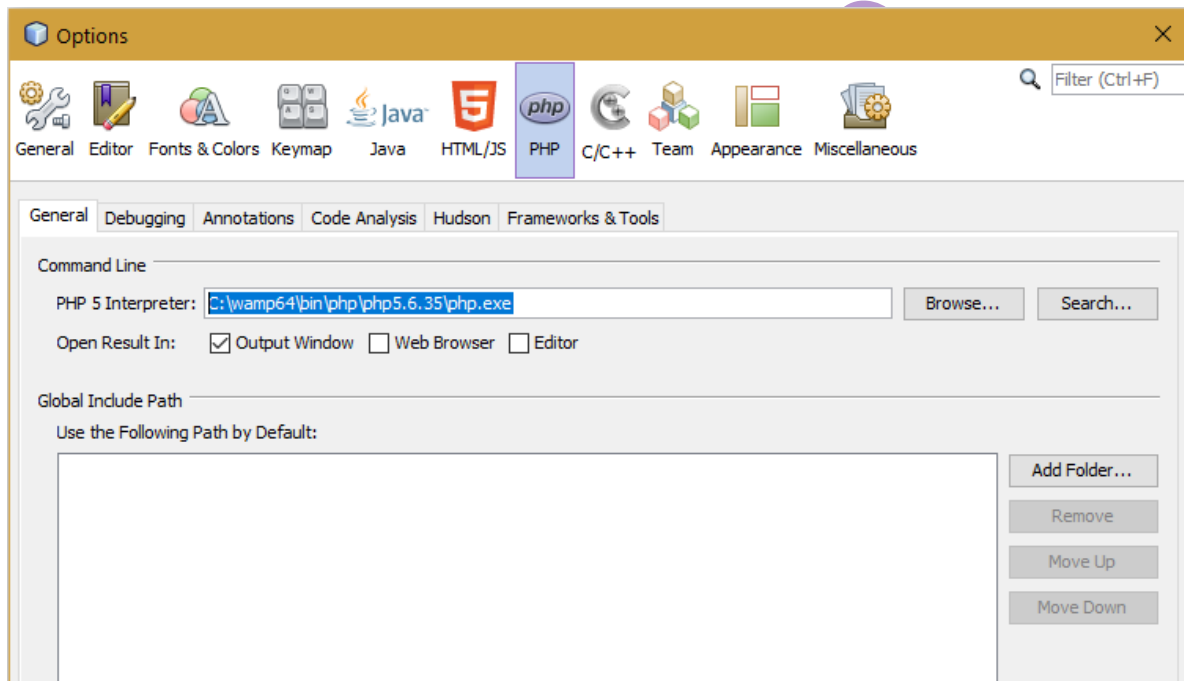


I. NOTIONS DE BASE SUR PHP

1. ENVIRONNEMENT DE TRAVAIL

Définition

- 1- Installer la pile WAMP. Site de téléchargement : <http://www.wampserver.com/>
- 2- Installer L'IDE NetBeans 8.2 pour PHP (peut nécessiter d'avoir une version 8+ du JDK).
 - L'IDE est téléchargeable sur cette page : <https://netbeans.org/downloads/index.html>
 - Choisir soit la version PHP ou la version complète.
 - Configurer l'interpréteur PHP dans NetBeans :
Pour pouvoir « compiler » les pages PHP, NetBeans a besoin d'un interpréteur PHP, la pile WAMP en contient déjà un, il faut donc spécifier son chemin à NetBeans.
Ouvrir NetBeans, et aller dans le menu Tools – Options, puis coller le chemin de l'outil php.exe qui existe dans le dossier bin de l'installation de WAMP.



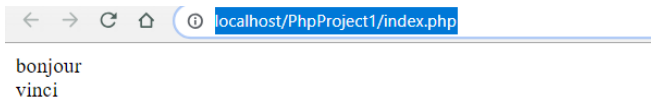
- 3- Créer et lancer un projet :
Dans la fenêtre de NetBeans, utiliser le raccourci CTRL+MAJ+N pour ouvrir la fenêtre de création d'un nouveau projet, puis choisir PHP puis PHP application. Nommer le projet puis valider.
Vous avez un projet avec un fichier index.php, taper le code suivant :

```
<html>
  <head>
    <meta charset="UTF-8">
  </head>
```

```
<body>
  <?php
    echo("bonjour<br>");
    echo("vinci");

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

Ouvrir l'outil WampServer pour que le serveur Apache soit démarré.
Ouvrir le navigateur et taper l'adresse de la page index.php :



The screenshot shows a web browser window. The address bar contains the URL `localhost/PhpProject1/index.php`. Below the address bar, the browser displays the output of the PHP code: `bonjour` followed by a line break and `vinci`.

- Solution : dans php on peut créer des pages indépendantes pour chaque partie de la page principale, et les inclure dans cette dernière, avec l'instruction `include("nom page")`

```
<body>
<div id="entete">
  <?php include("logo.php"); ?>
</div>

<div id="centre">
  <?php include("centre.php"); ?>
</div>

<div id="pied">
  <?php include("footer.php"); ?>
</div>
</body>
```

B. Les variables

Définitions

- Une variable est un espace mémoire qui contient une information, qu'on peut utiliser dans la page php (pour afficher des informations, faire des calculs, envoyer un mail, ...).
- Les types de variables en php :
 - bool : true / false
 - int : nombre entier signé
 - float : nombre réel
 - string : chaîne de caractères, toujours entourée de simples quotes '...' ou de doubles quotes "..."
 - NULL : contenu vide
- Déclaration & utilisation :

```
// declaration & initialisation
$nom_variable = valeur_initiale;

// écraser la valeur actuelle avec une nouvelle valeur
$nom_variable = nouvelle_valeur;
```

- Le nom de la variable commence toujours par \$ et ne peut pas contenir les caractères réservés aux opérations de php (., - + ; / ...).
- Les noms de variables sont « case-sensitive » en php, mais pas les noms de fonctions.

Exemple

```
<?php
$user_civilite = "M";
$user_nom = "ELYAHYAOUI";
echo("Bonjour<br>");
echo("Bienvenue ");
echo($user_civilite);
echo(".");
echo($user_nom);
?>
```

```
Bonjour
Bienvenue M.ELYAHYAOUI
```

C. Constantes / opérateurs / Tests / Boucles

Afficher des informations sur la page

```
ehco(...);
// ou bien
echo "...";
// ou bien
echo '...';
```

Concaténer des informations

La concaténation d'informations se fait avec l'opérateur « **point** », au contraire de beaucoup d'autres langages qui utilisent l'opérateur « + »

```
<?php
    $user_civilite = "M";
    $user_nom = "ELYAHYAUI";
    echo("Bonjour<br>". "Bienvenue ".$user_civilite." ".$user_nom);
?>
```

```
Bonjour
Bienvenue M.ELYAHYAUI
```

Les « constantes »

Les constantes sont définies avec la fonction `define` et ne peuvent pas être modifiées.

Exemple

```
<?php
    define("age", 50);
    echo 'votre age est : '.age;
?>
```

```
votre age est : 50
```

Opérateurs arithmétiques

```
=      opérateur d'affectation
==     égale : test d'égalité entre deux chiffres
!=     différent de : test d'inégalité entre deux chiffres
<      inférieur
>      supérieur
<=     inférieur ou égale
>=     supérieur ou égale
+      addition
*      multiplication
-      soustraction
/      division
%      modulo : reste de division euclidienne
```

Opérateurs arithmétiques d'incrément / décrémentation

+= (incrément)	x += a;	x = x + a;	<code>\$x = 1;</code> <code>\$x += 5; // la valeur de x est 6</code>
-= (décrément)	x -= a;	x = x - a;	<code>\$x = 9;</code> <code>\$x -= 5; // la valeur de x est 4</code>
++ (incrément par 1)	x++; ou bien ++x;	x = x + 1;	<code>\$x = 9;</code> <code>\$x++; // la valeur de x est 10</code> <code>++\$x; // la valeur de x est 11</code>

-- (décrémentation par 1)	x--; ou bien --x;	x = x - 1;	\$x = 9; \$x--; // la valeur de x est donc 8 --\$x; // la valeur de x est donc 7
---------------------------------	-------------------------	------------	--

Exemple

```
<?php
$age = 50;
echo '<ol>';

echo '<li>';
echo 'votre age est : '.$age;
echo '</li>';

echo '<li>';
echo 'votre age est : '.$age++;
echo '</li>';

echo '<li>';
echo 'votre age est : '.$age;
echo '</li>';

$age += 11;
echo '<li>';
echo 'votre age est : '.$age++;
echo '</li>';

$age /= 3;
echo '<li>';
echo 'votre age est : '.$age;
echo '</li>';

$age--;
echo '<li>';
echo 'votre age est : '.$age++;
echo '</li>';

echo '<li>';
echo 'votre age est : '.$age;
echo '</li>';

echo '</ol>';
?>
```

1. votre age est : 50
2. votre age est : 50
3. votre age est : 51
4. votre age est : 62
5. votre age est : 21
6. votre age est : 20
7. votre age est : 22

Opérateurs logiques

!

A	! A : non A
VRAI	FAUX
FAUX	VRAI

Le OU

| ou bien || ou bien or

A	B	A B : A ou B
VRAI	VRAI	VRAI
VRAI	FAUX	VRAI
FAUX	VRAI	VRAI
FAUX	FAUX	FAUX

Le ET

& ou bien && ou bien and

A	B	A && B : A et B
---	---	-----------------

	VRAI	VRAI	VRAI
	VRAI	FAUX	FAUX
	FAUX	VRAI	FAUX
	FAUX	FAUX	FAUX
le OU EXCLUSIF xor			
	A	B	A xor B : <i>A ou bien B</i>
	VRAI	VRAI	FAUX
	VRAI	FAUX	VRAI
	FAUX	VRAI	VRAI
	FAUX	FAUX	FAUX

Exemple

```
<?php
$b1 = TRUE; $b2 = TRUE;
$index = 1;

echo 'resultat '.$index++. ' : ' . ($b1 and $b2). '<br>';
echo 'resultat '.$index++. ' : ' . ($b1 or $b2). '<br>';
echo 'resultat '.$index++. ' : ' . ($b1 xor $b2). '<br>';
?>
```

```
resultat 1 : 1
resultat 2 : 1
resultat 3 :
```

Structures alternatives

▪ IF

```
if(condition) {
    // bloc 1
}
// bloc 2
```

- Le bloc 2 sera toujours exécuté.
- Le bloc 1 sera exécuté si la condition du IF donne **TRUE**.
- Les accolades { } du bloc 1 sont obligatoires si le bloc contient plusieurs instructions.

▪ IF – ELSE

```
if(condition) {
    // bloc 1
} else {
    // bloc 2
}
```

Le bloc 1 sera exécuté si la condition du IF donne **TRUE**, sinon c'est le bloc 2 qui sera exécuté.

▪ IF – ELSEIF – ELSE

```
if(condition1) {
    // bloc 1
} elseif(condition2) {
    // bloc 2
} elseif(condition3) {
    // bloc 3
} .....
```

Le dernier bloc **else** sera exécuté si toutes les conditions donnent **FALSE**.

```
else {  
    // bloc N  
}
```

▪ SWITCH

```
switch($une_variable) {  
    case valeur1 :  
        // bloc 1  
        break;  
    case valeur2 :  
        // bloc 2  
        break;  
    case valeur3 :  
        // bloc 3  
        break;  
    case .....  
    default :  
        // bloc par défaut  
}
```

Le dernier bloc **default** sera exécuté si la valeur de la variable du **switch** ne correspond à aucune des valeurs testées dans les blocs **case**.

Exemples

```
<?php  
$b1 = TRUE; $b2 = FALSE;  
if($b1 and $b2) {  
    echo 'bon weekend';  
}  
echo '<br>';  
echo 'bon debut de semaine';  
?>
```

bon debut de semaine


```
<?php
/*
 * AFFICHER LE NOM DU JOUR EN LETTRES
 * EN FONCTION DU JOUR EN CHIFFRES
 */
$jour = -2;
$jour_lettres = '';

if($jour == 1) {
    $jour_lettres = 'Lundi';
} elseif ($jour == 2) {
    $jour_lettres = 'Mardi';
} elseif ($jour == 3) {
    $jour_lettres = 'Mercredi';
} elseif ($jour == 4) {
    $jour_lettres = 'Jeudi';
} elseif ($jour == 5) {
    $jour_lettres = 'Vendredi';
} elseif ($jour == 6) {
    $jour_lettres = 'Samedi';
} elseif ($jour == 7) {
    $jour_lettres = 'Dimanche';
} else {
    $jour_lettres = 'VALEUR INCORRECTE!';
}
echo $jour_lettres;
?>
```

VALEUR INCORRECTE!

```
<?php
/*
 * AFFICHER LE NOM DU JOUR EN LETTRES
 * EN FONCTION DU JOUR EN CHIFFRES
 * EN UTILISANT LA STRUCTURE SWITCH/CASE
 */
$jour = 6;
$jour_lettres = '';

switch ($jour) {
    case 1 :
        $jour_lettres = 'Lundi';
        break;
    case 2 :
        $jour_lettres = 'Mardi';
        break;
    case 3 :
        $jour_lettres = 'Mercredi';
        break;
    case 4 :
        $jour_lettres = 'Jeudi';
        break;
    case 5 :
        $jour_lettres = 'Vendredi';
        break;
    case 6 :
        $jour_lettres = 'Samedi';
        break;
    case 7 :
        $jour_lettres = 'Dimanche';
        break;
    default :
        $jour_lettres = 'VALEUR INCORRECTE!';
}
echo $jour_lettres;
?>
```

Samedi



REMARQUES

- Les blocs « case » qui ont le même traitement peuvent être regroupés :

```
switch($une_variable) {
```

```
    case valeur1 :
```

```
    case valeur2 :
```

Le bloc 2 sera exécuté pour les valeurs valeur1 et valeur2.

