défaut.

// On dit que si rien n'a été ajouté dans l'URL (via un form par exemple) alors la valeur n sera de 5

// par défaut.

$n=5;

// En revanche, si l'URL propose une valeur pour n, dans ce cas on utilisera cette valeur.

if(isset($\_GET['n'])){

$n=$\_GET['n'];

}

Le isset marche aussi en POST. Il marche dès lors qu’il faut utiliser une valeur par défaut si l’utilisateur n’en a pas ajouté une.

Transformer du binaire en décimales :

Le binalre est uniquement composé de valeurs en 0 et 1 (et est compté en interval 2).

Pour transformer le binaire en décimales, il existe une function appelée bindec :

$a = array(

// Ce que l'on met à l'intérieur d'un bindec est une suite binaire que l'on veut transformer en décimales.

bindec("101111111"),

bindec("101100101"),

bindec("101111111"),

bindec("101001101"),

bindec("111111101"),

);

// Chaque bindec du tableau donne les nombres suivants:

//$a=array(383,357,383,333,509);

for ($i = 0; $i < 5; $i++) {

for ($j = 8; $j >= 0; $j--) {

// La function pow() sert à définir la puissance. Le premier chiffre équivaut au chiffre, le deuxième

// a la puissance que l'on veut lui attribuer. Par exemple, ici ça sera 2^$j.

if (($a[$i] & pow(2, $j)) == pow(2, $j)) {

echo "\*";

} else {

echo "&nbsp;";

}

if ($j % 3 == 0) {

echo '&nbsp;';

}

}

echo '<br>';

}

echo '<hr>';

Les classes :

// Par convenance, on met TOUJOURS la première lettre de la classe en majuscule.

class Personne{

// On déclare les propriétés de la classe

public $nom;

public $prenom;

// On définit les méthodes qui vont utiliser les propriétés de la classe. Une méthode est toujours une fonction.

function affiche(){

// Le $this est comme dans JS. Il servira à appeler une valeur propre à une variable.

echo 'je suis '.$this->$prenom.' '.$this->$nom.'<br>';

}

}

// Pour ajouter un objet de type Personne (la classe)

$marDel= new Personne();

// On utilise la flèche pour dire telle variable aura pour valeur prenom Marc. Comme la variable est au début,

// Il ne faut pas la remettre a la propriété.

$marDel->prenom='Marc';

$marDel->nom='Delpot';

// On dit que pour telle variable on applique la méthode de la classe Personne (d'où la flèche).

$marDel->affiche();

Les get et set dans les classes :

Les getters et setters sont basés sous le même principe que ceux utilisés en JavaScript :

class Person2

{

public $nom;

public $prenom;

function setNom($str){

// La variable précisera le prénom à entrer

$this->nom=$str;

}

function getNom(){

// Cette fonction viendra spécifier la valeur de la variable x.

return $this->nom;

}

function setPrenom($str){

$this->prenom=$str;

}

function getPrenom(){

return $this->prenom;

}

function affiche(){

echo 'Je suis '.$this->prenom.' '.$this->nom.'<br>';

}

}

$marcDel = new Person2();

// On précise ici ce que l'on veut ajouter comme objet avec les propriétés de la classe Person2

$marcDel->setNom('Delpot');

$marcDel->setPrenom('marc');

$marcDel->affiche();

Les constructeurs :

En PHP, il est possible de créer une fonction qui vient directement créer des objets avec les propriétés et les méthodes souhaitées :

Cette fonction se lance automatiquement dès que l’on tape le mot « new ». Il faut mettre deux underscores construct (\_\_construct) avant le nom de notre fonction pour en créer un.

Cette fonction reste une fonction comme une autre et peut tout faire comme une fonction classique.

On peut par exemple créer un constructeur qui vérifie dès que l’on ajoute des données si le lien entre nous et la base de données en question est possible.

Bien que le nom soit fixe, il est possible de créer plusieurs \_\_construct. Ce qui va les différencier est le nombre de paramètres qu’ils auront.

