Ecole Nationale des Sciences Appliquées
Al Hoceima

Support de Cours

Programmation Web en PHP

Première Année Génie Informatique

Module : Programmation Web

Version 28/05/2021

Enseignant : Tarik BOUDAA

Table des matières

1-Les variables :

1.1. Déclaration et initialisation des variables :

Syntaxe : $Nom_var = ...

Exemples :

```
$t="Test"; //Variable de type chaîne de caractère contenant « Test »
echo $t; //Afficher sur la page. On peut également utiliser la fonction
print
echo '<br>'; //On peut afficher du HTML  (Retour à ligne)
echo '<p>Bonjour</p>'; //On peut afficher du HTML  (Retour à ligne)

// On PHP la variable pourra changer le type en fonction de la valeur qu'on
//lui affecte.
$i = 1;
echo $i;
//Incrémentation
$i++;
//décrémentation
$i-- ;
echo $i;
$a = 'Faire l\'exemple ';
echo $a;
$tes = "$a test";
//Concaténation
$c = "Les deux variables".$a." et ".$b. " concatinées";
echo '<br>';
```

Remarques concernant les chaînes de caractères :

On peut utiliser guillemets doubles (" …") ou Guillemets simples (' …'). Comme dans :

```
$a = 'je suis un test';
$b = "Je suis un test";
```

La différence est que lorsqu'on utilise guillemets doubles si des variables sont insérées à l'intérieur de la chaîne de caractères elles seront évaluées. Par exemple :

```
$age = 9 ;
$a = 'Age :  $age ';  ==> affiche Age :  $age
$b = "Age :  $age"; ==> affiche Age :  9
```

Une autre remarque :

$a = 'l'exemple '; ne se compilera pas ➔ correction : $a = 'l\'exemple ';

$a = 'l"exemple '; se compilera

$a = "l"exemple "; ne se compilera pas ➔ correction : "l\"exemple ";

$a = "l'exemple "; se compilera

1.2. Types simples :

```
$i = 1; //entier
$i=0.01; //flotant
$i="Boudaa"; //String
$i='Boudaa';//String
$i = TRUE; //boolean
```

1.3. Types composés :

Les types composés : Tableau, objet

1.4. Types spéciaux :

- NULL
- resource : (*https://www.php.net/manual/fr/language.types.resource.php*)

1.5. Déterminer le type d'une variable avec la fonction gettype

Exemple :

```
$t='BOUDAA';
$f=1.
$d=10
echo "Type est ". gettype($t);
echo '<br>';
echo "Type est ". gettype($f);
echo '<br>';
echo "Type est ". gettype($d);
```

Le résultat de l'exemple ci-dessus est :

string
double
integer

1.6. Vérifier le type d'une variable

Les fonctions suivantes permettent de vérifier si une variable est d'un type précis

- *is_integer($var) ou is_int($var)*
- *is_double($var)*
- *is_string($var)*
- *is_bool($var)*
- *is_array($var)*
- *is_object($var)*
- *is_resource($var)*
- *is_null($var)*

Exemples :

```
$i=22;
if(is_int($i)){
    echo "$i est un entier";
}
```

```php
else{
    echo "N'est pas un Entier";
}
echo '<br>';
```

Le résultat de l'exemple ci-dessus est :

22 est un entier

1.7. Affectation par référence

```php
$k = 1;
$j= &$k; //$j contient maintenant la référence sur $k. càd que $k et $j
//référence le  même emplacement
echo $k; // affiche de 1
echo '<br>'; // retour à la ligne
echo $j;  // $k et $j référence le  même emplacement donc 1
echo '<br>';
$j=10; //On change la valeur de $j
echo $k; // $k référence le même emplacement que $j donc affichage de 10
```

Le résultat de l'exemple ci-dessus est :

1
1
10

1.8. La conversion de type

$result = (type_désiré) $mavar;

Exemple

```php
$var="3.52 kilomètres";
$var2 = (double) $var;
echo "\$var2= ",$var2;//affiche "$var2=3.52"
$var3 = (integer) $var2;
echo "\$var3= ",$var3;//affiche "$var3=3"
$var4 = (boolean) $var3;
echo "\$var4= ",$var4;//affiche "$var4=1" soit la valeur true
```

1.9. Les variables superglobales

- $GLOBALS
- $_SERVER
- $_GET
- $_POST
- $_FILES
- $_COOKIE
- $_SESSION
- $_REQUEST
- $_ENV

Voir : *https://www.php.net/manual/fr/language.variables.superglobals.php*

2- Constantes :

2.1. Définition des constantes

Deux manières de déclarer les constantes avec *define* ou le mot clé *const* :

```
echo '<br>';
define ("PI", 3.14);
echo PI;
echo '<br>';
echo pi;
echo '<br>';
define ("PI", 3.14, TRUE);
echo pi * 2; // p à la puisse 2
echo '<br>';
```

Nous pouvons également définir des constantes à l'aide du mot-clé ***const***

```
const PI= 3.14;
echo PI;
```

Une constante peut aussi être un tableau en écrivant par exemple :

```
const NOM = ['BOUDAA','KARIMI'];
echo NOM[0];
```

ou encore :

```
define('NOM', array('BOUDAA','KARIMI'));
```

2.2. Quelques constantes prédéfinies

```
echo PHP_OS ; //Système d'exploitation exécutant le scripte
echo PHP_VERSION; //Version de PHP installée sur le serveur
```

La liste complète : https://www.php.net/manual/fr/reserved.constants.php

Les constantes qui commencent par __ s'appellent constantes « magiques ». Ces constantes se distinguent des autres puisque leur valeur va changer en fonction de l'endroit dans le script où elles vont être utilisées.

```
echo __FILE__ ; //fichier en cours d'exécution
echo __LINE__; // Numéro de l'instruction qui est en cours d'exécution
echo __DIR__; // Contient le nom du dossier dans lequel est le script en
//cours d'exécution
```

3- Opérateurs :

3.3. Opérateur d'affectation combinée

Similaires au langage C :

Variable1 opération = Variable2 ⇔ Variable1 = Variable1 opération Variable2

avec opération est : +,-,*,**,%...