```
// bloc 2
break;
case valeur3 :
    // bloc 3
    break;
case .....

default :
    // bloc par défaut
}
```

▪ Même si ce n'est pas une erreur de compilation, il ne faut pas oublier l'instruction break à la fin de chaque bloc case, pour sortir du bloc switch sans passer vers les blocs case qui suivent :

```
<?php
/*...5 lines */
$jour = 1;
$jour_lettres = '';

switch ($jour) {
    case 1 :
        $jour_lettres = 'Lundi';
        echo 'test 1<br>';
    case 2 :
        $jour_lettres = 'Mardi';
        echo 'test 2<br>';
    case 3 :
        $jour_lettres = 'Mercredi';
        break;
    case 4 :
        $jour_lettres = 'Jeudi';
        break;
    case 5 :
        $jour_lettres = 'Vendredi';
        break;
    case 6 :
        $jour_lettres = 'Samedi';
        break;
    case 7 :
        $jour_lettres = 'Dimanche';
        break;
    default :
        $jour_lettres = 'VALEUR INCORRECTE!';
}
echo $jour_lettres;
?>
```

```
test 1
test 2
Mercredi
```

Structures itératives

■ FOR

Valeur initiale du compteur : 1

```
<?php
$a = 8;
for ($i = 1; $i <= 10; $i += 1) {
    echo ($a * $i);
    if ($i != 10) {
        echo ' - ';
    }
}
?>
```

Pat de chaque itération : +1

Condition pour continuer la boucle : $i \leq 10$

8 - 16 - 24 - 32 - 40 - 48 - 56 - 64 - 72 - 80

■ WHILE

```
<?php
$a = 8;
$i = 1;
while($i <= 10) {
    echo ($a * $i);
    if ($i != 10) {
        echo ' - ';
    }
    $i++;
}
?>
```

Ne jamais oublier l'incrémentation / décrémentation dans les boucles WHILE et DO-WHILE

■ DO-WHILE

```
<?php
$a = 8;
$i = 1;
do {
    echo ($a * $i);
    if ($i != 10) {
        echo ' - ';
    }
    $i++;
} while($i <= 10);
?>
```

- Uniquement dans la boucle do-while, la vérification de la condition d'itération se fait après la 1ère itération.
- Le point-virgule après while(...) est obligatoire.

■ FOREACH

```
<?php
$a = 8;
$tableau = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
foreach ($tableau as $value) {
    echo ($a * $value);
    if ($value != 10) {
        echo ' - ';
    }
}
?>
```

- La boucle foreach est utilisée uniquement avec les tableaux.
- Cette boucle ne contient ni condition d'itération ni incrémentation / décrémentation.

■ L'INSTRUCTION « BREAK »

```
<?php
$a = 8;
$tableau = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
foreach ($tableau as $value) {
    echo ($a * $value);
    if ($value != 5) {
        echo ' - ';
    } else {
        break;
    }
}
?>
```

L'instruction **break** sert à stopper une boucle même si elle contient encore des itérations.

8 - 16 - 24 - 32 - 40

■ L'INSTRUCTION « CONTINUE »

```
<?php
$a = 8;
$tableau = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
foreach ($tableau as $value) {
    if($value % 2 == 0) {
        continue;
    }
    echo ($a * $value);
    if($value != 10) {
        echo ' - ';
    }
}
?>
```

L'instruction **continue** sert à sauter une itération.

8 - 24 - 40 - 56 - 72 -

Exercice 1

Initialiser une variable avec la valeur d'un mois (1 .. 12). Puis afficher selon cette valeur le nombre de jours du mois (30 / 31 / 28 ou 29).

Exercice 2

Initialiser une variable avec une valeur N. Puis afficher les entiers de 1 à N 5 par 5 comme ceci :

Exemple N = 22

1 - 2 - 3 - 4 - 5

6 - 7 ... 10

... 15

... 20

21 - 22

D. Les chaînes de caractères

Affichage

- Afficher du texte

```
echo(...);  
ou bien  
echo ...;
```

- Différence entre echo "..." et echo '...'

```
<?php  
$nom = "ELYAHYAOUI";  
echo 'bonjour M. $nom';  
echo '<br>';  
echo "bonjour M. $nom";  
?>
```

```
bonjour M. $nom  
bonjour M. ELYAHYAOUI
```

- Longueur et positionnement

```
<?php  
$user = "vinci";  
$message = "bonjour";  
  
echo strlen("$message $user");  
echo '<br>';  
echo strpos("$message $user", "b");  
echo '<br>';  
echo strpos("$message $user", $user);  
?>
```

```
13  
0  
8
```

- Nombre de mots

```
str_word_count(chaine)
```

- Inverser une chaîne

```
strrev(chaine)
```

- Remplacer une sous-chaîne

```
str_replace(cible, remplacement, chaine);
```

- Echappement des caractères spéciaux

```
<?php  
echo "\\\" \\\\";  
?>  
  
" \
```

Exemple

```
<?php  
  
$text = "Bonjour tout le monde!";  
echo 'Nombre de mots : '.str_word_count($text).'\<br>';  
echo 'Chaîne à l\'envers : '.strrev($text).'\<br>';  
echo 'Chaîne remplacée : '.str_replace('Bonjour', 'Hello', $text).'\<br>';  
?>
```

Nombre de mots : 4
Chaine à l'envers : !ednom el tuot ruojnoB
Chaine remplacée : Hello tout le monde!

E. Les tableaux

Les tableaux indexés

- Déclarer un tableau

```
$tab = array(); // la taille n'est pas spécifiée
```

Ou bien

```
$tab = [];
```

- Déclarer et Remplir un tableau

```
$tab = array(element1, element2, ...);
```

Ou bien

```
$tab = [element1, element2, ...];
```

- Afficher un tableau tout entier

```
print_r($tab);
```

- Longueur du tableau

```
count($tab);
```

- Récupérer un élément

```
$tab[$i] // i : 0 .. taille-1
```

- Parcourir un tableau

```
foreach($tab as $element) { ... }
```

Ou bien

```
for($i = 0; $i < count($tab); $i++) { ... }
```

- Ajouter des éléments

// ajouter au début

```
array_unshift($tab, elem1, elem2, ...);
```

// ajouter à la fin

```
array_push($tab, elem1, elem2, ...);
```

- Remplacer des éléments

```
$tab[indice] = nouvelle_valeur
```

- Supprimer des éléments

```
array_splice($tab, indice, nbr);
```

// supprime "nbr" éléments, à commencer par l'élément se trouvant à la position "indice"

Exemples

```
<?php
$tab = array("test0", "test1", "test2", "test3", "test4");

array_push($tab, "test5");
print_r($tab);
echo '<br><br>';

array_unshift($tab, "test6");
print_r($tab);
echo '<br><br>';

array_splice($tab, 2, 2);
$nbr = count($tab);
for($i = 0; $i < $nbr; $i++) {
    echo $tab[$i];
    if($i < $nbr - 1) {
        echo " - ";
    }
}

?>
```

Array ([0] => test0 [1] => test1 [2] => test2 [3] => test3 [4] => test4 [5] => test5)

Array ([0] => test6 [1] => test0 [2] => test1 [3] => test2 [4] => test3 [5] => test4 [6] => test5)

test6 - test0 - test3 - test4 - test5

Exercice

Remplir un tableau de chaines de caractères, puis parcourir ce tableau pour afficher la chaine la plus longue.

Les tableaux associatifs

```
// tableaux qui contiennent des associations clef-valeur
```

```
$tab = array(clef1 => element1, clef2 => element2, ...);
```

- Récupérer un élément

```
$tab[clef] // on ne peut pas utiliser l'indice
```

- Récupérer la liste des clefs

```
$clefs = array_keys($tab);
```

Exemple

```
<?php
$tab = array(
    'etudiant1' => "CHRISTOPHER CHANCE",
    'etudiant2' => "ILSA POUCCHI"
);

$tab['etudiant3'] = "WINSTON LAVERNE";

foreach ($tab as $value) {
    echo $value."<br>";
}
?>
```

```
CHRISTOPHER CHANCE
ILSA POUCCHI
WINSTON LAVERNE
```

Les tableaux multi-dimensions

Exemple : tableau 2 dim

```
$tab = array(clef1 => array(...), clef2 => array(...), ...);
```

Exemple

```
<?php
$tab = array(
    'CHRISTOPHER' => array('Math' => 17, 'Physique' => 18),
    'ILSA' => array('Math' => 12, 'Physique' => 12),
    'WINSTON' => array('Math' => 15, 'Physique' => 14)
);

foreach ($tab as $value) {
    print_r($value);
    echo "<br>";
}
?>
```

```
Array ( [Math] => 17 [Physique] => 18 )
Array ( [Math] => 12 [Physique] => 12 )
Array ( [Math] => 15 [Physique] => 14 )
```


F. Lire / écrire dans des fichiers du disque dur

Introduction

Étapes :

- Ouvrir le fichier : fonction `fopen(nom_fichier, mode_ouverture)`
- Récupérer la taille du fichier : fonction `fsize()` (en cas de lecture)
- Lire depuis le fichier : fonction `fread()` / Écrire sur le fichier : fonction `fwrite()`
- Fermer le fichier : fonction `fclose()`

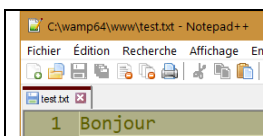
Les modes d'ouverture :

- **"r" : lecture seule** :
Ouvrir le fichier en lecture seule.
Le fichier doit être créé au préalable.
- **"w" : écriture seule** :
Ouvrir le fichier en écriture seule.
Le nouveau contenu écrasera l'ancien contenu.
Si le fichier n'existe pas, il sera créé.
- **"a" : mode d'ajout** :
Ouvrir le fichier en écriture seule.
Le nouveau contenu sera ajouté à la fin du contenu existant.
Si le fichier n'existe pas, il sera créé.
- **"r+" : lecture et écriture** :
Ouvrir le fichier en lecture & écriture.
Le fichier doit être créé au préalable.
L'écriture du nouveau contenu commence au début du fichier.
- **"w+" : lecture et écriture** :
Ouvrir le fichier en lecture & écriture.
Le nouveau contenu écrasera l'ancien contenu.
Si le fichier n'existe pas, il sera créé.
- **"a+" : ajout en lecture / écriture à la fin** :
Ouvrir le fichier en lecture & écriture.
Le nouveau contenu sera ajouté à la fin du contenu existant.
Si le fichier n'existe pas, il sera créé.

Exemple : Ecrire sur un fichier en le créant pour la 1ère fois

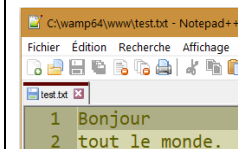
```
<?php
$filename = "C:/wamp64/www/test.txt";
$file = fopen($filename, "w");
if ($file == false) {
    echo ( "Opening file error!" );
} else {
    fwrite($file, "Bonjour");
    fclose($file);
    echo "Writing file done.";
}
?>
```

Writing file done.



Exemple : Ajouter du texte à un fichier existant

```
<?php
$filename = "C:/wamp64/www/test.txt";
$file = fopen($filename, "a");
if ($file == false) {
    echo ( "Opening file error!" );
} else {
    fwrite($file, "\ntout le monde.");
    fclose($file);
    echo "Writing file done.";
}
?>
```



Exemple : Lire depuis un fichier existant

```
<?php
$filename = "C:/wamp64/www/test.txt";
$file = fopen($filename, "r");
if ($file == false) {
    echo ( "Opening file error!" );
} else {
    $filesize = filesize($filename);
    $filetext = fread($file, $filesize);
    fclose($file);
    echo "<pre>$filetext</pre>";
}
?>
```

Bonjour
tout le monde.

Sérialisation / désérialisation

Sérialisation : fonctions `file_put_contents(...)` + `serialize(...)`

Action de stocker un objet ou tableau d'objets sur un fichier.

Désérialisation : fonctions `file_get_contents(...)` + `unserialize(...)`

Action inverse – récupérer le contenu d'un fichier dans un objet ou un tableau.

Exemple 1 : Sérialiser un tableau dans un fichier

```
<?php
$stab = array(
    'CHRISTOPHER' => array('Math' => 17, 'Physique' => 18),
    'ILSA' => array('Math' => 12, 'Physique' => 12),
    'WINSTON' => array('Math' => 15, 'Physique' => 14),
    'GUERRERO' => array('Math' => 0, 'Physique' => 0)
);

$string_data = serialize($stab);
file_put_contents("C:/wamp64/www/test.txt", $string_data);
echo 'done';
?>
```

Contenu écrit sur le fichier :

```
a:4:{s:11:"CHRISTOPHER";a:2:{s:4:"Math";i:17;s:8:"Physique";i:18;}}s:4:"ILSA";
a:2:{s:4:"Math";i:12;s:8:"Physique";i:12;}}s:7:"WINSTON";a:2:{s:4:"Math";i:15;s:8:"Physique";i:14;}}s:8:"GUERRERO";
a:2:{s:4:"Math";i:0;s:8:"Physique";i:0;}}
```

Exemple 2 : Désérialiser un fichier vers un tableau