Un construct est attaché à une classe. On peut donc avoir 1 construct avec le même nombre de paramètres par construct.

class Person3

{

public $nom;

public $prenom;

// Cette fonction se lancera dès que l'on utilisera le mot new.

function \_\_construct($prenom,$nom){

echo 'je suis entrain de naitre...';

$this->prenom = $prenom;

$this->nom=$nom;

}

function setNom($str){

$this->nom=$str;

}

function getNom(){

return $this->nom;

}

function setPrenom($str){

$this->prenom=$str;

}

function getPrenom(){

return $this->prenom;

}

function affiche(){

echo 'Je suis '.$this->prenom.' '.$this->nom.'<br>';

}

}

// Ici la fonction se lance et il suffit de mettre les paramètres (dans l'ordre de la fonction évidemment)

$alexD = new Person3('alex','Dupont');

$alexD->affiche();

$alexD->setNom('Delpot');

$alexD->setPrenom('marc');

L’héritage :

On peut « étendre » une classe déjà existante en en crééant une nouvelle qui va utiliser les données de la classe existante en ajoutant des nouvelles propriétés grâce au mot extends.

Plusieurs classes peuvent hériter d’une classe. Par contre, une classe enfant peut avoir un seul parent.

***Il faut être vigilant :*** C’est toujours la dernière valeur qui sera utilisée. Par exemple, si un parent a une méthode qui affiche en string classique alors que la fille héritière a une fonction qui affiche sous forme de tableau, alors la string sera écrasée et c’est l’affichage en tableau qui sera effectué.

// Le require sert à appeler un autre fichier PHP. Person3 est stockée sur un autre fichier.

require 'person3.php';

// Cette fonction signifie que la nouvelle classe Person5 vient compléter Person3 avec de nouveaux paramètres

// et de nouvelles méthodes.

class Person5 extends Person3

{

public $age;

function \_\_construct($p,$n,$a){

// parent:: est une super globale. Elle permet de faire appel à une fonction x de la classe parent.

// La fonction \_\_construct va demander à la fonction \_\_construct parent (qui a deux paramètres) de gérer

// les deux paramètres qui la concernent et par la suite elle viendra ajouter l'âge.

parent::\_\_construct($p,$n);

$this->age=$a;

}

function setAge($a){

$this->age=$a;

}

function getAge(){

return $this->age;

}

function affiche(){

// Ici, on demande d'utiliser la fonction affiche du parent qui fera la phrase avec le nom et le prénom

// et on vient y ajouter la fin de la phrase avec l'âge.

parent::affiche();

echo 'et j\'ai '.$this->age.'<br>';

}

}

$alexD = new Person5('alex','Dupont',25);

$alexD->affiche();

echo 'je vais changer d\'identité <br>';

$alexD->setNom('Delpot');

$alexD->setPrenom('marc');

$alexD->setAge(28);

$alexD->affiche();

Les différents types de propriétés :

Les propriétés publiques :

Elles sont accessibles par les classes enfants et tout le reste du programme.

public $nom;

Les propriétés privées (private) :

Elles ne sont accessibles qu’à la classe en question. Les enfants ne peuvent y accéder que si une fonction a été créée dans la classe parent pour les modifier.

private $prenom;

es propriétés protégées (protected) :

C’est un mélange des deux. Elles ont accessibles aux classes héritières mais sont protégées par un programme externe.

protected $prenom;

La fonction func\_get\_arg :

Il est possible d’avoir une fonction construct a trois arguments mais ne pas avoir les données pour tous les remplir.

Il existe une fonction qui permet de définir que si tel argument n’est pas ici, alors on donnera un paramètre par défaut :

class Maison

{

private $rue;

private $num;

private $ville;

public function \_\_construct()

{

// Cette fonction permet de savoir combien de paramètres ont été pacés dans la fonction.

$nbreargs = func\_num\_args();

// Le switch permet de gérer les cas qui nous intéressent

switch ($nbreargs) {

// On met case x: pour chaque cas que l'on veut traiter.

case 1:

// La fonction func\_get\_args est une fonction intégrée à PHP qui permet de cibler le paramètre d'une fonction.