Exemples :

```
$i+= 5; // $i = $i + 5
$s='test';
$s.='OK';// càd : $s = $s . 'OK' ==>  $s = 'testOK'
```

3.4. Opérateur puissance

```
$s = 4;
echo $s ** 2;
echo '<br>';
$s**=2; //opérateur puissance combiné
echo $s;
```

3.5. Les autres opérateurs : +,-,++,--,%,...

(Similaires au langage C)

3.6. Opérateur ==

Opérateur == : Teste l'égalité de deux valeurs dans prendre en compte le type. L'expression $a == $b vaut TRUE si la valeur de $a est égale à celle de $b et FALSE dans le cas contraire :

```
$a = 345;
$b = "345";
$c = ($a==$b);
```

Ici $c est un booléen qui vaut TRUE car dans un contexte de comparaison numérique, la chaîne "345" est évaluée comme le nombre 345. Si $b="345 KM" nous obtenons le même résultat.

3.7. Opérateur != ou <>

Teste l'inégalité de deux valeurs.

3.8. Opérateurs === et !==

Opérateur === : Teste l'identité des valeurs et des types de deux expressions.

!== : Teste la non-identité de deux expressions.

3.9. Opérateur <=>

Opérateur <=> : Avec $a<=>$b, retourne -1, 0 ou 1 respectivement si $a<$b, $a=$b ou $a>$b ($a et $b peuvent être des chaînes de caractères).

3.10. Opérateurs < , >, <=, =>

(Analogues au langage C)

3.11. Les opérateurs logiques :

- || (ou or) : Ou logique
- xor : Ou exclusive
- && (ou and) : Et logique

- ! : Négation

Préférez généralement || et && au lieu de « or » et « and »
En effet, y a une différence dans leur priorité

$e = false || true; ➔ équivalente à ($e = (false || true))

$f = false or true; ➔ équivalente à (($f = false) or true)

4- Les tableaux

4.1. Tableaux simples

```
$a =array();
$b=array(1,2,3);
echo $b[1];
```

Autre syntaxe

```
$aa[0]='X';
$aa[1]=12;
echo '<br>', $aa[0],'<br>', $aa[1];
```

Nombre d'élément d'un tableau : fonction count

```
echo count($aa);
```

4.2. Tableaux associatifs :

```
$tab['Nador'] ="Ville";
$tab['Maroc']='Afrique';
$tab['Java_version']=11;
echo '<br>', $tab['Nador'],'<br>', $tab['Maroc'], '<br>',
$tab['Java_version'];
```

Le résultat de l'exemple ci-dessus est :

Ville
Afrique
11

Autre syntaxe :

```
$tab =array("nador"=>"Ville","Maroc"=>"Afrique");
```

✓ Les clés des tableaux associatifs étant sensibles à la casse
✓ De plus, les chaînes définissant les clés ne doivent pas comporter d'espaces.

4.3. Tableaux multidimensionnels

```
$tabmulti=array(
    array("ligne 0-colonne 0","ligne 0-colonne 1","ligne 0-colonne 2"),
    array("ligne 1-colonne 0","ligne 1-colonne 1","ligne 1-colonne 2"),
    array("ligne 2-colonne 0","ligne 2-colonne 1","ligne 2-colonne 2"),
    array("ligne 3-colonne 0","ligne 3-colonne 1","ligne 3-colonne 2")
);
```

Ou encore :

```
$tabmulti=[ ["ligne 0-colonne 0","ligne 0-colonne 1","ligne 0-colonne 2"],
    ["ligne 1-colonne 0","ligne 1-colonne 1","ligne 1-colonne 2"],
    ["ligne 2-colonne 0","ligne 2-colonne 1","ligne 2-colonne 2"],
    ["ligne 3-colonne 0","ligne 3-colonne 1","ligne 3-colonne 2"]];
echo '<br>';
echo $a = $tabmulti[2][1];
echo '<br>';
```

Le résultat de l'exemple ci-dessus est :

ligne 2-colonne 1

5- Instructions de contrôle

5.1. L'instruction If

```
if (condition) {
// Une ou plusieurs instructions. Si une seule instruction on peut
 //enlever les {}
}
```

5.2. If else

```
if (condition) {
    // Une ou plusieurs instructions
} else {
    // Une ou plusieurs instructions
}
```

5.3. If … elseif ….else

```
if(condition1){
    // Une ou plusieurs instructions
}elseif(condition2){
    // Une ou plusieurs instructions
}
...éventuellement d'autre blocs elseif
else{
    // Une ou plusieurs instructions
}
```

5.4. Opérateur ternaire :

$var = expression ? valeur1 : valeur2 (similaire au langage C)

5.5. Les instructions break & continue

Sont les mêmes que celles du langage C

5.6. L'instruction switch…case

```
switch(expression)
{
    case valeur1:
```

```
        bloc d’instructions 1;
        break;
    case valeur2:
        bloc d’instructions 2;
        break;
        ......................
    case valeurN:
            bloc d’instructions N;
            break;
    default:
        bloc d’instructions par défaut;
        break;
}
```

5.7. L’instruction for

```
for(expression1; expression2; expression3)
{
    //     instruction ou bloc;
}
```

Exemple :

```
for($i=0; $i< count($tab); $i++)
{
    echo $tab[$i] ;
}
```

5.8. L’instruction while

```
while(expression)
{
    //Bloc d'instructions à répéter
}
```

5.9. L’instruction do.. while

```
do {
    // bloc d'instructions
} while (expression);
```

5.10. L’instruction foreach

Parcourir un tableau avec *foreach*

```
foreach($tableau as $valeur)
{
    //bloc utilisant la valeur de l'élément courant
}
```

Exemple :

```
$tab =[1,2,3];
foreach($tab as $cle=>$valeur)
{
    echo  $valeur , '<br>';
}
```

Le résultat de l'exemple ci-dessus est :

1
2
3

Parcourir un tableau associatif avec *foreach*

```
foreach($tableau as $cle=>$valeur)
{
    //bloc utilisant la valeur et la clé de l'élément courant
}
```