```
<?php
$filename = "C:/wamp64/www/test.txt";
$string_data = file_get_contents($filename);
$tab = unserialize($string_data);
foreach ($tab as $value) {
    echo "Note en math : ".$value["Math"]."<br>";
}
?>
```

CHRISTOPHER - Note en math : 17
ILSA - Note en math : 12
WINSTON - Note en math : 15
GUERRERO - Note en math : 0

Exercice : Gestion des clients

Créer une application qui propose les actions suivantes :

Ajouter / modifier / supprimer un client, afficher la liste des clients.

La liste des clients doit être lue depuis un fichier et stockée dans un tableau au lancement de l'application.

Optionnel : Après tous les changements sur la liste, elle doit être sérialisée sur le fichier.

G. Notions de base sur le WEB

INTERNET

Internet est un réseau : ensemble de machines, de câbles, de routeurs, ...qui fait circuler des informations partout dans le monde.

WEB

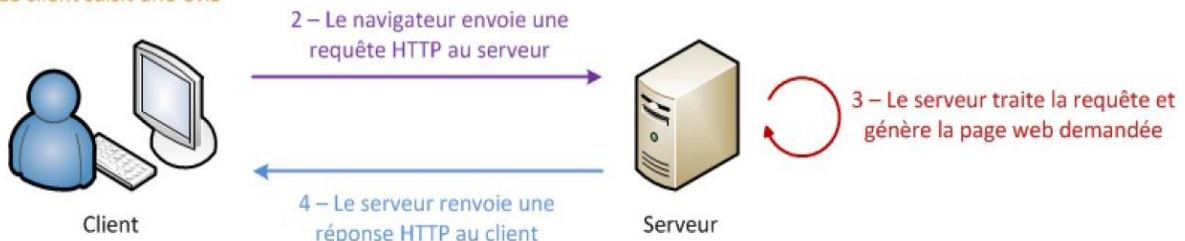
La navigation sur site Web est l'un des services accessibles par internet, parmi d'autres : transfert de fichiers, téléphonie IP, messagerie, ...

La navigation sur site Web est une communication entre deux parties :

- Client : généralement le browser.
- Serveur : machine distante sur laquelle se trouve toutes les ressources du site web (pages web, médias, fichiers, base de données, ...)

HTTP : protocole d'échange entre le client et le serveur

1 - Le client saisit une URL



HTML / CSS / Javascript

La page web générée par le serveur contient des données.

- Le format d'échange de ces données est le langage **HTML**.
- Le langage **CSS** quant à lui sert à la décoration & l'organisation du contenu des pages web.
- Les traitements qui doivent être effectués sur ces données du côté du navigateur sont écrits en **Javascript**.

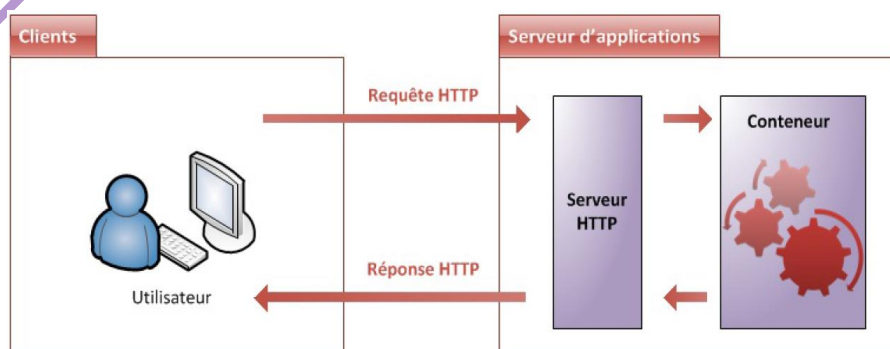
Page web statique / dynamique

- Page web statique : générée uniquement à partir des langages HTML, CSS et Javascript. Ce sont les seuls langages que peut exécuter le client (le navigateur).
- Page web dynamique : générée à partir d'un langage de programmation de sites web dynamiques. Ce langage est exécuté uniquement par le serveur. (PHP, JAVA, C#, ...)

La machine serveur doit donc réaliser les tâches suivantes :

- Traiter les requêtes http. Cette tâche appartient à un programme qu'on appelle « serveur http ». (Généralement Apache HTTP server ou Microsoft IIS)
- Compiler le code écrit en langage PHP, JAVA, ... Cette tâche appartient à un programme qu'on appelle « un conteneur ».

Le « serveur http » et « conteneur » forment ce qu'on appelle le « **serveur d'applications** ».



Envoyer des informations au serveur

Une page a besoin d'envoyer des informations au serveur.

Il y a 2 manières pour le faire :

Soit elle envoie ses infos dans l'URL de la requête http : c'est la méthode **GET**,

Soit elle les envoie dans un formulaire : c'est la méthode **POST**

La méthode GET

▪ La requête du client

1. Demander une ressource (généralement une page) via l'URL ou via un lien (*href*).
2. Possibilité de transmettre au serveur, des paramètres appelés « query strings », directement dans l'URL.
3. La taille des infos envoyées au serveur est limitée.
4. Ne vise - selon les normes du HTTP - que la lecture de données.

▪ La réponse du serveur

1. Renvoie la ressource demandée.
2. Renvoie aussi des informations comme la longueur des données renvoyées, ou la date d'envoi, placées dans l'en-tête de la réponse.

La méthode POST

▪ La requête du client

1. On peut envoyer des données plus volumineuses au serveur, ex. données d'un formulaire, des fichiers, etc.
2. La taille des infos envoyées au serveur n'est pas limitée.
3. Peut viser des modifications sur les données, donc le fait de renvoyer la même requête POST au serveur, est soumis à l'approbation de l'utilisateur.

▪ La réponse du serveur

1. De même que la méthode GET.

\$ _GET et \$ _POST

Quand une page p1 cible une autre page p2, avec la méthode GET ou POST, on peut récupérer les paramètres envoyés dans la requête par deux tableaux associatifs prédéfinis en php qui sont **\$ _GET** et **\$ _POST**

```
// en cas de requête GET
$variable = $_GET["nom_parametre"]

// en cas de requête POST
$variable = $_POST["nom_parametre"]
```

Exemple 1 : GET

▪ 1ère manière d'envoyer des paramètres en méthode GET :
Afficher un lien **href** avec la fonction echo.

Page index1.php

```
<?php
    $nom = "ELYAHYAUI";
    $profession = "ENSEIGNANT";

    /* AFFICHER UN LIEN HTML VERS UNE AUTRE PAGE, AVEC
    DES PARAMETRES DANS LA QUERY STRING */
    echo "<a href='\"index2.php?nom=$nom&pro=$profession\"'>Afficher</a>";
?>
```

localhost/PhpProject1/get_post/ x +
localhost/PhpProject1/get_post/index1.php
[Afficher](#)

localhost/PhpProject1/get_post/index2.php?nom=ELYAHYAOU&pro=ENSEIGNANT

Page index2.php

```
<?php
$user_name = $_GET["nom"];
$user_pro = $_GET["pro"];
if ($user_name && $user_pro) {
    echo "Bonjour M. $user_name,<br>Votre profession est : $user_pro";
} else {
    echo "Vous devez specifier votre nom et votre profession.";
}
?>
```

localhost/PhpProject1/get_post/ x +
localhost/PhpProject1/get_post/index2.php?nom=ELYAHYAOU&pro=ENSEIGNANT

Bonjour M. ELYAHYAOU,
Votre profession est : ENSEIGNANT

2ieme manière d'envoyer des paramètres en méthode GET :
Taper les paramètres directement dans l'URL de la page :
http://adresse?param1=valeur1¶m1=valeur2&...

Exemple 2 : POST

Page index1.php

```
<body>
<form action="index2.php" method="POST">
    <input type="text" name="nom" placeholder="taper le nom">
    <br><br>
    <input type="text" name="pro" placeholder="taper la profession">
    <br><br>
    <button type="submit">ENVOYER</button>
</form>
</body>
```

taper le nom

taper la profession

ENVOYER

Page index2.php

```
<?php
$user_name = $_POST["nom"];
$user_pro = $_POST["pro"];
if ($user_name && $user_pro) {
    echo "Bonjour M. $user_name,<br>Votre profession est : $user_pro";
} else {
    echo "Vous devez specifier votre nom et votre profession.";
}
?>
```

localhost/PhpProject1/get_post/index2.php

Bonjour M. ELYAHYAOU,
Votre profession est : ENSEIGNANT

TP : Gestion des clients v1.1**Partie 1 :**

Utiliser le même fichier de données de la version précédente.

Créer une page ajoutClient.php qui doit contenir un formulaire d'ajout de client.

Le page ajoutClient.php doit envoyer ses données vers une autre page listeClients.php, qui se chargera de l'ajout du client, puis de l'affichage de la liste.

Lors de l'ajout du client, on doit l'ajouter dans le tableau, sérialiser le tableau, puis afficher la liste des clients.

Les informations des clients ne doivent pas être envoyées dans l'URL.

La page listeClients.php qui sera la page d'accueil de l'application, doit charger le tableau de clients depuis le fichier de sérialisation.

Partie 2 :

Créer une autre page editClient.php, qui contient un formulaire de mise à jour d'un client, qui doit elle aussi envoyer ses données vers la page listeClients.php.

Quel-est le problème que vous rencontrez ? Proposez une solution.

Indication : utiliser les « input » de type « hidden »

Inclure un fichier PHP dans un autre

▪ La fonction include

1. Quand un fichier A.php inclue un autre B.php avec la fonction **include**, toutes les variables, structures de données (ex. tableaux, classes, ...), fonctions, etc, sont utilisables dans A.php.
2. Syntaxe :

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

▪ La fonction require

1. Même effet que la fonction **include**.
2. Au cas où le fichier ciblé n'existe pas, la fonction **require** retourne une erreur fatale, ce qui arrête l'exécution du script php, tandis que la fonction **include** ne retourne qu'un warning, le script continue donc son exécution, ce qui peut causer des erreurs ou des incohérences de données ou de traitements.

▪ Les fonctions require_once et include_once

1. Même effet que **require** et **include**, mais elles permettent d'empêcher que le fichier soit importé plusieurs fois.

Exemple

Dans un dossier nommé include_require, créer **un fichier** serialize_array.php et **une page web** index.php.

Le fichier **serialize_array.php** contient :

- Une constante nommée **filename**, qui contiendra le nom complet du fichier de sérialisation,
- Une fonction **serialiser(\$tableau)** qui doit sérialiser le tableau passé en paramètre dans le fichier.
- Une fonction **deserialiser()** qui doit lire le tableau depuis le fichier et le retourner.

```
<?php
define("filename", "C:/wamp64/www/test.txt");

function serialiser($tableau) {
    $string_data = serialize(filename, $tableau);
    file_put_contents(filename, $string_data);
}

function deserialiser() {
    $string_data = file_get_contents(filename);

    $tableau = unserialize($string_data);
    return $tableau;
}
```

La page **index.php** contient :

- Importer le fichier **serialize_array.php**,
- Faire l'appel de la fonction **deserialiser()** et récupérer le tableau qu'elle retourne,
- Parcourir et afficher les éléments du tableau.

```
<?php
require_once "serialize_array.php";

$tab = deserialiser();
$noms = array_keys($tab);
$i = 0;
foreach ($tab as $value) {
    echo $noms[$i++]. " - Note en math : ".$value["Math"]. "<br>";
}
?>
```


CHRISTOPHER - Note en math : 17
ILSA - Note en math : 12
WINSTON - Note en math : 15
GUERRERO - Note en math : 0



REMARQUE

L'utilisation des fonctions sera détaillée dans le chapitre de la POO.

Les redirections

▪ La fonction header

Chaque requête du browser demande une page spécifique, cette page peut effectuer une redirection de la requête vers une autre page (dans le même site ou dans un autre) avec la fonction **header**

```
<?php
// traitements (si nécessaire)
header("location: adresse");
?>
```

Pour éviter l'interprétation du code qui vient après l'appel de la fonction **header**, on peut appeler la fonction **exit**.

Exemple :

```
<?php
// traitements
if(conditions) {
    header("location: adresse");
    exit();
} else {
    // suite des traitements
}
?>
```

TP : Gestion des clients v1.2

- Créer un nouveau fichier **crud.php** qui contiendra les fonctions ajouter, récupérer, modifier et supprimer client.
- Créer un autre fichier **serialiser.php** qui contiendra les fonctions de sérialisation et de désérialisation.
- Les autres pages php vont devoir désormais utiliser les fonctions de ces 2 fichiers.
- La page d'accueil doit être listeClients.php (configurer cela dans les propriétés du projet NetBeans), mais cette page doit rediriger vers la page ajoutClient.php si la liste des clients est vide, avec affichage du message « la liste est vide, vous devez ajouter des clients ».

Interface : Liste des clients

NOM	ADRESSE	TEL	OPTIONS	
ECOLE SUPÉRIEURE VINCI	RABAT VILLE	05.37.37.37.38	<input type="button" value="editer"/>	<input type="button" value="supprimer"/>
ECOLE VINCI	RABAT VILLE	0537373737	<input type="button" value="editer"/>	<input type="button" value="supprimer"/>
ELYAHYAOU	KENITRA	0665656565	<input type="button" value="editer"/>	<input type="button" value="supprimer"/>

[Ajouter](#)

Interface : Ajout de client (cas d'une liste vide)

la liste est vide, vous devez ajouter des clients

NOM

ADRESSE

TEL

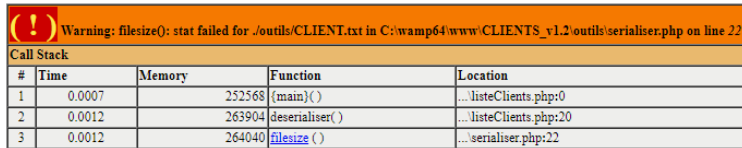
- Les pages editClient.php et ajoutClient.php doivent cibler la page listeClients.php dans leurs formulaires.
- Le formulaire « supprimer » de la page listeClients.php doit cibler la même page.

La gestion des erreurs

▪ La fonction die()

1. Certains traitements peuvent causer des erreurs qui peuvent interrompre l'exécution d'une page.

Ex. : Effectuer une opération sur un fichier qui n'existe pas.



Warning: filesize(): stat failed for ./outils/CLIENT.txt in C:/wamp64/www/CLIENTS_v1.2/outils/serialiser.php on line 22

#	Time	Memory	Function	Location
1	0.0007	252568	{main}()	...listeClients.php:0
2	0.0012	263904	deserialiser()	...listeClients.php:20
3	0.0012	264040	filesize()	...serialiser.php:22

2. Pour éviter que de tels messages d'erreurs apparaissent dans les pages, et traiter les éventuelles erreurs de manière efficace, on utilise la fonction **die(message)**. Cette fonction arrête l'exécution du code php en affichant le message spécifié en paramètre.

```
if(test de l'éventuelle erreur) {
    die("message à afficher");
} else {
    // traitement à effectuer si aucune erreur
}
```

Exemple

```
<?php
$filename = "C:/wamp64/www/TEST.txt";
if (!file_exists($filename)) {
    die("Opening file error!");
} else {
    $file = fopen($filename, "r");
    $filesize = filesize($filename);
    $filetext = fread($file, $filesize);
    fclose($file);
    echo "<pre>$filetext</pre>";
}
?>
```

Cas normal :

```
a:2:{s:11:"CHRISTOPHER";a:2:{s:4:"Math";i:17;s:8:"Physique";i:18;}}s:4:"ILSA";a:2:{s:4:"Math";i:12;s:8:"Physique";i:12;}}
```

Cas d'erreur :

Opening file error!

▪ Le bloc TRY-CATCH

1. Les Exceptions :

La gestion des erreurs est plus efficace en utilisant les « **exceptions** ». Par définition, chaque erreur qui survient **lors de l'exécution**, et **non lors de la vérification de la syntaxe**, est une exception.

Tout comme les autres langages, php aussi offre un mécanisme d'interception des exceptions, mais celui-ci est différent, car **on doit provoquer explicitement l'exception pour pouvoir la traiter**.

2. Le bloc TRY-CATCH :

C'est la structure qui permet de traiter l'exception

- 1- tester un code,
- 2- en cas de problème, soulever une exception avec l'instruction **throw**,
- 3- appel de la fonction **exit()** pour interrompre l'exécution du code,

4- dans le bloc **catch**, définir le code de remplacement si l'exception est soulevée.