// Les paramètres se comptent comme un tableau, le premier étant 0.

$this->mettreLesChamps(func\_get\_arg(0), '10', 'LIEVIN');

break;

case 2:

$this->mettreLesChamps(func\_get\_arg(0), func\_get\_arg(1), 'LIEVIN');

break;

case 3:

$this->mettreLesChamps(func\_get\_arg(0), func\_get\_arg(1), func\_get\_arg(2));

break;

default:

// On met ce qui ce passe si aucun des cas que l'on a prévu n'est atteint.

echo 'Erreur lors de l\'instanciation de l\'objet (' . $nbreargs . ' arguments): ';

$this->mettreLesChamps('INCONNU', -999, 'SANS PAYS');

//die();

break;

}

}

public function mettreLesChamps($rue, $num, $ville)

{

$this->rue = $rue;

$this->num = $num;

$this->ville = $ville;

}

public function afficher()

{

echo 'Rue : ' . $this->num . ', Numéro = ' . $this->num . ' et Ville :' . $this->ville . '<br>';

}

}

// On test en entrant un seul paramètre:

$x1 = new Maison('Pierre Legrand');

$x1->afficher();

// Avec deux:

$x2 = new Maison('Pierre Legrand', 25);

$x2->afficher();

// Avec 3:

$x3 = new Maison('Pierre Legrand', 50, 'TOULOUSE');

$x3->afficher();

// Avec 4 (le message d'erreur en default sera affiché)

$x4 = new Maison('Pierre Legrand', 50, 'TOULOUSE',"32 498");

$x4->afficher();

Les include :

Pour générer une page en PHP, la meilleure méthode est de faire une page index classique qui va appeler d’autres éléments PHP grâce au include.

A l’endroit où on veut faire apparaître le contenu dans l’index, on met ***include ‘nomdufichier.php’ ;***

include 'header.php';

Au refresh de la page, (ou dès le lancement selon ce que l’on veut) on va générer une page qui est l’addition de plusieurs morceaux de pages.

<!-- Cette page est la page index.php -->

<?php

// Ici, on a une variable title qui est dans le header.php (directement dans la balise titre HTML)

$title="titre test";

// On importe le fichier header.php qui contient le doctype, l'intégralité du head, la balise ouvrante body, celle du

// container bootstrap et le header. Comme le include vient après la variable title, le titre de l'index va s'adapter.

include "header.php";

// on génère une variable titre avec le titre d'article que l'on veut donner à cette page.

$titreContent ="Hello World!";

// Comme la variable a été générée avant l'include, le écho de la variable dans le H1 va lire Hello World.

// Le include contient uniquement le contenu.

include "content.php";

// Enfin, on importe le footer qui contient bien évidemment le footer, mais aussi la balise fermante du container

// Bootstrap, la balise fermante body et html.

include "footer.php";

?>

Créer un menu pour include grâce à un tableau :

<?php

// On créé une fonction navbar qui va générer les nav items contenus dans un tableau.

function navbar()

{

// On créé un tableau associatif qui va correspondre à chaque nav-item. Ici, chaque mot en clé correspond à un lien

// en valeur.

$tab = ["Accueil" => "index.php", "Page 1" => "index2.php", "Page 2" => "#", "Page 3" => "#", "About" => "#"];

foreach ($tab as $key => $value) {

// On fait un foreach qui va générer un li par élément dans le tableau. On affiche la value dans le href pour

// renvoyer au lien et la clé est entre les balises <a>. C'est ce que l'utilisateur verra en mot qui renverra au lien.

echo '<li class="nav-item">

<a class="nav-link" href="' . $value . '">' . $key . '</a>

</li>';

}

}

?>

<!-- On vient générer un header tout à fait classique, avec une navbar réalisée via Bootstrap. -->

<header >

<nav class="navbar navbar-expand-lg navbar-light bg-primary">

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav">

<!-- On a juste à appeler la fonction navvar() qui va créer un nombre de li équivalant au nombre

d'éléments contenus dans le tableau. -->

<?php navbar();?>

</ul>

</div>

</nav>

</header>

<!-- Quand on fera l'include, le header sera généré avec les li créés par la fonction navbar(). -->

Connecter PHP à SQL :

Il existe des fonctions intégrées à PHP :

Mysqli- (uniquement pour MySQL)

PDO (qui est un convertisseur qui fonctionne avec toutes les bases de données.