Exemple :

```
$tab =array("nador"=>"Ville","Maroc"=>"Afrique");
foreach($tab as $cle=>$valeur)
{
    echo  $cle ."=>". $valeur , '<br>';
}
```

Le résultat de l'exemple ci-dessus est :

nador=>Ville
Maroc=>Afrique

6- Contrôler l'état d'une variable

6.1. Fonction boolean isset($var)

Détermine si une variable est déclarée et est différente de null

```
if(isset($var11)){
    echo 'Variable $var11 existe';
}
else{
    echo 'Variable $var11 n\'existe pas';
}
echo '<br>';
$var12;
if(isset($var12)){
    echo 'Variable $var12 existe';
}
else{
    echo 'Variable $var12 n\'existe pas';
}
echo '<br>';
$var12 = 1;
if(isset($var12)){
    echo 'Variable $var12 existe';
}
else{
    echo 'Variable $var12 n\'existe pas';
}
```

Le résultat de l'exemple ci-dessus est :

Variable $var11 n'existe pas
Variable $var12 n'existe pas
Variable $var12 existe

6.2. Fonction boolean empty(expression)

Détermine si une variable est vide. Ainsi, elle retourne la valeur TRUE si l'expression passée en paramètre n'est pas initialisée, a une des valeurs suivantes : 0, NULL, FALSE, la chaîne "0", ou est un tableau vide, et la valeur FALSE si elle a une quelconque autre valeur.

```
echo '<br>';
$var13;
if (empty($var13)) {
    echo 'Variable $var13 est vide';
} else {
    echo 'Variable $var13 n\'est pas vide';
}
echo '<br>';
$var13 = 1;
if (empty($var13)) {
    echo 'Variable $var13 est vide';
} else {
    echo 'Variable $var13 n\'est pas vide';
}
```

Le résultat de l'exemple ci-dessus est :

Variable $var13 est vide
Variable $var13 n'est pas vide

7-Les fonctions

Il existe deux façons de passer une variable à une fonction en PHP :

- Par valeur (ce qui est le comportement par défaut)
- Par référence en utilisant le symbole & devant le nom de la variable.

7.1. Passage par valeurs

```
function somme($x, $y)
{
    $s = $x+ $y;
    return $s ; //
}

$s= somme(3, 1);
echo 'Somme = ' . $s;
```

7.2. Passage par référence

```
function echanger(&$x, &$y)
{
```

```
    $aide = $x;
    $x= $y;
    $y = $aide;
}

$a = 1;
$b= 2;
echanger($a, $b);
echo "a=$a et b=$b";
```

Le résultat de l'exemple ci-dessus est :

a=2 et b=1

Nous pouvons également retourner une référence

```
function &test(){
    $b = 2;
    return $b;
}

$a = &test();
echo $a;
```

7.3. Portée des variables

```
Une variable peut être globale ou locale
$a = 1; /* portée globale */

function test()
{
    $a = 3;
    echo $a; /* portée locale */
}

echo $a; // Contient la valeur 1
test(); //  Va afficher 3
```

7.4. Accéder aux variables globales

Première méthode : Avec le mot clé global

```
$a = 1;

$b = 2;

function somme() {

    global $a, $b;

    $b = $a + $b;

}

somme();

echo $b; //affiche 3
```

Deuxième méthode : avec la table $GLOBALS

$GLOBALS est un tableau associatif contenant des références sur toutes les variables disponibles dans un contexte global.

Exemple :

```
$a = 1;

$b = 2;

function somme() {

    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];

}

somme();

echo $b; //affiche 3
```

7.5. Récursivité

En PHP comme en langage C vous pouvez définir des fonctions récursives :

```
function factoriel($n){
    if($n==0){
        return 1;
    }
    return $n * factoriel($n-1);
}
echo factoriel(4); //affiche 24
```

7.6. Fonction prédéfinie

PHP dispose de plusieurs fonctions prédéfinies, nous découvrirons quelques fonctions dans les TPs.

8- Gestion des exceptions :

```
try
{
    // Code à surveiller
    if(erreur prévue)
    {
        throw new Exception();
    }
    else{
        // Résultat;
    }
}
catch(Exception $except)
{
    // Gestion de l'erreur
}
finally
```

```
    {
        //Code qui sera forcément exécuté
    }
```

Exemple :

```
$a = 1;
$b = 0;
try
{     // Code à surveiller
    if($b==0)
    {
        //cette instruction arrête l'exécution
        throw new Exception("Erreur division par 0");
    }
    echo $b;
}
catch(Exception $e)
{
    echo $e->getMessage();
}
finally
{
    //Code qui sera forcément exécuté
    echo "<br> FIN";
}
```

Le résultat de l'exemple ci-dessus est :

Erreur division par 0
FIN

9- Rappels sur HTTP :

9.1. Communication entre le client et le serveur via http :

Soit l'URL: *http://serveur.ensah.ma:80/*

Le navigateur l'interprète de la façon suivante :

- http:// : Utiliser le protocole http
- Serveur.ensah.ma : Contacter le serveur de nom d'hôte serveur.ensah.ma
- :80 : Se connecter sur le port 80. (C'est le port par défaut pour http)
- / : Tout ce qui suit est considéré comme le chemin du document dans le serveur.

Le message envoyé au serveur est:

```
GET / HTTP1.1                                              1
Accept: image/gif, image/x-xbitmap, image/jpeg, */*        2
Accept-Language: en-us                                     3
Accept-Encoding: gzip, deflate                             4
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT) 5
Host: serveur.ensah.ma                                     6
Connection: Keep-Alive                                     7
```

La réponse du serveur

```
HTTP1.1 200 OK                                  1   ┐
Date: Mon, 15 Jul 2002 11:50:20 GMT             2   │
Server: Tomcat-Apache/4.0.3 Win NT              3   │
Last-Modified: Fri, 04 oct 2001 14:05:22 GMT    4   │  En-tête
Etag: "2f8ce-732-351e1ad6"                      5   │
Accept-Ranges: bytes                            6   │
Content-length: 212                             7   │
Connection: close                               8   │
Content-type: text/html                         9   ┘
                                                10      (Ligne vide)

<html>                                                  ┐
<head><title>Hello</title></head>                       │
<body>                                                  │  Document
   ...                                                  │
</body>                                                 │
</html>                                                 ┘
```

9.2. Méthodes du client (verbes http)

Méthode GET :

Est utilisée pour la récupération (lecture) d'une ressource sur le serveur. Ce peut être : Le contenu d'un fichier statique (html) l'invocation d'un programme (un script php, une servlet Java,...) qui va générer une réponse.