```
try {
    instructions
    if(test de l'éventuelle erreur) {
        throw new Exception("message de l'exception");
        exit();
    }
}
```

```
} catch(Exception ex) {
```

Traitements à effectuer si l'exception est levée. Par ex. : rediriger vers une page d'erreur, qui va afficher la cause de l'erreur.

L'exception est passée en paramètre à l'opérateur catch(...). C'est une variable de type « Exception ». Chaque exception possède des fonctions qui renseignent sur l'erreur qui s'est produite :

```
ex->getMessage();    donne la cause de l'exception
ex->getFile();        donne le nom du fichier php où l'exception a été levée
ex->getLine();        donne le numéro de la ligne où l'exception a été levée
etc.
```

```
}
```

Exemple

index.php

```
<?php
$filename = "C:/wamp64/www/TEST.txt";
try {
    if (!file_exists($filename)) {
        throw new Exception('Le fichier n'existe pas.');
```

if(chaine) retourne **true** si la chaine de caractères n'est pas vide.

```
        exit();
    }
    $file = fopen($filename, "r");
    $filesize = filesize($filename);
    $filetext = fread($file, $filesize);
    if (!$filetext) {
        throw new Exception('Le fichier est vide.');
```

```
        exit();
    }
    fclose($file);
    echo "<pre>$filetext</pre>";
} catch (Exception $exc) {
    $error_msg = $exc->getMessage();
    header("location: error_page.php?msg=$error_msg");
}
```

error_page.php

```

<?php
    $text = $_GET["msg"];
?>
<h3>
    une erreur s'est produite!
</h3>
<table cellpadding="10">
    <tbody>
        <tr>
            <td>Cause de l'erreur : </td>
            <td>
                <?php
                    echo "$text";
                ?>
            </td>
        </tr>
    </tbody>
</table>
<br>
<a href="index.php">Retourner à l'accueil.</a>

```

Cas : fichier inexistant

une erreur s'est produite!

Cause de l'erreur : Le fichier n'existe pas.

[Retourner à l'accueil.](#)

Cas : fichier vide

une erreur s'est produite!

Cause de l'erreur : Le fichier est vide.

[Retourner à l'accueil.](#)

TP : Gestion des clients v1.3

Ajouter la page **erreur.php** pour la gestion des erreurs liées à la lecture / écriture du tableau dans le fichier.

Utiliser les cookies

▪ Définition

Les cookies sont de simples fichiers textes (logés dans le dossier temporaire du navigateur), que le programmeur php demande au navigateur de stocker dans l'ordinateur client, afin d'enregistrer des paramètres liés à l'utilisateur (nom, âge, identifiant, préférences graphiques, ...)

L'utilisation des cookies consiste en 3 étapes :

1. Le serveur envoie des informations sur l'utilisateur au navigateur,
2. Le navigateur dépose ses informations sous forme de cookies dans la machine,
3. Lors de la prochaine connexion de l'utilisateur, le navigateur envoie les informations stockées dans les cookies au serveur pour que ce dernier reconnaisse l'utilisateur.

▪ Créer des cookies

```
setcookie(nom, valeur, durée_de_validité)
```

nom : nom du cookie

valeur : contenu du cookie

durée_de_validité : durée dans laquelle le cookie sera utilisable, au-delà il sera supprimé.

Tout comme les tableaux \$_POST et \$_GET qui contiennent les paramètres d'une requête, php propose un autre tableau **\$_COOKIE** qui contient tous les cookies ayant été créés par l'application.

▪ Accéder à un cookie

```
$_COOKIE["nom_cookie"]
```

▪ Vérifier si un cookie existe ou non

On peut vérifier si un cookie existe en suivant 2 méthodes :

```
// 1ere méthode : la fonction array_key_exists()
```

```
if(array_key_exists("nom_cookie", $_COOKIE)) {  
    // le cookie existe  
}
```

```
// 2ieme méthode : la fonction isset()
```

```
if(isset($_COOKIE["nom_cookie"])) {  
    // le cookie existe  
}
```

▪ Supprimer un cookie

Il n'y a pas de fonction spécifique pour supprimer un cookie, on utilise la même fonction de création **setcookie()** avec un délai d'expiration dépassé.

```
setcookie(nom, "", time()-60)
```

Exemple

```
<body>
<?php
    if(array_key_exists("nom", $_COOKIE) && array_key_exists("profession", $_COOKIE)) {
        echo "Bonjour " . $_COOKIE["nom"] . ".  

            "<br>Votre profession est : " . $_COOKIE["profession"];
        echo "<br><i>Ces informations proviennent des cookies</i>";
    } else {
        $username = "M. ELYAHYAOUI";
        $userjob = "ENSEIGNANT";
        // time() : retourne l'heure actuelle en secondes
        // 3600 : 60minutes
        setcookie("nom", $username, time() + 3600);
        setcookie("profession", $userjob, time() + 3600);
        echo "Bonjour $username".  

            "<br>Votre profession est : $userjob";
    }
?>
</body>
```

Bonjour M. ELYAHYAOUI
Votre profession est : ENSEIGNANT
Ces informations proviennent des cookies

Utiliser la session

La session est un deuxième moyen pour enregistrer des données sur l'utilisateur, et les rendre utilisables dans n'importe quelle page visitée par **cet utilisateur**.

Une session crée aussi un fichier, dans lequel on peut enregistrer autant de variables qu'on veut. Ce fichier se trouve à l'emplacement spécifié par le paramètre `session.save_path` du fichier de configuration **php.ini** de la version installée de PHP.

Lorsque la session d'un utilisateur est démarrée :

1. PHP crée pour cette session un identifiant unique : une chaîne de 32 caractères.
2. Un cookie appelé **PHPSESSID** est créé automatiquement pour contenir l'identifiant de 32 caractères.
3. Le fichier de la session est lui aussi créé (à l'emplacement `session.save_path`), il contient lui aussi l'identifiant de la session, sous cette forme `sess_identifiant`.
4. Quand une instruction php essaie de récupérer la valeur d'une variable appartenant à la session, PHP récupère l'identifiant contenu dans **PHPSESSID** et cherche dans le dossier `session.save_path` le fichier de la session contenant le même identifiant.
5. Si le fichier de la session est trouvé, la valeur de la variable demandée est retournée.
6. Fin de la session : une session est arrêtée quand l'utilisateur quitte le site. A partir de cet instant, passé un délai prédéfini (souvent 30min), le serveur détruit la session, et toutes les valeurs qu'elle contenait seront perdues.

▪ Démarrer une session

```
session_start(); /* Démarre la session si elle n'est pas encore démarrée.
```

```
L'appel de cette fonction doit être effectué au début de la page. */
```

▪ Ajouter une variable dans la session

```
$_SESSION["nom_variable"] = valeur;
```

▪ Vérifier si une variable existe dans la session

Les variables de session sont enregistrées dans un tableau nommé **\$_SESSION**

```
if(isset($_SESSION["nom_variable"])) {  
    // la variable existe  
}
```

▪ Détruire une variable de session / détruire la session

```
// détruire une variable de session
```

```
unset($_SESSION["nom_variable"]);
```

```
// détruire la session
```

```
session_destroy();
```

▪ Démarrer la session automatiquement

On peut se passer de l'appel de la fonction `session_start()` à chaque fois que l'application est lancée, et ceci en activant le paramètre **session.auto_start** dans le fichier de configuration **php.ini**.

```
session.auto_start = 1
```


PHP AVEC HTML5 ET BOOTSTRAP

TP : Gestion des clients v1.4

Partie 1 :

- Chercher dans les cookies les informations « nom d'utilisateur », et « date de dernière connexion » et les afficher comme ceci :

Bonjour M. ELYAHYAOU, I,

Date de dernière connexion : October 14 - 05:52 PM

NOM	ADRESSE	TEL	OPTIONS	
ELYAHYAOU	RABAT	0537363636	<input type="button" value="editer"/>	<input type="button" value="supprimer"/>
BENDRISS	RABAT	0537373737	<input type="button" value="editer"/>	<input type="button" value="supprimer"/>
ELYAHYAOU	KENITRA	06.65	<input type="button" value="editer"/>	<input type="button" value="supprimer"/>
ELYAHYAOU	RABAT	05 37 36 36 36	<input type="button" value="editer"/>	<input type="button" value="supprimer"/>
VINCI	RABAT	05.37.37.37.37	<input type="button" value="editer"/>	<input type="button" value="supprimer"/>

- Si l'info n'est pas stockée, diriger l'utilisateur vers une page qui lui demande son nom, enregistrer le nom et la date courante dans des cookies, puis aller vers la page d'accueil (liste_clients.php).

Veuillez vous identifier

La date courante du système est récupérée avec la fonction `date(chaine_formatee)`.
Exemple :

```
date(F j - h:i A) // donne : October 14 - 05:52 PM
```

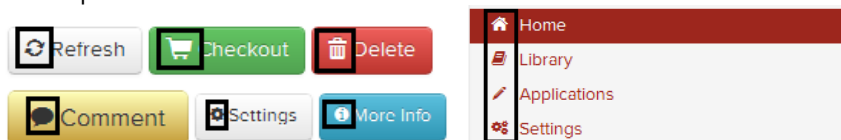
Chercher la documentation de cette fonction sur le site de documentation officiel de php :
<http://php.net/manual/>

Partie 2 :

- Télécharger et installer le frame-work Bootstrap (version 4) dans le projet, dans un dossier qui sera nommé « bootstrap ».
Pour son fonctionnement, Bootstrap 4 (version 4.1.0) a besoin aussi des bibliothèques Javascript suivantes : jquery-3.3.1, popper-1.14.js et tooltip-1.2.js qui doivent être téléchargées aussi.
- Pour utiliser Bootstrap dans une page, on doit inclure (dans cet ordre)
 - La feuille de style CSS qui contient le thème bootstrap, nommée **bootstrap.min.css** : à l'intérieur de la balise **head**, avec la balise **link**.
 - En bas de la page (de préférence) les bibliothèques jquery, popper et tooltip, puis **bootstrap.min.js** avec la balise **script**.
- Remplacer les balises html de la version actuelle par des balises qui utilisent les classes CSS3 du frame-work Bootstrap. S'inspirer des exemples de code HTML proposés sur le site <http://bootswatch.com>.
- Choisir un thème sur <http://bootswatch.com> et remplacer le thème par défaut de Bootstrap :
 - télécharger le fichier bootstrap.min.css du thème choisi,
 - copier son contenu,
 - remplacer le contenu du même fichier dans la version installée de Bootstrap, par le contenu copié depuis l'autre fichier.

- Sur le site <https://fontawesome.com/how-to-use/on-the-desktop/setup/getting-started> voir comment télécharger, installer et utiliser le frame-work Font-Awesome pour personnaliser les boutons de l'application.

Exemples :



- Le fichier CSS de fontawesome **fontawesome-all.css** doit être inclus avant la feuille de style de bootstrap.
- Optionnel : En utilisant les cookies, demander et enregistrer la préférence de l'utilisateur concernant le thème graphique (proposer un choix de plusieurs thèmes graphiques).

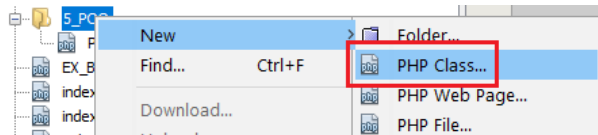
3. NOTIONS DE POO

A. Classes et objets

Les classes

▪ Définition

Une « classe » est une catégorie d'individus appelés « objets », ayant tous les mêmes propriétés – appelées « attributs » – et les mêmes comportements – appelées « méthodes » ou « opérations ».



Syntaxe de base :

```
class nom_classe {
    // déclaration des attributs et des méthodes
}
```

Exemple :

```
<?php
/**
 * Description of Personne
 *
 * @author s_elyahyaoui
 */
class Personne {
}
```

▪ Déclaration des attributs – Syntaxe de base

```
var $nom_attribut [= valeur par défaut];
// la valeur par défaut de l'attribut n'est pas obligatoire.
```

Exemple :

```
class Personne {
    var $nom;
    var $id = 0;
    var $adresse;
}
```

Déclarer & utiliser des objets

▪ Définition

- Chaque individu de la classe est représenté par un « objet ».
- Chaque objet détermine ses propres valeurs pour les attributs.
- En mémoire, deux objets ayant les mêmes valeurs pour les attributs sont considérés comme égaux.
- Chaque objet est créé par l'opérateur **new**.
- Cet opérateur fait l'appel du « **constructeur** » de la classe, si la classe ne définit pas explicitement son constructeur, le compilateur php lui en donne un par défaut.
- Le constructeur est une méthode qui sert à « instancier » la classe, c'à-d créer des objets de la classe.

▪ Créer & utiliser des objets avec le constructeur par défaut

```
$p1 = new Personne;
$p1->nom = "elyahyaoui"; $p1->id = 1; $p1->adresse = "maroc";
```

▪ Définir explicitement un constructeur

Le constructeur doit avoir la signature suivante : `__construct(parametres)`

Si une classe définit son constructeur, elle **n'a plus droit au constructeur par défaut.**

```
class Personne {
    var $nom;
    var $id = 0;
    var $adresse;

    function __construct($nom, $id, $adresse) {
        $this->nom = $nom;
        $this->id = $id;
        $this->adresse = $adresse;
    }
}
```



REMARQUE

- Dans toutes les méthodes de la classe, un objet fait référence à lui-même par l'opérateur `$this`.
- L'accès aux attributs se fait par l'opérateur `->`

Exemple :

```
<?php
require_once 'Personne.php';
$p1 = new Personne("ELYAHYAUI", 11, "Maroc");
echo "Bonjour, Je m'appelle M. $p1->nom";
?>
```

Bonjour, Je m'appelle M. ELYAHYAUI

▪ Les attributs statiques

- Un attribut statique ne change pas selon chaque objet, il garde la même valeur quel que soit l'objet. C'est pour cela que les attributs statiques sont souvent appelés « attributs de classe ».
- On peut accéder à des attributs statiques depuis la classe, sans instancier des objets.
- La déclaration des attributs statiques ne doit pas contenir le mot-clef `var`.
- A l'intérieur de la classe, un attribut statique est accessible à l'aide de l'opérateur `self::` au lieu de `$this->`
- A l'extérieur de la classe, un attribut statique est accessible comme ceci `NomClasse::attribut` ou `NomObjet::attribut`

▪ Le destructeur

- A l'inverse du constructeur, le destructeur est appelé lors de la destruction de la référence d'un objet.
- Le destructeur doit avoir la signature suivante : `__destruct()`
- Le destructeur est utilisé notamment pour libérer des ressources mémoires utilisées par l'objet (comme une connexion à une BD par ex.), mettre à jour des données statiques, ...

Exemple