<?php

// Pour relier PHP à une base de données MYSQL, on utilise la fonction new mysqli.

// Il y a quatre paramètres:

// 1/ Le serveur (ici localhost)

// 2/ Le username (root puisqu'on travail en local)

// 3/ Le password (root, pour les mêmes raisons)

// 4/ Le nom de la base de données.

$mysqli = new mysqli("localhost", "root", "root", "dwwm\_antoine");

// Le connect\_error est déjà programmé par PHP. Il permet de voir le type d'erreur que l'on rencontre.

if($mysqli->connect\_error){

echo 'erreur de connection :'.$mysqli->connect\_error.'<br>';

}

else{

echo "connexion réussie";

}

// Traitement ...

// Il faut toujours fermer la DB à la fin.

$mysqli->close();

?>

En cas d’erreur, un warning apparaît. Si on veut éviter que l’utilisateur voit le chemin de notre base de donnée, il suffit de rajouter un @ devant n’importe quelle commande PHP :

$mysqli = @new mysqli("localhost", "root", "root", "dwwm\_antoine");

@$mysqli->close();

Une bonne chose à faire est de créer un fichier qui traite uniquement de la connexion à la db et un autre qui gère uniquement la fermeture.

On fait le traitement dans un fichier qui fera un require du début, le traitement et un require de la fermeture :

<?php

// Require suffit. require once signifie que si on a déjà require une fois alors les autres ne se feront pas.

require\_once('ouvertureDb.php');

// Traitement

require\_once("fermetureDb.php");

L’utilisation de la DB en elle-même :

<?php

require\_once('ouvertureDb.php');

// On créé une variable qui est la commande SQL que l'on veut utiliser.

$sql="select \* from note";

// La méthode query va se connecter au serveur et réaliser le résultat que l'on veut.

$result=$mysqli->query($sql); // S'il y a une erreur, le résultat sera false.

// On fait toujours un result free après un traitement pour libérer la mémoire de la place occupée par un résultat.

$result->free();

require\_once("fermetureDb.php");

Le premier query renvoie au tableau entier avant le premier enregistrement. (C’est un peu la position -1).

***Si on veut entrer dans le tableau et afficher les résultats il faut utiliser fetch\_array :***

require\_once('ouvertureDb.php');

$sql="select \* from note";

$result=$mysqli->query($sql); // S'il y a une erreur, le résultat sera false.

// Le while est une boucle qui dit que tant que le résultat est dans le tableau il le retourne dans la variable rec.

// De ce fait, chaque ligne du tableau est retournée dans la boucle.

while ($rec=$result->fetch\_array()){

echo 'nom: '.$rec['NOM'].' <br> prenom: '.$rec['PRENOM'].'<br> sexe: '.$rec['sexe'];

echo "<br>";

}

$result->free();

require\_once("fermetureDb.php");

A chaque requête, il faut recréer une variable avec une nouvelle requête, refaire un query pour interroger le tableau et ensuite faire un fetch array pour rentrer dans le tableau.

$sql="select \* from note where nom='Marchand'";

$result=$mysqli->query($sql); // S'il y a une erreur, le résultat sera false.

// On fait une requête qui retourne la personne qui a comme nom de famille "Marchand":

while ($rec=$result->fetch\_array()){

echo 'nom: '.$rec['NOM'].' <br> prenom: '.$rec['PRENOM'].'<br> sexe: '.$rec['sexe'];

echo "<br> <br>";

}

$result->free();

// On fait une deuxième requête qui va uniquement afficher les femmes:

$sql="select \* from note where sexe='F'";

$result=$mysqli->query($sql);

while ($rec=$result->fetch\_array()){

echo 'nom: '.$rec['NOM'].' <br> prenom: '.$rec['PRENOM'].'<br> sexe: '.$rec['sexe'];

echo "<br> <br>";

}

$result->free();

Mettre à jour une DB via PHP :

// La fonction update (ou delete) se note de cette manière:

$sql="update note set prenom ='Claude', password='MOTDEPASSE' where nom='Dupont'";

$result=$mysqli->query($sql);

// Quand on fait un update ou un delete, rien ne s'affiche. La requête génère un booléen qui dit true si

// ça a marché, sinon false.