Caractéristiques :

- ✓ La chaîne de requête (*query string*) de paires "nom/valeur" est envoyée dans l'URL d'une requête GET.
- ✓ Les requêtes GET peuvent être mises en cache (can be cached)
- ✓ Les requêtes GET restent dans l'historique du navigateur
- ✓ Les requêtes GET peuvent être mises en signet (can be bookmarked)

- ✓ Les requêtes GET ne doivent jamais être utilisées pour traiter des données sensibles
- ✓ Les requêtes GET ont des restrictions de taille de données à envoyer
- ✓ Les requêtes GET ont des restrictions sur le type de données (Seuls les caractères ASCII autorisés)
- ✓ Les requêtes GET sont uniquement utilisées pour demander des données (pas pour les modifier)

POST :

Cette méthode est utilisée pour transmettre des données en vue d'un traitement à une ressource sur le serveur (le plus souvent depuis un formulaire HTML). Les données ne sont pas visibles dans l'URL. On peut envoyer du binaire (par exemple un fichier en pièce jointe, donc pas de restriction sur le type de données à envoyer).

Autres caractéristiques :

- ✓ Les requêtes POST n'ont aucune restriction sur la longueur des données à envoyer
- ✓ Les requêtes POST ne sont jamais mises en cache
- ✓ Les requêtes POST ne restent pas dans l'historique du navigateur
- ✓ Les requêtes POST ne peuvent pas être mises en signet (cannot be bookmarked)

HEAD :

Est presque identique à GET, mais sans le corps de réponse. Généralement utilisée pour demander d'info sur un document. Par exemple : la date du document afin de savoir s'il est déjà dans le cache)

Les requêtes HEAD sont utiles pour vérifier ce qu'une requête GET retournera avant de faire une demande GET - comme avant de télécharger un fichier volumineux ou un corps de réponse.

PUT :

Est utilisée pour envoyer des données à un serveur afin de créer / mettre à jour une ressource.

La différence entre POST et PUT est que les requêtes PUT sont *idempotentes*. C'est-à-dire qu'appeler plusieurs fois la même demande PUT produira toujours le même résultat. En revanche, l'appel répété d'une requête POST a pour effet secondaire de créer la même ressource plusieurs fois.

DELETE :

Cette méthode permet de supprimer sur le serveur la ressource spécifiée

OPTIONS :

La méthode OPTIONS est utilisée pour décrire les options de communications avec la ressource visée.

TRACE :

La méthode TRACE réalise un message de test aller/retour en suivant le chemin de la ressource visée.

PATCH :

La méthode PATCH est utilisée pour appliquer des modifications partielles à une ressource

9.3. Méthodes sûres et idempotentes

- Les méthodes sûres (*safe methods*) sont des méthodes HTTP qui ne modifient pas les ressources.
- Une méthode HTTP idempotente (*Idempotent method*) est une méthode HTTP qui peut être appelée plusieurs fois sans aboutir à des résultats différents.

HTTP Method	Idempotent	Safe
OPTIONS	yes	yes
GET	yes	yes
HEAD	yes	yes
PUT	yes	no
POST	no	no
DELETE	yes	no
PATCH	no	no
TRACE	yes	yes

10- Gestion des requêtes POST et GET :

10.1. Envoi des données par un GET :

Si nous invoquons le lien suivant via la barre d'adresse du navigateur ou un lien dans une page web, une requête de type GET sera générée et envoyer au serveur au scripte newsDetails.php:

https://ensah.ma//public/**newsDetails.php?idNews=1189&catagorie=divers**

Au niveau serveur, dans le scripte newsDetails.php, nous pouvons récupérer les valeurs des données envoyées comme paramètres de requête dans la variable superglobale $_GET qu'est un tableau associatif. Par exemple :

```
echo $_GET[' idNews'] ;
echo $_GET[' catagorie] ;
```

10.2. Envoi des données par un POST :

La validation du formulaire ci-dessous engendrera une requête de type POST qui va invoquer le scripte action.php:

```
<form action="action.php" method="POST">
    <label>First name:</label><br>
    <input type="text"  name="fname" ><br>
    <label>Last name:</label><br>
    <input type="text" name="lname" ><br><br>
    <input type="submit" value="Submit">
</form>
```