```
<?php
/**
 * Description of Personne
 * @author s_elyahyaoui
 */
class Personne {
    var $nom;
    var $id = 0;
    var $adresse;

    static $nombre_personnes = 0;

    function __construct($nom, $id, $adresse) {
        $this->nom = $nom;
        $this->id = $id;
        $this->adresse = $adresse;
        self::$nombre_personnes++;
    }

    function __destruct() {
        self::$nombre_personnes--;
    }
}

<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <?php
    require_once 'Personne.php';
    $p1 = new Personne("ELYAHYAUI", 11, "Maroc");
    $p2 = new Personne("ELYAHYAUI", 11, "Maroc");
    if($p1 == $p2) {
        echo "p1 et p2 sont identiques";
    }
    echo "<br>test 1 - Nombre de personnes : ".Personne::$nombre_personnes;
    $p1 = NULL;
    echo "<br>test 2 - Nombre de personnes : ".Personne::$nombre_personnes;
    ?>
</body>
</html>
```

p1 et p2 sont identiques
test 1 - Nombre de personnes : 2
test 2 - Nombre de personnes : 1

B. Déclarer et utiliser des méthodes

Les méthodes

▪ Déclaration

```
function nom_fonction(paramètres) {  
    instructions  
    return une_valeur; // pour les méthodes qui doivent retourner une valeur  
}
```

▪ Appel

```
nomObjet->nom_fonction(paramètres);
```

Les méthodes statiques

▪ Syntaxe

Concernant la syntaxe, les méthodes statiques sont identiques aux méthodes d'objets.

▪ Mode d'accès

Concernant le mode d'accès, les méthodes statiques sont identiques aux attributs statiques.

Exemple

```
class Personne {  
    var $nom;  
    var $id = 0;  
    var $adresse;  
  
    static $nombre_personnes = 0;  
  
    function __construct($nom, $id, $adresse) {  
        $this->nom = $nom;  
        $this->id = $id;  
        $this->adresse = $adresse;  
        self::$nombre_personnes++;  
    }  
  
    function __destruct() {  
        self::$nombre_personnes--;  
    }  
  
    function sePresenter() {  
        echo "Bonjour,<br>Mon nom est $this->nom.";  
    }  
}  
  
<html>  
    <head>  
        <meta charset="UTF-8">  
        <title></title>  
    </head>  
    <body>  
        <?php  
            require_once 'Personne.php';  
            $p1 = new Personne("ELYAHYAOU", 11, "Maroc");  
            $p1->sePresenter();  
        ?>  
    </body>  
</html>
```

Bonjour,
Mon nom est ELYAHYAOU.

C. Principe « d'encapsulation »

Définition

- Les méthodes d'un objet A utilisent les attributs de cet l'objet. Donc si un autre objet B modifie la valeur de l'un des attributs de A, alors les méthodes de ce dernier pourraient ne pas fonctionner correctement.
- Donc pour bien fonctionner, les objets doivent avoir un mécanisme qui interdit l'accès libre (surtout en écriture) à leurs attributs, et qui le remplace par un accès « personnalisé » ou « contrôlé ».
- Ce mécanisme s'appelle l'encapsulation. Et impose aux classes de définir des méthodes appelées « accesseurs » ou bien « *getters and setters* », qui vont contrôler l'accès aux attributs des objets en lecture et en écriture.
- Exemple :

```
class Personne {
    static $nombre_personnes = 0;
    var $nom;
    var $id = 0;
    var $adresse;

    public function getNom() {
        return $this->nom;
    }
    public function getId() {
        return $this->id;
    }
    public function getAdresse() {
        return $this->adresse;
    }
    public function setNom($nom) {
        $this->nom = $nom;
    }
    public function setId($id) {
        $this->id = $id;
    }
    public function setAdresse($adresse) {
        $this->adresse = $adresse;
    }

    function __construct($nom, $id, $adresse) {
        $this->nom = $nom;
        $this->id = $id;
        $this->adresse = $adresse;
        self::$nombre_personnes++;
    }
    function __destruct() {
        self::$nombre_personnes--;
    }

    function sePresenter() {
        echo "Bonjour,<br>Mon nom est $this->nom.";
    }
}
```

Niveaux d'accès

- Pour que l'encapsulation fonctionne, il faut « restreindre » l'accès aux attributs (et aux méthodes) à l'aide de ce qu'on appelle des « modificateurs d'accès » :
 - **private** : invisible (pour la lecture et l'écriture) pour toutes les autres classes.
 - **protected** : invisible (pour la lecture et l'écriture) pour toutes les classes sauf les classes filles de la classe en question.

Les classes filles d'une classe sont les classes qui « héritent » de cette classe. Ex : la classe Client hérite de la classe Personne. Le principe d'héritage fera l'objet du prochain chapitre.

- **public** : visibles partout, ce qui veut dire que l'accès est libre pour les autres classes pour la lecture et l'écriture.
- Lorsque les modificateurs d'accès sont utilisés, on ne peut pas ajouter le mot-clef var.

▪ Exemple :

```
// classe Personne

class Personne {
    private static $nombre_personnes = 0;
    private $nom;
    private $id = 0;
    private $adresse;


    public static function getNombre_personnes() {
        return self::$nombre_personnes;
    }

    public function getNom() {
        return $this->nom;
    }
    public function getId() {
        return $this->id;
    }
    public function getAdresse() {
        return $this->adresse;
    }
    public function setNom($nom) {
        $this->nom = $nom;
    }
    public function setId($id) {
        $this->id = $id;
    }
    public function setAdresse($adresse) {
        $this->adresse = $adresse;
    }
}
```

Ex. : Si la classe décide d'interdire l'accès en écriture à son attribut id, elle peut supprimer le setter de cet attribut.

// index.php

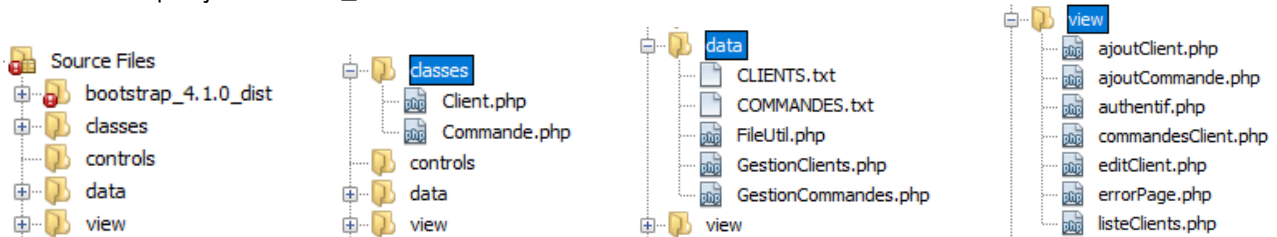
```
7 <html>
8 <head>
9 <meta charset="UTF-8">
10 <title></title>
11 </head>
12 <body>
13 <?php
14 require_once 'Personne.php';
15 $p1 = new Personne("ELYAHYAUI", 11, "Maroc");
16 $p1->nom = "test";
17 $p1->sePresenter();
18 ?>
19 </body>
20 </html>
```

 Fatal error: Cannot access private property Personne::\$nom in C:\wamp64\www\PhpProject1\5_POO\index_poo.php on line 16

Call Stack				
#	Time	Memory	Function	Location
1	0.0118		237928 {main}()	...index_poo.php:0

TP : Gestion des clients v1.5

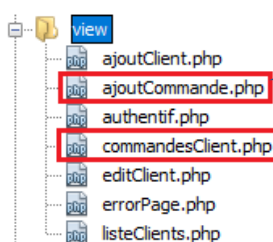
- Créer un projet Clients_1.5 dont la structure sera comme ceci :



- Le dossier **classes** :
 - **Classe Client** : **id**, **nom**, **adresse** et **tel**, privés, avec accesseurs, et **constructeur qui ne prend pas l'id en paramètre**.
 - **Classe Commande** : **id**, **date**, **type** (Normale / Express), **payée** (Oui / Non), et **idClient**, privés, avec accesseurs, et **constructeur qui ne prend pas l'id en paramètre**.
- Le dossier **data** :
 - **CLIENTS.txt** et **COMMANDES.txt** : contiendront respectivement la liste totale des clients et commandes.
 - **Classe FileUtil** : contiendra 2 méthodes statiques :
 - **serializeArray(\$tableau, \$nom_fichier)** : sérialise le tableau dans le fichier et ne retourne rien.
 - **unserializeArray(\$nom_fichier)** : désérialise le contenu du fichier dans un tableau et retourne ce tableau.
 - **Classe GestionClients** :
 - Attributs (privés) : **liste** (tableau des clients), et **filename** (chemin du fichier où sera sérialisé le tableau de clients).
 - Méthodes : constructeur sans paramètres, **getAll()**, **insert(\$client)**, **update(\$client)**, **delete(\$idClient)** et **incrementID()**.
 - **Constructeur** : initialise l'attribut **filename**, récupère la liste des clients et met cette liste dans la session.
 - **getAll** : récupère les clients depuis le fichier et met le résultat dans l'attribut **liste**.
 - **insert** : ajoute le client dans le tableau et met à jour le tableau dans la session. L'identifiant du client ajouté doit être **calculé dans la méthode incrementID**. La **clef** de ce client dans le tableau sera **son identifiant**.
 - **update** : remplace le client par celui qui est passé en paramètre, et met à jour le tableau dans la session.
 - **delete** : supprime le client dont l'id est passé en paramètre, met à jour le tableau dans la session, et remplace l'idClient de ses commandes par **0**.
 - **incrementID** : retourne la valeur de l'identifiant du nouveau client, en fonction de celui du dernier client dans le tableau.
 - **Classe GestionCommandes** :
 - Attributs (privés) : **liste** (tableau des commandes d'un client spécifique), **listeComplete** (liste de toutes les commandes), **filename** (chemin du fichier où sera sérialisé le tableau **listeComplete**).
 - Méthodes : constructeur sans paramètres, **getAll()**, **getByClient(\$idClient)**, **insert(\$client)**, **update(\$client)**, **delete(\$idClient)** et **incrementID()**. Les méthodes de cette classe n'utiliseront pas la session.
 - **Constructeur** : initialise l'attribut **filename**.

- **getAll** : récupère toutes les commandes depuis le fichier et met le résultat dans l'attribut **listeComplete**.
- **getByClient** : récupère toutes les commandes d'un client et met le résultat dans l'attribut **liste**.
- **insert** : ajoute une commande dans **listeComplete**. L'identifiant est **calculé dans la méthode incrementID**. La **clef** de la commande sera **son identifiant**.
- **update** : remplace une commande.
- **delete** : supprime une commande.
- **incrementID** : idem que dans la classe **GestionClient**.

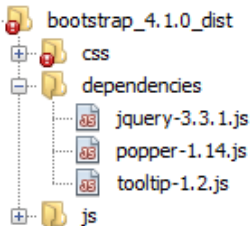
- Le dossier **view** : contiendra uniquement les pages php, dont les 2 nouvelles sont ajoutCommande.php et commandesClient.php.



Ajouter une commande pour un client. Sera accessible depuis la page commandesClient.php

Afficher les commandes d'un client. Sera accessible depuis la page listeClients.php

- Le dossier **bootstrap_dist** : contiendra les fichiers nécessaires pour l'utilisation de Bootstrap.



- Aperçus d'IHM :

Tapez votre nom d'utilisateur :

Bonjour M. ELYAHYAOU,
 Date de dernière connexion : November 24 - 10:48 AM.

Liste clients

[Ajouter](#)

ID	NOM	ADRESSE	TEL	OPTIONS		
1	M. ELYAHYAOU SOUFIANE	KENITRA	05.37.37.37.37	afficher commandes	editer	supprimer
2	ECOLE SUPÉRIEURE VINCI	RABAT VILLE	05.37.37.37.37	afficher commandes	editer	supprimer

Modifier un client

NOM

ADRESSE

TEL

Liste des commandes du client :

ID - 1

NOM - M. ELYAHYAOU SOUFIANE

ADRESSE DE LIVRAISON - KENITRA

TEL - 05.37.37.3737

[Ajouter une commande](#)

ID	DATE	TYPE	PAYEE	ID CLIENT	OPTIONS
1	2018-11-14	Express	Oui	1	<input type="button" value="Voir détails"/>
2	2018-11-05	Normale	Non	1	<input type="button" value="Voir détails"/>

[Retourner à la liste des clients](#)

Liste des commandes du client :

ID - 2

NOM - ECOLE SUPÉRIEURE VINCI

ADRESSE DE LIVRAISON - RABAT VILLE

TEL - 05.37.37.3737

[Ajouter une commande](#)

ID	DATE	TYPE	PAYEE	ID CLIENT	OPTIONS
aucune commande					

[Retourner à la liste des clients](#)

Nouvelle commande pour le client :

ID - 1

NOM - M. ELYAHYAOU SOUFIANE

ADRESSE DE LIVRAISON - KENITRA

TEL - 05.37.37.3737

DATE

TYPE

PAYEE

Nouvelle commande pour le client :

ID - 1

NOM - M. ELYAHYAOU SOUFIANE

ADRESSE DE LIVRAISON - KENITRA

TEL - 05.37.37.3737

DATE

TYPE

PAYEE

D. Classes abstraites et Héritage

Méthodes abstraites

- Une méthode est dite « abstraite » quand elle n'a pas de « corps », c'àd qu'on ne sait pas quels traitements elle va effectuer.
- Syntaxe :

```
abstract function nom(paramètres);
```

Classes abstraites

- Une classe qui contient au moins une méthode abstraite est elle aussi abstraite.
- Une classe abstraite ne peut pas être instanciée, même si elle peut avoir des attributs et des constructeurs.
- Une classe abstraite peut avoir des méthodes concrètes. Une classe peut être abstraite même si elle n'a aucune méthode abstraite.
- Syntaxe :

```
abstract class Personne {
    // déclarations
}
```

Exemple

```
<?php
/**
 * @author s_elyahyaoui
 */
abstract class Personne {
    protected $nom, $id, $adresse, $profession;

    public function getNom() { return $this->nom; }
    public function getId() { return $this->id; }
    public function getAdresse() { return $this->adresse; }
    public function getProfession() { return $this->profession; }
    public function setNom($nom) { $this->nom = $nom; }
    public function setId($id) { $this->id = $id; }
    public function setAdresse($adresse) { $this->adresse = $adresse; }
    public function setProfession($profession) { $this->profession = $profession; }

    function __construct($nom, $id, $adresse, $profession) {
        $this->nom = $nom;
        $this->id = $id;
        $this->adresse = $adresse;
        $this->profession = $profession;
    }

    abstract function sePresenter();
}
```

Héritage

- Une classe – abstraite ou concrète – peut avoir des « classes filles » (on dit aussi classes dérivées ou sous-classes)
- Syntaxe :

```
// classe fille C1 hérite de la classe mère C2
class C1 extends C2 {
    // contenu de C1
}
```

- Une classe ne peut pas avoir plusieurs classes mères.
- Tous les membres (attributs et méthodes) publics et protégés de la classe mère sont hérités dans les classes filles.
- Si la classe mère contient des méthodes abstraites, alors la classe fille doit les définir, ou elle doit être elle aussi abstraite.