// Pour vérifier:

if($result===true){

echo "Mise à jour réussie";

}

else{

echo "Erreur de mise à jour";

}

Sur un booléen, il est inutile d’utiliser la fonction $variable ->free() ;

La gestion d’un mot de passe hashé :

En matière de sécurité, il est obligatoire de ne pas entrer de mots de passe en clair dans une base de données. En cas de problème, si un utilisateur porte plainte, un mot de passe pas suffisamment protégé constitue un délit pénal.

Pour remédier à ce genre de problèmes, il existe une fonction toute simple qui permet de transformer un mot de passe entré en un code pratiquement indéchiffrable, y compris par le gestionnaire de la base.

<?php

// On se connecte à la DB:

$mysqli = new mysqli("localhost", "root", "root", "bd\_hash");

// On permet de s'enregistrer en hashant le MDP:

if (isset($\_POST["pseudo"]) && isset($\_POST["password"])){

$pseudo=$\_POST["pseudo"];

// Cette fonction a deux paramètres: le premier est le mot de passe que l'on veut entrer, le deuxième est soit

// PASSWORD\_DEFAULT qui va créer une chaîne de caractères qui varie.

//soit PASSWORD\_BCRYPT, qui va créer une chaîne hashée de 72 caractères maximum.

// Les deux sont fiables.

$password=password\_hash($\_POST["password"],PASSWORD\_BCRYPT);

// On va ajouter à la table la variable password qui sortira donc le mot entré par l'utilisateur en version hashée:

$ajoutUser='

insert into Personne (pseudo,password)

values

("'.$pseudo.'" ,"'.$password.'");

';

echo $ajoutUser."<br>";

// On utilise la requête:

$nouvelUser = $mysqli->query($ajoutUser);

}

?>

Une fois que l’on a pu créer un mot de passe hashé, il est très difficile pour le commun des mortels de l’extraire et de s’en servir. En théorie, l’utilisateur seul connaît son mot de passe. Lorsque celui-ci s’identifie, il faut donc comparer le mot de passe qu’il entre au mot de passé hashé dans la base de données.

Comparer un mot de passe à sa version hashée dans la base de données :

L’utilisateur inscrit va vouloir se reconnecter sur le site. Pour ce faire, il va s’identifier. Le mot de passe qu’il va entrer dans le formulaire ne sera pas hashé. Il faut donc pouvoir comparer le mot de passe non hashé à celui stocké dans la base.

Il existe, une fois encore, un moyen simple de le faire :

<?php

$mysqli = new mysqli("localhost", "root", "root", "bd\_hash");

if (isset($\_POST["pseudo"]) && isset($\_POST["password"])) {

$pseudo = $\_POST["pseudo"];

$password = $\_POST["password"];

// On vient sélectionner la ligne dans la DB où le pseudo est égal au pseudo entré par l'utilisateur:

$connexion = 'select \* from Personne where pseudo ="'. $pseudo.'";';

// Cette ligne sert à se connecter à la table, executer la requête et on en ressort un tableau:

$connectUser = $mysqli->query($connexion);

// Cette ligne sert à extraire une seule ligne de l'objet retourné (un enregistrement, ici la ligne du pseudo entré):

$row = $connectUser->fetch\_array();

// Cette ligne permet de vérifier si le mdp hashé dans la base de donnée correspond au hash du MDP saisi.

// password\_verify a deux paramètres. Le premier est le mot de passe à vérifier, le second est le mot de passe inscrit

// dans la base de données:

if (password\_verify($password,$row['password'])) {

header('Location:secret.php');

} else {

echo 'Echec de connexion.';

}

}

?>