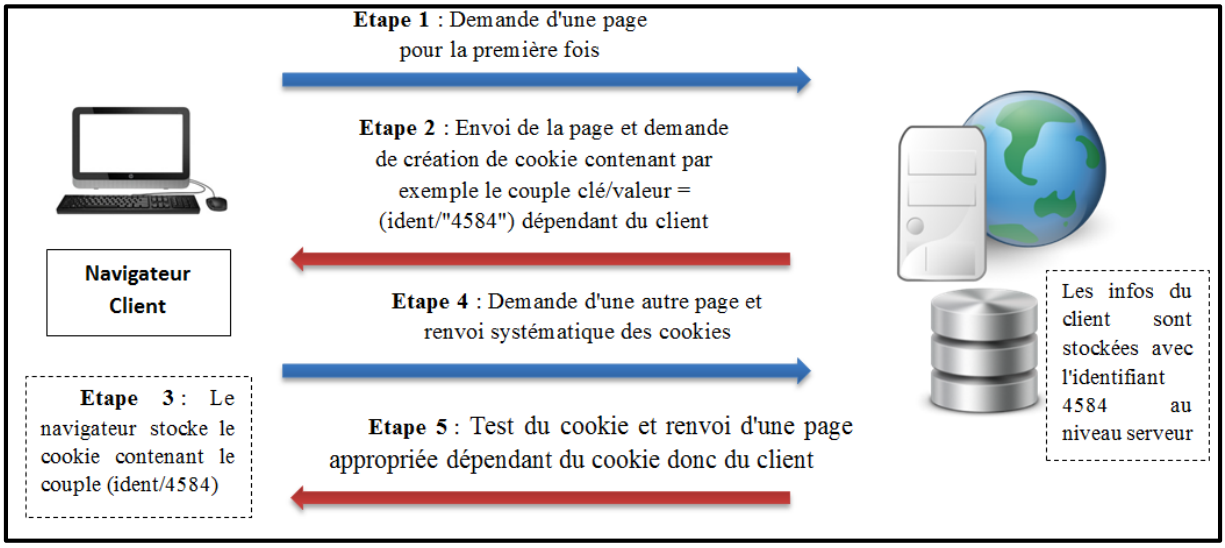
Au niveau serveur, dans le scripte action.php, nous pouvons récupérer les valeurs des données envoyées dans la variable superglobale $_POST qu'est un tableau associatif. Par exemple :

```
echo $_POST[' fname] ;
echo $_POST[' lname] ;
```

11- Gestion des Cookies

11.1. Notion de cookie

C'est une information (couple nom/valeur) généralement écrite par un serveur sur le poste du client le consultant



11.2. Travailler avec les cookies en PHP :

- Pour envoyer un cookie au client on utilise setcookie (https://www.php.net/manual/fr/function.setcookie.php).

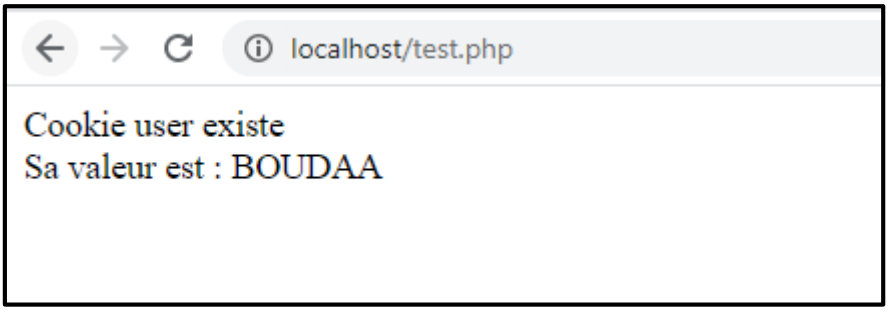
- Une fois que les cookies ont été placés, ils seront accessible lors du prochain chargement de page dans le tableau $_COOKIE. Les valeurs des cookies peuvent aussi exister dans la variable $_REQUEST.

- **Remarque:** la fonction setcookie () doit apparaître AVANT la balise <html>.

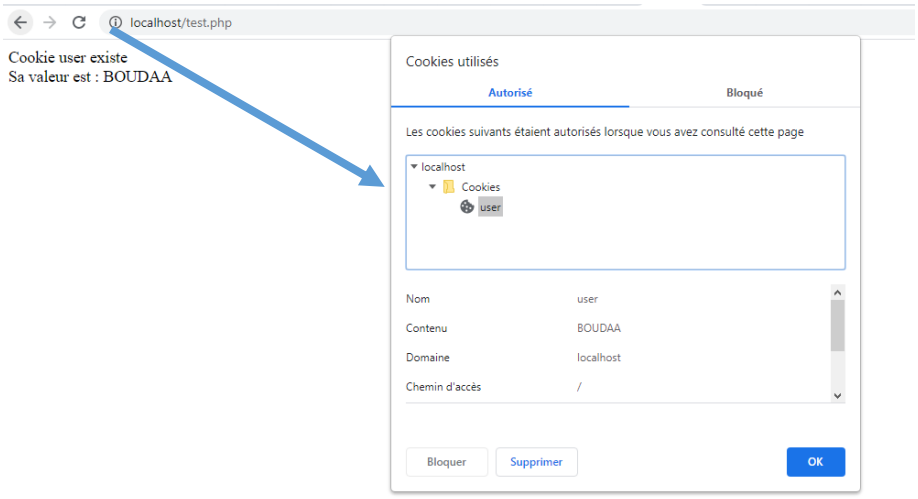
```php
<?php
$cookie_name = "user";
$cookie_value = "BOUDAA";
// expire dans 30* 86400 ms = 30* 1 day = 1 mois
setcookie($cookie_name, $cookie_value, time() + (86400 * 30));
?>
<html>
<body>

<?php
if (! isset($_COOKIE['user'])) {
    echo "Cookie ayant le nom  $cookie_name  n'existe pas !";
} else {
    echo "Cookie $cookie_name  existe <br>";
    echo "Sa valeur est : " . $_COOKIE['user'];
}
?>
</body>
</html>
```

Pour la première exécution :

Pour la deuxième exécution :



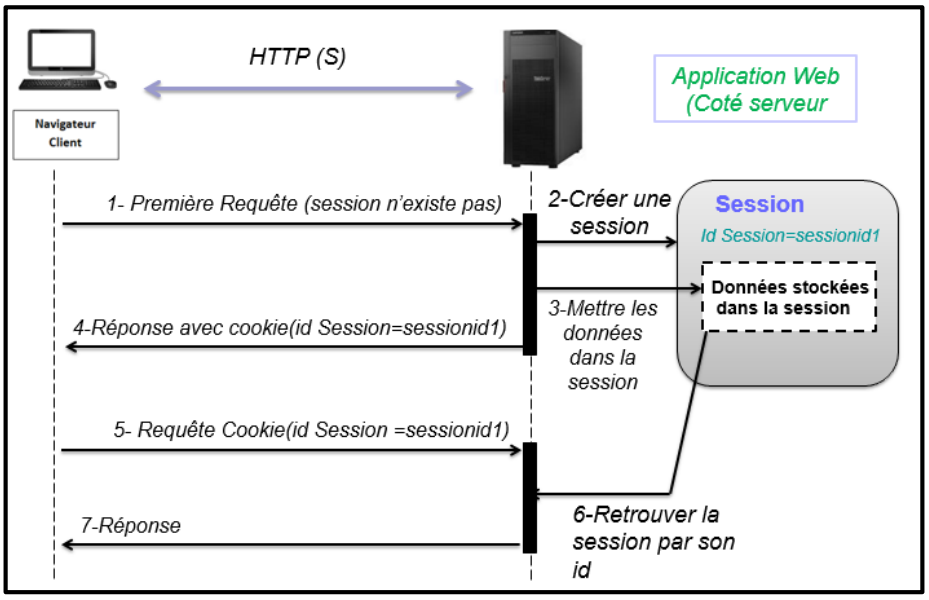
Vous pouvez visualiser le contenu du cookie sur le navigateur :