```
class Etudiant extends Personne {
}
//\Etudiant is not abstract and does not override abstract method sePresenter() in \Personne
//-----
```

- Lors de la définition / redéfinition d'une méthode héritée depuis la classe mère, la classe fille **ne doit pas modifier** la signature de la méthode.

```
class Enseignant extends Personne {
    public function sePresenter($param) {
        echo "Bonjour, c'est $this->nom, $this->profession";
    }
}
```

 Fatal error: Declaration of Enseignant::sePresenter() must be compatible with Personne::sePresenter() in C:\wamp64\www\PhpProject15_POO\Enseignant.php on line 18

Call Stack

#	Time	Memory	Function	Location
1	0.0030	239792	{main}()	...index_poo.php:0

- Les méthodes déclarées avec le mot-clef **final** **ne peuvent pas être redéfinies** dans les classes filles.
- Le constructeur de la classe fille peut faire l'appel de celui de la classe mère :

```
class C1 extends C2 {
    public function __construct(paramètres) {
        parent::__construct(paramètres_du_constructeur_parent);
    }
}
```

Exemple

```
class Etudiant extends Personne {
    public function __construct($nom, $id, $adresse, $profession) {
        parent::__construct($nom, $id, $adresse, $profession);
    }

    public function sePresenter() {
        echo "Salut, je m'appelle $this->nom, je suis $this->profession";
    }
}

class Enseignant extends Personne {
    public function __construct($nom, $id, $adresse, $profession) {
        parent::__construct($nom, $id, $adresse, $profession);
    }

    public function sePresenter() {
        echo "Bonjour, c'est $this->nom, $this->profession";
    }
}

<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <?php
        require_once 'Personne.php';
        require_once 'Etudiant.php';
        require_once 'Enseignant.php';
        $p1 = new Etudiant("M.ELYAHYAUI", 1, "kenitra", "etudiant");
        $p2 = new Enseignant("M.ELYAHYAUI", 2, "kenitra", "enseignant");

        $p1->sePresenter();
        echo '<br>';
        $p2->sePresenter();
    <?>
</body>
</html>
```

Salut, je m'appelle M.ELYAHYAOUI, je suis etudiant Bonjour, c'est M.ELYAHYAOUI, enseignant

S.ELYAHYAOUI - Poo en PHP

E. Interfaces et Implémentation

Définition

- Tout comme les classes, une interface est aussi considérée comme un type, mais ce n'est pas vraiment une classe.
- Une interface peut être vue comme un *masque*, ou un *déguisement*, qu'on fait porter à nos objets, pour qu'ils adoptent des attitudes différentes, dans certaines situations.
- Une interface peut être vue aussi comme un squelette qui servira à créer des classes qui ont les mêmes comportements.
- Vu que ce sont des *attitudes à adopter*, et non des classes, les interfaces sont **implémentées** - et non héritées - par les classes.
- Toutes les méthodes d'une interface sont **publiques et abstraites**, même sans utiliser les modificateurs `public abstract`.
- Les interfaces **n'ont pas d'attributs d'objets**, uniquement des constantes statiques, même si le modificateur `static` est interdit.

```
interface NomInterface {  
    const constante1 = valeur;  
  
    function fonction1(paramètres);  
    function fonction2(paramètres);  
}
```

- Une classe peut hériter une seule classe, mais peut implémenter plusieurs interfaces.

```
class C1 extends C2 implements Interface1, Interface2 {  
    /* implémentation des méthodes de C2, Interface1 et Interface2 */  
}
```

- Une interface peut hériter une (ou plusieurs) autre interface.

4. ACCES AUX BASES DE DONNEES MYSQL

Choix du pilote

- MySQL est un SGBD (système de gestion de bases de données) parmi d'autres (Oracle, SQLServer, Access, PostgreSQL, SQLite, ...).
- Un SGBD ne comprend que les instructions – appelées « requêtes » – écrites en langage SQL (*Structured Query Language*).
- PHP n'est pas un langage d'accès aux SGBD, donc pour dialoguer avec une base de données hébergée dans un SGBD, une application PHP (ou Java, ou C#, ...) a besoin d'un « pilote » pour ce SGBD.
- Ce pilote est un ensemble de classes qui contiennent des méthodes qui servent à effectuer des actions comme : se connecter à une base de données, chercher des données dans une table, insérer des données, ...
- PHP propose 3 pilotes, appelé aussi extensions :
 - **MYSQL :**
 - Obsolète depuis la version 5.5 de PHP et retirée de PHP depuis la version 7,
 - Son utilisation n'est pas recommandée.
 - **MYSQLi :**
 - Fonctionnalités améliorées par rapport à l'extension MYSQL,
 - Mais adapté uniquement aux SGBD MySQL et MariaDB.
 - **PDO :**
 - Aussi complet que MYSQLi,
 - Plus utilisé que MYSQLi dans les dernières versions de PHP,
 - On peut l'utiliser avec plusieurs SGBD (Oracle, SQLServer, Access, PostgreSQL, SQLite et autres)
 - Permet d'écrire le même code php quel que soit le SGBD.

Adresser des requêtes à MySQL avec PDO

1. Etablir une connexion avec une BD MySQL.
2. Rédiger le code SQL qu'on souhaite exécuter. Il existe 2 types de requêtes SQL :
 - Requête de « sélection » de données.
 - Requête de « mise à jour » de données.
3. Exécuter le code SQL pour obtenir un résultat sous forme de « curseur ».
4. Exploiter le curseur.
5. Fermer le curseur pour pouvoir :
 - Réexécuter à nouveau la même requête,
 - Ou exécuter une nouvelle requête,
 - Ou tout simplement libérer les ressources mémoire utilisées par la requête.

A. Se connecter à une base de données MySQL

Créer un objet PDO

- On se connecte à une BD en créant un objet de type PDO. Cet objet représentera la connexion avec la BD et c'est à travers lui qu'on pourra exécuter des requêtes SQL.
- On crée un objet PDO avec un appel du constructeur, qui doit prendre 3 paramètres.
- Syntaxe :

```
$bdd = new PDO('mysql:host=adresse_ip;port=numero_port;dbname=nom_BD',
               'username',
               'password');
```

Chaîne de connexion.

Compte d'utilisateur MySQL.

Mot de passe du compte.

adresse_ip

Adresse ip ou nom réseau de la machine sur laquelle MySQL est hébergé. « localhost » ou « 127.0.0.1 » si MySQL est sur la machine locale.

numero_port

Numéro du port réseau du service mysql. MySQL prend par défaut le port 3306, si cette valeur n'est pas explicitement modifiée lors de l'installation de MySQL, on peut ne pas la spécifier dans la chaîne de connexion : 'mysql:host=*adresse_ip*;dbname=*nom_BD*'

nom_BD

Nom de la base de données. La BD doit être accessible pour le compte spécifié dans le 3ième paramètre du constructeur.

Username

On utilise généralement l'utilisateur admin de MySQL, appelé « root », pour s'assurer de pouvoir accéder d'avoir tous les privilèges de lecture et écriture sur toutes les BD existantes dans MySQL.

Password

Mot de passe du compte. Lors de l'installation de MySQL, le pass du compte root est vide, il est recommandé de le modifier.

- Les opérations relatives aux BD (connexion et exécution de requêtes) peuvent lever des exceptions, principalement dues aux raisons suivantes :
 - Erreur de communication réseau avec le SGBD, ex. : adresse IP erronée,
 - Mot de passe erroné,
 - Erreur SQL,
 - Etc.

Exemple

```
<html>
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <?php
    $host = "127.0.0.1";
    $port = "3307";
    $dbName = "personnes_pdo";
    $userName = "root";
    $pass = "pass1156";
    try {
      $bdd = new PDO("mysql:host=$host;port=$port;dbname=$dbName",
        $userName,
        $pass);
      echo "connexion etablie ...";
    } catch (Exception $ex) {
      $err = $ex->getMessage();
      echo $err;
    }

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

connexion etablie ...

B. Requête de mise à jour de données

Définition

- Toutes les requêtes INSERT – UPDATE – DELETE ou CREATE TABLE – ALTER TABLE ... sont considérées comme des requêtes de mise à jour.
- Syntaxe :

```
// 1iere methode (déconseillée)
$pdo->query("code sql concaténé avec les valeurs à insérer / modifier / supprimer");

// 2ieme methode : requêtes paramétrées (recommandée)
$requete = $pdo->prepare("code sql contenant des paramètres");
$tableau_valeurs = array(
    "param1" => valeur1,
    "param2" => valeur2,
    .....
);
$requete->execute($tableau_valeurs);
$requete->closeCursor(); // fermer la requête.
```

Requête INSERT

```
require_once '../6_pdo/Personne.php';

$host = "127.0.0.1";
$port = "3307";
$dbName = "personnes_pdo";
$username = "root";
$pass = "pass1156";
try {
    $bdd = new PDO("mysql:host=$host;port=$port;dbname=$dbName",
        $username,
        $pass);
    echo "connexion etablie ...";
    $personne = new Personne(0, "CHRISTOPHER", "CHANCE");
    $requete = $bdd->prepare("INSERT INTO personne VALUES(NULL, :nom, :prenom)");
    $liste_valeurs = array(
        "nom" => $personne->getNom(),
        "prenom" => $personne->getPrenom()
    );
    $requete->execute($liste_valeurs);
    echo "<br>id de la personne personne ajoutée : ".$bdd->lastInsertId();
} catch (Exception $ex) {
    $err = $ex->getMessage();
    echo $err;
}
```

Récupération de l'identifiant de la ligne insérée, au cas où la clef primaire est auto-incrémentée.

connexion etablie ...

id de la personne personne ajoutée : 1

Objects		
personne @personnes_pdo...		
Begin Transaction Memo Filter So		
id	nom	prenom
1	CHRISTOPHER	CHANCE

Requête UPDATE ou DELETE

```
$host = "127.0.0.1";
$port = "3307";
$dbName = "personnes_pdo";
$username = "root";
$password = "pass1156";
try {
    $bdd = new PDO("mysql:host=$host;port=$port;dbname=$dbName",
        $username,
        $password);
    echo "connexion etablie ...";
    $requete = $bdd->prepare("UPDATE personne SET NOM = :nom, PRENOM = :prenom WHERE ID = :id");
    $liste_valeurs = array(
        "nom" => "LAVERNE",
        "prenom" => "WINSTON",
        "id" => "2"
    );
    $requete->execute($liste_valeurs);
    echo "<br>valeurs modifiees";
} catch (Exception $ex) {
    $err = $ex->getMessage();
    echo $err;
}
```

Objects | personne @personnes_pdo...

Begin Transaction | Memo | Filter | Sort

id	nom	prenom
1	CHANCE	CHRISTOPHER
2	LAVERNE	WINSTON

C. Requête de sélection de données

Définition

- Requête `SELECT val1, val2, ... FROM ...`
- Ce genre de requête retourne un curseur sous forme de tableau de lignes, avec plusieurs valeurs (`val1`, `val2`, ...) dans chaque ligne.
- Chaque ligne est un tableau associatif, qu'on récupère avec la fonction `fetch()`.
- Syntaxe :

```
// 1ier cas : requête nom paramétrée
$requete = $pdo->query("requête SELECT");
while($ligne = $requete->fetch()) {
    pour récupérer une valeur : $ligne["nom_valeur"]
}

// 2ième cas : requête paramétrée
$requete = $pdo->prepare("requête SELECT contenant des paramètres");
$tableau_valeurs = array(
    "param1" => valeur1,
    "param2" => valeur2,
    .....
);
$requete->execute($tableau_valeurs);
while($ligne = $requete->fetch()) {
    pour récupérer une valeur : $ligne["nom_valeur"]
}
```

Exemple : afficher toutes les personnes

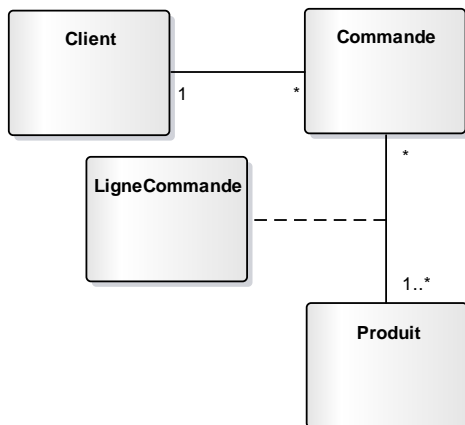
id	nom	prenom
1	CHANCE	CHRISTOPHER
2	LAVERNE	WINSTON
3	PUCCI	ILSA
4 (Null)		AMES
5 (Null)		GUERRERO
6	WALTERS	KATHERINE
7 (Null)		BAPTISTE
8	GALLEGO	MARIA

```
$host = "127.0.0.1";
$port = "3307";
$dbName = "personnes_pdo";
$username = "root";
$pass = "pass1156";
try {
    $bdd = new PDO("mysql:host=$host;port=$port;dbname=$dbName",
        $username,
        $pass);
    echo "connexion etablie ...";
    echo "<br>liste des personnes<br>";
    $requete = $bdd->query("SELECT * FROM personne");
    while ($ligne = $requete->fetch()) {
        echo "*" . $ligne['id'] . " " . $ligne['prenom'] . " " . $ligne['nom'];
        echo "<br>";
    }
    $requete->closeCursor();
} catch (Exception $ex) {
    $err = $ex->getMessage();
    echo $err;
}
```

```
connexion etablie ...
liste des personnes
* 1 CHRISTOPHER CHANCE
* 2 WINSTON LAVERNE
* 3 ILSA PUCCI
* 4 AMES
* 5 GUERRERO
* 6 KATHERINE WALTERS
* 7 BAPTISTE
* 8 MARIA GALLEGO
```