12- Gestion de la session

12.1. Notion de session http :

- Une session permet de suivre un utilisateur lors de sa navigation sur une application web, d'une ressource à une autre (d'une page en page par exemple). Elle peut être maintenue pendant un temps assez long, même de plusieurs jours, afin d'identifier un « utilisateur » qui reviendrait sur l'application web.
- La notion de session crée donc une notion de persistance, durant une certaine période fixée à l'avance. Au-delà d'une certaine durée d'activité, une session HTTP expire, et toutes les informations qui y sont attachées expirent.



12.2. Travailler avec la session en PHP :

```
<?php
// On démarre la session avant le code HTML
```

```php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
 // Mettre des données dans la session. On utilise la variable
superglobale    $_SESSION     //qu'est un tableau associatif
     $_SESSION["favcolor"] = "green";
     $_SESSION["favanimal"] = "cat";
 ?>
</body>
</html>
```

Remarque : La fonction *session_start ()* doit être la toute première chose dans votre document. Avant toute balise HTML.

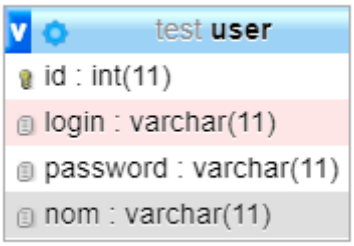
13- Accéder à la base de données avec PDO :

13.1. Se connecter à la base de données

```php
$pdo_options [PDO::ATTR_ERRMODE] = PDO::ERRMODE_EXCEPTION;
$bdd = new PDO ( $dburl, $logindb, $passwordb, $pdo_options );
$bdd->exec ( "SET CHARACTER SET utf8" );

//fermer la connexion
$bdd = null;
```

13.2. Exécution d'un SELECT

Soir la table suivante :




```php
/* Création d'un objet PDOStatement */
$stmt = $dbh->prepare('SELECT *  FROM User');

/* Exécute la requête */
$stmt->execute();

/* Récupération de la première ligne uniquement depuis le résultat */
$stmt->fetch();

/* Récupération de toutes les lignes */
while($data = $stmt->fetch()){
```

```
    echo $data['nom']  , '<br>';
    echo $data[login]  , '<br>';
    echo $data[password]  , '<br>';
}

//On ferme le curseur
$stmt->closeCursor();
```

13.3. Exécution d'un UPDATE

```
$stmt = $dbh->prepare('UPDATE User set nom= :n and password = :p where id=
:id');
$stmt->execute(array('n'=>'BOUDAA', ' p '=> '123' , 'id'=> 1) );
```

13.4. Exécution d'un INSERT

```
$stmt = $dbh->prepare('INSERT into User(nom, nom, password) values ( :n
:p,:id) ');
$stmt->execute(array('n'=>'BOUDAA', ' password '=> '123' , 'id'=> 1) );


//On peut si on veut récupérer l'identifiant de la dernière ligne insérée
echo  $dbh->lastInsertId();
```

Méthode lastInsertId :

lastInsertId — Retourne l'identifiant de la dernière ligne insérée

13.5. Exécution d'un DELETE

```
$stmt = $dbh->prepare('DELETE from User where id= :id');
$stmt->execute(array('id'=>1) );
```

13.6. Transactions

```
try {
  //se connecter
 $dbh = new PDO('mysql:host=localhost;dbname=test', 'username',
'password');

$dbh->beginTransaction();

 //Opérations SQL à exécuter dans la transaction
 //...

$dbh->commit();
} catch(PDOExecption $e) {
    If($dbh != null){
        $dbh->rollback();
    }
    //Autres instructions ….
}
```

13.7. Méthode rowCount

PDOStatement::rowCount: Retourne le nombre de lignes affectées par le dernier appel à la fonction PDOStatement::execute()

```
/* Effacement de toutes les lignes de la table FRUIT */
$del = $dbh->prepare('DELETE FROM fruit');
$del->execute();

/* Retourne le nombre de lignes effacées */
print("Retourne le nombre de lignes effacées :\n");
$count = $del->rowCount();
print("Effacement de $count lignes.\n");
```

14- Les classes

14.1. Définir une classe

```
class Personne
{
    //Les attributs
    private $age = 20;   // L'age d'une personne, par défaut à 20.
    private $ville = 'Al Hoceima'; // Sa ville, par défaut à Al Hoceima.
    private $nom;
    private $prenom;

    //Méthode pour afficher l'age
    public function afficherAge()
    {
        echo ("<p> Age : $this->age </p> ");
    }

    public function ModifierAge($nouveauAge)
    {
        $this->age = $nouveauAge ;
    }
 }
```

14.2. Instanciation et appel des méthodes

```
$p = new Personne; // ou $p = new Personne();

//Appeler les méthodes de l'objet
echo $p->afficherAge();

//On modifie l'age
$p->ModifierAge(50);

//On affiche à nouveau
echo $p->afficherAge();
```

14.3. Les accesseurs et mutateurs (Getters / Setters)

```php
class Personne
{
    private $age = 20;
    // L'age d'une personne, par défaut à 20.
    private $ville = 'Al Hoceima';
    // Sa ville, par défaut à Al Hoceima.
    private $nom;
    private $prenom;

    public function getAge()
    {
        return $this->age;
    }

    public function setAge($age)
    {
        $this->age = $age;
    }

}
```

14.4. Constructeur :

```php
class Personne
{
    private $age;
    private $ville;
    private $nom;
    private $prenom;

    // Constructeur demandant 4 paramètres
    public function __construct($age, $ville, $nom, $prenom)
    {
        // Initialisations
        $this->setAge($age);
        $this->ville = $ville;
        $this->nom = $nom;
        $this->prenom = $prenom;
    }
    public function setAge($age)
    {
        $this->age = $age;
    }
}
```

14.5. Instanciation :

```php
$p = new Personne(20, "Imzouren", "Boudaa", "Mohamed");
```

14.6. Les niveaux de visibilité des propriétés, méthodes et constantes en POO PHP

A partir de PHP 7.1.0 il est possible de définir trois niveaux de visibilité pour les attributs, les méthodes et les constantes. Ceci grâce aux mots clefs *public*, *private* et *protected*.

- Les propriétés, méthodes ou constantes définies avec *public* vont être accessibles depuis l'intérieur ou l'extérieur de la classe.
- Les propriétés, méthodes ou constantes définies avec *private* ne vont être accessibles que depuis l'intérieur de la classe qui les a définies.
- Les propriétés, méthodes ou constantes définies avec *protected* ne vont être accessibles que depuis l'intérieur de la classe qui les a définies ainsi que depuis les classes qui en héritent ou la classe parente.

Il est obligatoire de définir explicitement le niveau de visibilité pour les attributs. Cependant, les méthodes et les constantes on peut ne pas spécifier le niveau de visibilité dans ce cas le niveau de visibilité est le niveau par défaut qu'est *public*.

14.7. Attributs et méthodes statiques

- Les attributs et les méthodes statiques appartiennent à la classe et ne sont pas des éléments de l'instance ou de l'objet. Ainsi ils sont accessibles sans recourir à une instance de classe ou à un objet.
- Les attributs et méthodes statiques ne sont pas accessibles par l'opérateur -> mais plutôt par un autre opérateur qui s'appelle l'opérateur de résolution de portée **::** précédé par le **nom de la classe** dans laquelle ils sont définis.
- Dans la définition d'une classe, « $this » se réfère à l'objet actuel, tandis que « self » se réfère à la classe actuelle.
- « self » n'est pas précédé par « $ » car « self » ne représente pas une variable mais la classe elle-même.
- Il est nécessaire de faire référence à un élément de classe en utilisant « self » et de se référer à un élément d'objet en utilisant « $this ». On utilise $this->var pour les variables non statiques, et self::$var pour les variables statiques et de même pour les méthodes.

Attributs statiques :

```php
<?php

class Personne
{
    private $age;
    private $ville;
    private $nom;

    public static $nombrePersonne = 12;
}

//Accéder à un attribut statique par l'opérateur ::
echo Personne::$nombrePersonne;
```

Méthodes statiques :

```php
<?php

class Personne
{
    private $age;
    private $ville;
    private $nom;

    public static $nombrePersonne = 12;

    public static function  incrementer(){
        Personne::$nombrePersonne++;

        //OU

        self::$nombrePersonne++;
    }

}

//Apppeler une méthode statique par l'opérateur ::

Personne::incrementer();

//Accéder à un attribut statique par l'opérateur ::
echo Personne::$nombrePersonne;
```

Les constantes de classe

On peut définir des constantes à l'intérieur d'une classe, ces constantes sont également accessibles via l'opérateur de résolution de portée.

```php
class Personne
{
    private $age;
    private $ville;
    private $nom;

    //Constante de classe
    const AGE_MAXIMAL = 100;

}

//Accéder à une constante de classe par l'opérateur ::
echo Personne::AGE_MAXIMAL;
```

15- Héritage

15.1. Définition d'une hiérarchie d'héritage

```php
class Personne
{
```

```
}

class Etudiant extends  Personne{

}

class EtudiantENSAH extends  Etudiant{

}
```

15.2. Redéfinir les méthodes

```
class Personne
{

    protected $nom ="BOUDAA";

    protected $cin ="111111";

    public function afficher()
    {
        echo "<p> Nom=" . $this->nom . " cin=" . $this->cin . "</p>";
    }
}

class Etudiant extends Personne
{

    protected $cne ="123125";

    public function afficher()
    {
        echo "<p> CNE=" . $this->cne . " Nom=" . $this->nom . " cin=" .
$this->cin . "</p>";
    }
}

class EtudiantENSAH extends Etudiant
{

    private $loginEservice="12121";

    public function afficher()
    {
        echo "<p> Code Eservices " . $this->loginEservice . " CNE=" .
$this->cne . " Nom=" . $this->nom . " cin=" . $this->cin . "</p>";
    }
}
```

```php
$personne = new Personne();
$personne->afficher();

$etd = new Etudiant();
$etd->afficher();

$etdEnsah = new EtudiantENSAH();
$etdEnsah->afficher();
```

Le résultat de l'exemple ci-dessus est :

Nom=BOUDAA cin=111111
CNE=123125 Nom=BOUDAA cin=111111
Code Eservices 12121 CNE=123125 Nom=BOUDAA cin=111111

15.3. Appel des méthodes de la classe mère

```php
class Personne
{
    protected $nom ="BOUDAA";
    protected $cin ="111111";

    public function afficher()
    {
        echo "<p> Nom=" . $this->nom . " cin=" . $this->cin . "</p>";
    }
}

class Etudiant extends Personne
{

    protected $cne ="123125";

    public function afficher()
    {
        echo "<p> CNE=" . $this->cne  . "</p>";
        parent::afficher();
    }

}

$etd = new Etudiant();
$etd->afficher();
```

Le résultat de l'exemple ci-dessus est :