TP : Gestion des clients v2.0

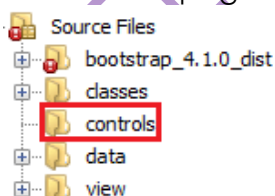
- Créer une BD nommée « commercial » dans MySQL.
- Peupler la BD avec la traduction SQL du diagramme de classes suivant :



```

1 CREATE TABLE CLIENT(
2   ID INT(11) PRIMARY KEY AUTO_INCREMENT,
3   NOM VARCHAR(20),
4   ADRESSE VARCHAR(200), TEL VARCHAR(15)
5 );
6
7 CREATE TABLE COMMANDE(
8   ID INT(11) PRIMARY KEY AUTO_INCREMENT,
9   DATECOM DATE,
10  TYPE INT(1), -- 1 : EXPRESS / 0 : NORMALE
11  PAYEE INT(1), -- 0 : NON PAYEE / 1 : PAYEE
12  ID_CLIENT INT(11) NOT NULL,
13  CONSTRAINT FOREIGN KEY (ID_CLIENT) REFERENCES CLIENT(ID)
14 );
15
16 CREATE TABLE PRODUIT(
17   ID INT(11) PRIMARY KEY AUTO_INCREMENT,
18   DESIGNATION VARCHAR(200),
19   QTE_STOCK INT(3),
20   PRIX_UNIT DECIMAL(7,2)
21 );
22
23 CREATE TABLE LIGNECOMMANDE(
24   ID_COMMANDE INT(11),
25   ID_PRODUIT INT(11),
26   QTE_COMMANDEE INT(2),
27   PRIMARY KEY (ID_PRODUIT, ID_COMMANDE),
28   CONSTRAINT FOREIGN KEY (ID_COMMANDE) REFERENCES COMMANDE(ID),
29   CONSTRAINT FOREIGN KEY (ID_PRODUIT) REFERENCES PRODUIT(ID)
30 );
  
```

- Créer un projet Clients_2.0.
- Les fichiers texte contenant les listes d'objets seront abandonnés, ainsi que la classe FileUtil.
- Les classes GestionClient, GestionCommande, ... vont désormais interroger la BD « commercial » plutôt que les anciens fichiers texte, avec la contrainte suivante : la connexion avec la BD doit être faite **une seule fois**. Il faut donc créer une classe DBConnect qui contiendra un attribut PDO qui sera un « singleton ».
- Les classes GestionClient, GestionCommande et GestionProduit vont hériter d'une classe abstraite nommée Gestion, qui contiendra les méthodes getAll, insert, delete, update et les attributs tableName et sqlString.
- Les méthodes de la classe Gestion qui auront le même comportement dans les classes filles n'ont pas besoin d'être abstraites.
- Dans les classes d'accès aux données, après chaque action effectuée (récupérer, ajouter, supprimer, ...), les méthodes (getAll, insert, delete, ...) doivent retourner la nouvelle liste.
- Dans le dossier « controls », créer une classe nommée Controls.php qui fera l'intermédiaire entre les pages web et les classes GestionClient, GestionCommande, ...



- Toutes les actions provenant des pages web du dossier « view » feront désormais appel à des méthodes de la classe Controls.php, qui feront appel aux méthodes des classes GestionClient, GestionCommande, ...
Exemple - traitement de la demande de mise à jour d'un client au niveau de la page listeClients.php :

```
if ($_POST["action"] == "edit") {  
    // RECUPERER LES DONNEES DU CLIENT A EDITER  
    $nomcl = $_POST["nom"];  
    $adrcl = $_POST["adr"];  
    $telcl = $_POST["tel"];  
    $id = $_POST["idcl"];  
    // REMPLACER LE CLIENT DANS LE TABLEAU  
    $new_client = new Client($nomcl, $adrcl, $telcl);  
    $new_client->setId($id);  
    $gestionClients->update($new_client);  
}
```

Dorénavant la page listeClients.php n'appellera plus les méthodes de la classe GestionClient, mais celles de la classe Controls, qui à son tour va appeler les méthodes de la classe GestionClient.

Exemple d'une méthode de la classe Controls.php :

```
ajouterClient($client) {  
    // créer un objet GestionClient  
    // appeler la méthode ajout de cet objet  
    // retourner la nouvelle liste de clients  
}
```

- La classe Controls.php contiendra 3 attributs de types GestionClient, GestionCommande et GestionProduits qui seront utilisés pour gérer les différentes actions de CRUD sur les clients, commandes et produits.

Etapes à suivre :

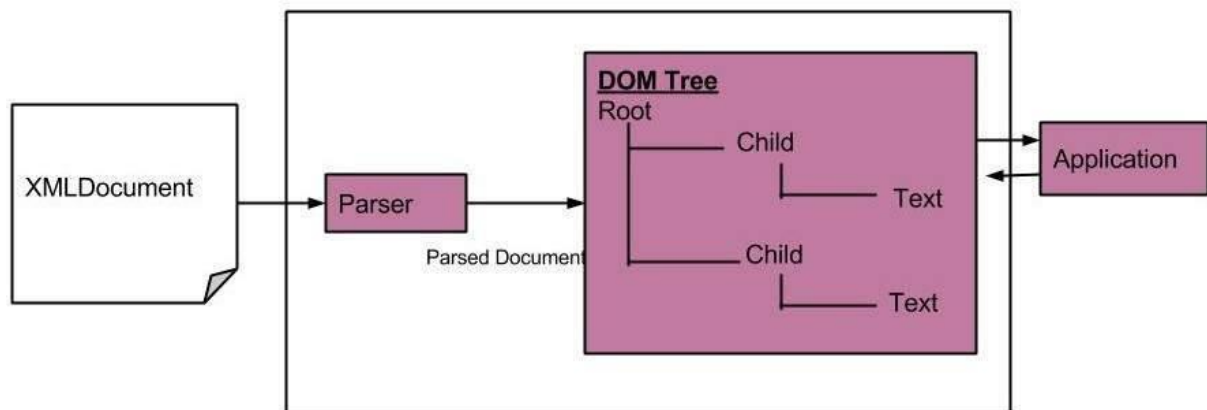
1. Gestion des produits :
Liste, ajout, modification et suppression.
2. Gestion des clients :
Liste, ajout, modification, suppression et affichage des commandes d'un client.
Si un client possède des commandes, il ne doit pas être supprimé.
3. Gestion des commandes d'un client :
Liste, ajout, suppression, afficher la liste des produits d'une commande, ajouter un produit dans une commande (utiliser uniquement un combobox pour afficher la liste des produits).
Une commande payée ne peut être ni supprimée ni modifiée.

II. LES APPELS AJAX

1. NOTIONS DE BASE SUR « DOM »

Définition

- Pour pouvoir exploiter un document HTML, il nous faut un ensemble de règles et de fonctionnalités qui vont nous permettre de lire et interpréter le contenu de ce document (la balise racine, les sous-balises, les attributs, le CSS, ...).
- Cet ensemble de règles et de fonctionnalités est appelé « **un parseur** », et le langage Javascript que nous utiliserons en parallèle avec PHP dans ce chapitre, utilise le parseur DOM.
- Le parseur « DOM » (*Document Object Model*) définit la manière logique dont la structure d'un document XML ou HTML doit être lue / modifiée par Javascript (d'autres langages l'utilisent aussi : Java, C#, ...).
- DOM considère un document HTML comme **un arbre** ou **arborescence**, et chaque élément (balise, ou attribut de balise, ou contenu de balise, ...) est considéré comme **un nœud**. Les nœuds sont liés entre eux par des relations parent/enfant.



- DOM fournit à javascript un ensemble d'objets et de fonctions qui vont permettre d'accéder au code HTML de la page pour récupérer ou modifier le contenu d'une balise, ajouter des balises, etc.
- Quand le navigateur envoie une requête au serveur, ce dernier renvoie une page web dont la structure (ou l'arbre) est « fixe », et on ne peut pas la changer, sauf en utilisant DOM et Javascript, mais l'avantage – par rapport à PHP – c'est qu'un appel de code PHP produira le rechargement de toute la page, tandis que Javascript permet de cibler uniquement une partie de l'arbre DOM de la page.

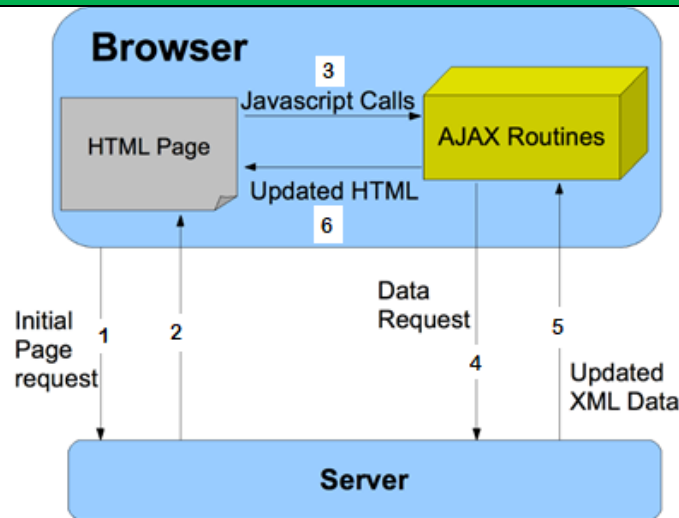
2. NOTIONS DE BASE SUR AJAX

Définition

La technologie AJAX (*Asynchronous Javascript And XML*) permet au langage Javascript d'envoyer des requêtes au **serveur** (donc un appel de code PHP), pour charger **uniquement une partie de la page**, et non toute la page. Ces requêtes envoyées par Ajax, sont gérées par Javascript, et le format d'échange de données - au contraire des requêtes http traditionnelles (format HTML) - est soit XML, JSON, HTML ou du simple « texte ».

Ajax est très utilisé, quelle que soit la technologie de l'application web (Java, PHP, ASP.Net, ...), surtout pour les applications de type *Single Page Applications*.

Mode de fonctionnement



1. Le browser envoie une requête http traditionnelle envoyée au serveur.
2. La page est chargée toute entière.
3. Suite à un évènement (bouton, combobox, ...), un appel javascript est adressé au moteur Ajax.
4. Le moteur Ajax envoie une requête http au serveur.
5. Le moteur Ajax reçoit la réponse du serveur sous format XML ou JSON.
6. A l'aide de Javascript on modifie l'arbre DOM de la page en fonction des données XML ou JSON récupérées par la requête Ajax.

3. LES FORMATS D'ECCHANGE DE DONNEES XML ET JSON

XML & JSON

- XML et JSON sont tous deux des formats d'échange de données qu'on peut utiliser avec Ajax (dans les services web aussi, et dans d'autres architectures)
- XML est un langage de balisage (comme HTML) générique, qui représente les données (Ex : le contenu d'un objet php) sous forme d'un ensemble de balises mères / filles.
- JSON (Javascript Object Notation) est un format de représentation de données qui utilise – comme son nom l'indique – la syntaxe des objets Javascript.
- JSON est mieux adapté que XML pour faire des appels Ajax, car un résultat JSON est nativement interprétable par Javascript, au contraire d'un résultat XML qui doit être transformé en objet Javascript.
- Dans un contenu JSON on peut avoir :
 - Un seul objet ayant des attributs,
 - Une liste d'objets, avec des attributs dans chaque objet,
 - Un attribut peut lui-même être un objet ayant des attributs,
 - Un attribut peut lui-même être une liste d'objets,
- Syntaxe JSON :
 - { ... } Objet. Peut avoir un ou plusieurs attributs.
 - "nom" : "valeur" Attribut.
 - "nom" : { ... } Attribut de type objet.
 - [...] Tableau. Un élément peut être un attribut, objet ou tableau.

- o "nom" : [...] Attribut de type tableau.

Exemple

```
<?xml version="1.0" encoding="UTF-8" ?>
  <RestResponse>
    <result>
      <countryIso2>US</countryIso2>
      <stateAbbr>CA</stateAbbr>
      <postal>94043</postal>
      <continent>North America</continent>
      <state>California</state>
      <longitude>-122.0574</longitude>
      <latitude>37.4192</latitude>
      <ds>II</ds>
      <network>AS15169 Google Inc.</network>
      <city>Mountain View</city>
      <country>United States</country>
      <ip>172.217.3.14</ip>
    </result>
  </RestResponse>
```

Mêmes données représentées sous format JSON :

```
{
  "RestResponse" : {
    "result" : {
      "countryIso2" : "US",
      "stateAbbr" : "CA",
      "postal" : "94043",
      "continent" : "North America",
      "state" : "California",
      "longitude" : "-122.0574",
      "latitude" : "37.4192",
      "ds" : "II",
      "network" : "AS15169 Google Inc.",
      "city" : "Mountain View",
      "country" : "United States",
      "ip" : "172.217.3.14"
    }
  }
}
```

4. EFFECTUER UN APPEL AJAX AVEC JAVASCRIPT ET JQUERY

A. Notions de base sur JQuery

Définition

jQuery est une bibliothèque de Javascript, téléchargeable gratuitement, qu'on utilise pour faciliter et réduire le code qu'on écrit avec Javascript.

La cible de son utilisation est la facilitation des tâches suivantes :

- Modifier la structure DOM de la page,
- Modifier le CSS, les animations, ...
- Traitement des événements,
- **Faire des requêtes avec Ajax.**

Toutes les fonctionnalités de la bibliothèque sont accessibles en appelant la fonction principale `jquery`, ou son abréviation `$`

Exemple

```
// récupérer un élément avec javascript par son ID
var elem = document.getElementById('un_id');
// équivalent avec jquery
var elem = $('#un_id');

// modifier le contenu d'une zone de texte
var elem = document.getElementById('un_id');
elem.value = 'nouvelle_valeur';
// équivalent avec jquery
$('#un_id').val('nouvelle_valeur');
```

B. Effectuer une requête Ajax avec JQuery

Syntaxe de base

```
$.ajax(
{
    type: "POST" ou bien "GET",
    url: "une_url", // url d'une page php, page jsp, servlet, ...
    data: {param1: 'value1', param2: 'value2', .....}, // paramètres de la requête - pour POST
    success: fonction1(res), // fonction qui sera appelée en cas de réussite de la requête
    error: fonction2(res), // fonction qui sera appelée en cas d'échec de la requête
    dataType: "XML" ou bien "JSON" ou bien "TEXT" ..... // type des données qui seront récupérées
}
);
```



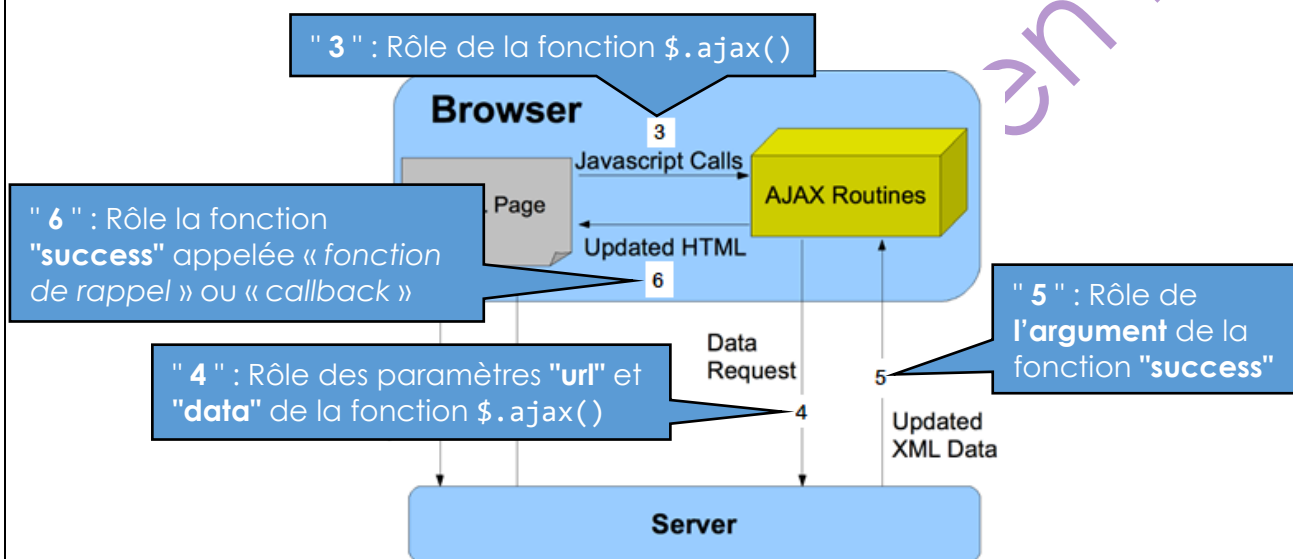
REMARQUES

- On peut laisser le dernier paramètre (`dataType`) vide, pour forcer JQuery à détecter automatiquement le format des données retournées par la requête.

- La fonction spécifiée dans le paramètre `success` doit prendre un argument, que la fonction `$.ajax()` va automatiquement initialiser avec le flux de données (XML, JSON, ...) retournées par la requête.

Rappels :

- Le browser envoie une requête http traditionnelle envoyée au serveur.
- La page est chargée toute entière.
- Suite à un évènement (bouton, combobox, ...), un appel javascript est adressé au moteur Ajax.
- Le moteur Ajax envoie une requête http au serveur.
- Le moteur Ajax reçoit la réponse du serveur sous format XML ou JSON.
- A l'aide de Javascript on modifie l'arbre DOM de la page en fonction des données XML ou JSON récupérées par la requête Ajax.



Exemple

```
$.ajax( {
  type: 'POST',
  url: 'requeteAjax.php',
  data: {
    'action': 'insert',
    'nom': '.....',
    'adresse': '.....'
  },
  // fonction de rappel (ou callback)
  success: function(retour) {
    alert('ajouté avec succès');
  },
  error: function(retour) {
    alert('erreur !');
  },
  dataType: 'JSON'
}
```

```
);
```

Les raccourcis \$.post et \$.get

Ces deux fonctions permettent de raccourcir la syntaxe de la fonction **\$.ajax()**

```
$.get(  
    'url?param1=value1&param2=value2.....',  
    function(retour) { ... } // fonction de callback  
);
```

```
$.post(  
    'url',  
    {param1: 'value1', param2: 'value2', .....},  
    function(retour) { ... } // fonction de callback  
);
```

5. UTILISER JS ET JQUERY POUR EXPLOITER LE RETOUR JSON D'UN APPEL AJAX

A. Produire du JSON depuis PHP

Transformer un objet en une chaîne de caractères JSON

```
// 1ier cas : en utilisant la fonction json_encode :  
json_encode($objet);  
  
// 2ieme cas : en définissant la méthode toString() dans la classe concernée :  
class Client {  
    public $id = 0, $nom, $adresse, $tel;  
  
    public function toString() {  
        return '{"id":'$this->id','nom':'$this->nom','adresse':'$this->adresse','tel':'$this->tel'}';  
    }  
}
```

Retourner un « JSON response » pour un « Ajax request »

```
// appel ajax depuis javascript :  
$.ajax( {  
    type: 'POST',  
    url: 'requetesAjaxClients.php',  
    data: { 'crud_action': 'insert', ...  
  
// fichier 'requetesAjaxClients.php'  
$action = $_POST["crud_action"];  
if($action == 'insert') {  
    $objet = new Client(0, $_POST["nom"], $_POST["adr"], $_POST["tel"]);  
    $objet = $controls->client_insert($objet);  
    echo json_encode($objet);  
}
```

B. Exemples JS et JQuery pour modifier le DOM de la page PHP

Définir l'action d'un nœud (bouton, combobox, ...)
Vue php
<code><button onclick="insertClient()" class="btn btn-success">AJOUTER</button></code>
Code javascript ou jquery
<pre>function insertClient() { \$.ajax({ type: 'POST', url: 'requetesAjaxClients.php', data: { 'action': 'insert', 'nom': '.....', 'adresse': '.....' }, success: function(retour) { alert('client ajouté avec succès'); }, error: function(retour) { alert('erreur !'); }, dataType: 'JSON' }); }</pre>
Récupérer un nœud (une zone de texte, un td, un tr, un combobox, ...) par son ID
Vue php
<pre>foreach (\$listeClients as \$value) { \$client_id = \$value->getId(); \$client_nom = \$value->getNom(); \$client_adr = \$value->getAdresse(); \$client_tel = \$value->getTel(); \$selected_client = \$value; ?> <tr id="<?php echo \$client_id; ?>"> <td> <?php echo \$client_id; ?> </td> <td id="<?php echo 'nom' . \$client_id; ?>"> <?php echo \$client_nom; ?> </td> <td id="<?php echo 'adr' . \$client_id; ?>">?> </tr> }</pre>
Code javascript ou jquery
<pre>function showClient(idclient) { // javascript var nom = document.getElementById('nom' + idclient); }</pre>

```
// jquery
var nom = $('#nom' + idclient);
...
}
```

Récupérer / modifier la valeur d'un nœud (une zone de texte, un td, un combobox, ...)

Vue php

```
</td>
<td id="<?php echo 'nom' . $client_id; ?>">
    <?php echo $client_nom; ?>
</td>
```

Code javascript ou jquery

```
// cas d'une zone de texte ou un composant d'entrée
var value = $('#identifiant').val(); // recuperer
$('#identifiant').val(nouvelle_valeur); // modifier

// cas d'un composant d'affichage - ex. : un td ou un span ou un div ...
var value = $('#identifiant').html(); // recuperer
$('#identifiant').html(nouvelle_valeur); // modifier

// EXEMPLE :
function editClient(idclient) {
    // 1iere partie :
    // Appel ajax pour modifier le client dans la base de données
    $.ajax({
        type: 'POST',
        url: 'requetesAjaxClients.php',
        data: {
            'action': 'update',
            'nom': $('#update_client_nom'),
            'adresse': $('#update_client_adr')
        },
        success: function(retour) {
            // 2ieme partie :
            // Modifier les td qui affichent les infos du client
            // javascript
            var nom = document.getElementById('nom' + idclient);
            nom.innerHTML = nouvelle_valeur;
            // jquery
            $('#nom' + idclient).html(nouvelle_valeur);
            .....
        },
    },
```



```

    error: function(retour) {
        alert('erreur !');
    },
    dataType: 'JSON'
}
);
}

```

Supprimer une ligne (ayant un ID) d'une table

Vue php

```

foreach ($listeClients as $value) {
    $client_id = $value->getId();
    $client_nom = $value->getNom();
    $client_adr = $value->getAdresse();
    $client_tel = $value->getTel();
    $selected_client = $value;
    ?>
    <tr id="<?php echo $client_id; ?>">
        <td>

```

Code javascript ou jquery

```

function deleteClient(idclient) {
    // 1iere partie :
    // Appel ajax pour modifier Le client dans La base de données
    $.ajax({
        type: 'POST',
        url: 'requetesAjaxClients.php',
        data: {
            'action': 'delete', 'id': idclient
        },
        success: function(retour) {
            // 2ieme partie :
            // Supprimer La Ligne du client
            // jquery
            $('#'+ idclient).remove();
        },
        error: function(retour) {
            alert('erreur !');
        },
        dataType: 'JSON'
    }
);
}

```

Ajouter une ligne dans un TABLE (ayant un ID)

Vue php

```

<tr id="<?php echo $client_id; ?>">
  <td><?php echo $client_id; ?></td>
  <td id="<?php echo 'nom' . $client_id; ?>"><?php echo $client_nom; ?></td>
  <td id="<?php echo 'adr' . $client_id; ?>"><?php echo $client_adr; ?></td>
  <td id="<?php echo 'tel' . $client_id; ?>"><?php echo $client_tel; ?></td>
  <td>
    <button onclick="showCommandesClient(<?php echo $client_id; ?>)"
      class="btn btn-success btn-block">
      commandes
    </button>
  </td>
  <td>
    <button
      type="button" data-toggle="modal" data-target="#updateModal"
      onclick="showEditClient(<?php echo $value->toString(); ?>)"
      class="btn btn-success btn-block">
      editor
    </button>
  </td>
  <td>
    <button
      type="button" data-toggle="modal" data-target="#deleteModal"

```

Code javascript ou jquery

```

var table_liste_clients = document.getElementById('liste_clients');
// creation de la nouvelle ligne (tr) et definition de son attr "id"
var new_row = document.createElement('tr');
new_row.setAttribute('id', jsonResult['id']);

// creation des td de la nouvelle tr et leurs identifiants et contenus
var td_id = document.createElement('td');
td_id.innerHTML = jsonResult['id'];
var td_nom = document.createElement('td');
td_nom.setAttribute('id', 'nom' + jsonResult['id']);
td_nom.innerHTML = jsonResult['nom'];
var td_adr = document.createElement('td');
td_adr.setAttribute('id', 'adr' + jsonResult['id']);
td_adr.innerHTML = jsonResult['adresse'];
var td_tel = document.createElement('td');
td_tel.setAttribute('id', 'tel' + jsonResult['id']);
td_tel.innerHTML = jsonResult['tel'];

// creation des td qui contiendront les boutons
var td_commandes = document.createElement('td');
td_commandes.innerHTML =
  '<button onclick="showCommandesClient(' + jsonResult['id'] + ')" class="btn btn-success btn-block">commandes</button>

```

```
// ajouter les td dans la tr
new_row.appendChild(td_id);
new_row.appendChild(td_nom);
new_row.appendChild(td_adr);
new_row.appendChild(td_tel);
new_row.appendChild(td_commandes);
new_row.appendChild(td_edit);
new_row.appendChild(td_supprim);

// ajouter la tr dans la table
table_liste_clients.appendChild(new_row);
```

Ajouter une ligne dans un COMBOBOX (ayant un ID)

Vue php

```
<body>
<select id="selectVille">
  <?php
    // put your code here
    $villes = ['marrakech', 'casablanca', 'agadir', 'rabat'];
    foreach($villes as $ville) {
  ?>
  <option><?php echo $ville;?></option>
  <?php
    }
  ?>
</select>
<button onclick="afficherQuartiers()">OK</button>
<br>
<select id="selectQuartier">
  <option>choisissez une ville</option>
</select>
</body>
<script src="js/jquery-3.3.1.js"></script>
<script src="js/modeles.js"></script>
```

Code javascript ou jquery

```
function afficherQuartiers() {
  var nom_ville = $('#selectVille').val();
  $.ajax({
    type: 'POST',
    url: 'controls/AjaxHandler.php',
    data: {
      'action': 'quartiers',
      'ville': nom_ville
    },
    dataType: 'JSON',
    success: function (jsonResult) {
      $('#selectQuartier').empty();
      $.each(
        jsonResult,
        function(indice,valeur){
          var elem = '<option value="" + indice + "">' + valeur + '</option>';
          $('#selectQuartier').append(elem);
        });
    },
    error: function (jsonResult) {
      alert('erreur : ' + jsonResult);
    }
  });
}
```

```
// AjaxHandler.php
if($action == 'quartiers') {
  $ville = $_POST["ville"];
  $quartiers = array();
  if($ville == 'rabat') {
    array_push($quartiers, 'hassan');
    array_push($quartiers, 'agdal');
    array_push($quartiers, 'hay riad');
  }
  echo json_encode($quartiers);
}
```

A screenshot of a software interface showing a dropdown menu. The menu is open, displaying a list of names: 'hassan', 'hassan', 'agdal', and 'hay riad'. The first 'hassan' entry is highlighted in blue, indicating it is the selected option. Above the menu, the text 'rabat' is visible, and to the right is an 'OK' button.

Fonction \$.each(liste, une_fonction(index,element){ ... })

Sert à parcourir toute sorte de listes d'éléments, pour exécuter en boucle la fonction passée en 2ieme paramètre. La fonction passée en 2ieme paramètre doit prendre 2 paramètres qui seront initialisés par **\$.each** comme ceci :

- Le premier sera initialisé par chaque indice trouvé dans la liste,
- Le deuxième sera initialisé par chaque élément trouvé dans la liste,

```
$.each(liste, // dans notre exemple : la liste json des noms de villes
  function(indice,element) {
    // traitement adéquat pour chaque élément de la liste
  }
);
```

Afficher une boîte de dialogue modale Bootstrap depuis un bouton html

Vue php

```
<button type="button" data-toggle="modal" data-target="#nom_fenetre_modale" onclick=".....">
  texte bouton
</button>
```

```
<button
    type="button" data-toggle="modal" data-target="#updateModal"
    onclick="showEditClient(<?php echo $value->toString(); ?>)"
    class="btn btn-success btn-block">
        editor
</button>
```

```
<div class="modal fade" id="nom_fenetre" role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">un titre</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">

      </div>
      <div class="modal-footer">
        <button type="button" data-dismiss="modal">Annuler</button>
        <button type="button" onclick="..." data-dismiss="modal">
          Un autre bouton
        </button>
      </div>
    </div>
  </div>
</div>
```