CNE=123125
Nom=BOUDAA cin=111111

16- Comparaison et clonage des objets

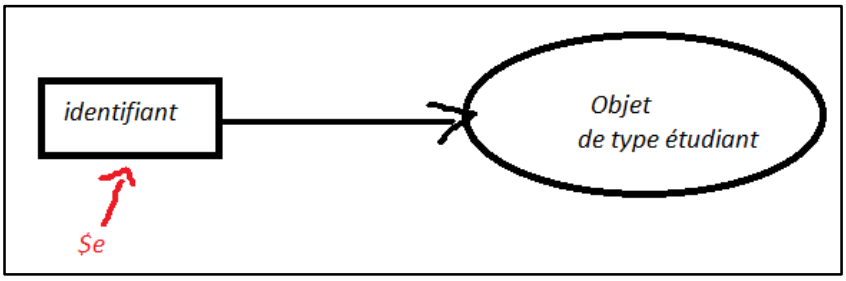
15.1. Identifiant d'un objet

Une référence PHP est un alias, qui permet à deux variables différentes de représenter la même valeur. Dans PHP, une variable objet ne contient plus l'objet en lui-même comme valeur. Elle contient seulement un identifiant d'objet, qui permet aux accesseurs d'objets de trouver cet objet. Lorsque l'objet est utilisé comme argument, retourné par une fonction, ou assigné à une autre variable, les différentes variables ne sont pas des alias : elles contiennent des copies de l'identifiant, qui pointent sur le même objet.

On considère l'instruction suivante :

```
$e =  new Etudiant("Boudaa", "Tarik");
```

$e ne contient pas l'objet mais uniquement son identifiant :



15.2. Comparer deux objets

La comparaison peut se faire naturellement avec ==. Le résultat de la comparaison est TRUE si les valeurs des attributs sont les mêmes et la classe des objets comparé est également la même.

Exemple :

Soit les instructions suivantes

```
$e1 =  new Etudiant("Boudaa", "Tarik");
$p = new Prof("Boudaa", "Tarik");
$e2 =  new Etudiant("Boudaa", "Tarik");
```

Le code suivant affichera : « Non » car les objets comparés ne sont pas de même classe :

```
if($e1==$p){
    echo "Oui";
}
else {
    echo "Non";
}
```

Le code suivant affichera : « Oui » car les objets comparés sont de la même classe et avec des valeurs égales pour les attributs :

```
if($e1==$e2){
```

```
    echo "Oui";
}
else {
    echo "Non";
}
```

ATTENTION :
L'expression $e1 === $e2 retournera FALSE, car « === » compare les identifiants des objets.

Exemple :

```
// Vérifier $e1 et $e2 référence la même instance
if ($e1 === $e2) {
    echo "OUI<br>";
} else {
    echo "NON<br>";
}

//Affectation entre IDs d'objets
$e1 = $p;

if ($e1 === $p) {
    echo "IDENTIQUES<br>";
}
```

Le résultat de l'exécution est :

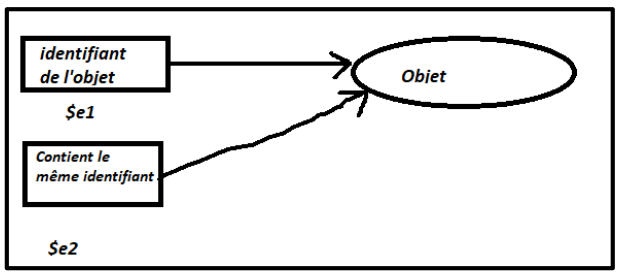
NON
IDENTIQUES

15.3. Clonage des objets

On considère les instructions :

```
$e3 =  new Etudiant("Boudaa", "Tarik");
$e4 =  new Etudiant("Boudaa", "Tarik");

//Affectation entre IDs d'objets
$e4 = $e3;
```

Après cette affectation voici comment on peut représenter la situation :



Il donc facile de comprendre pourquoi le code suivant affichera : « BOUDAA1 »

```php
$e3->setNom("BOUDAA1");
echo $e4->getNom() ,'<br>';
```

Pour copier les valeurs d'un objet vers un autre objet on utilise le clonage des objets avec le mot clé clone.

Syntaxe :

```php
$nom_objet1 = clone nom_objet2;
```

Exemple :

```php
$e3 =  new Etudiant("Boudaa1", "Tarik1");
$e5 =  new Etudiant("Boudaa", "Tarik");
//Clonage des objets
$e5 = clone $e3;
echo $e5->getNom() ,'<br>';

$e3->setNom("BOUDAA2");

echo $e5->getNom() ,'<br>';
```

Le résultat de l'exécution est :

Boudaa1
Boudaa1

17- parcourir les valeurs des attributs d'un objet avec foreach

Exemple 1 :

```php
$e =  new Etudiant("Boudaa", "Tarik");

//parcourir les valeurs des attributs visibles
foreach($e as $att=>$valAtt){
    echo "==".  $att . " ".$valAtt;
}
```

Exemple 2 :

```php
class Etudiant
{

    private $nom;
    private $prenom;

    public function __construct($nom, $prenom)
    {
        $this->nom = $nom;
        $this->prenom = $prenom;
    }
```

```
    public function testValAtt(){
        foreach($this as $att=>$valAtt){
            echo  $att . "=".$valAtt .' <br>';
        }
    }
}
$e = new Etudiant("BOUDAA", "Tarik");
$e4->testValAtt();
```

Le résultat de l'exécution est :

nom=BOUDAA
prenom=Tarik

18- méthodes magiques PHP

Les méthodes magiques sont des méthodes qui vont être appelées automatiquement dans le cas d'un évènement particulier.

Les méthodes magiques leur nom commence par un double underscore __.

La méthode __construct ()

Un exemple de ces méthodes est le construction d'un objet: __construct().En effet, cette méthode s'exécute automatiquement dès que l'on instancie une classe dans laquelle on a défini un constructeur. Donc il est exécuté suite à l'événement d'instanciation.

La méthode __destruct()

est la méthode magique qui est appelée dès qu'il n'y a plus de référence sur un objet donné ou dès qu'un objet n'existe plus dans le contexte d'une certaine application.

La méthode magique __toString()

La méthode magique __toString() est appelée dès que l'on va traiter un objet comme une chaîne de caractères (par exemple lorsqu'on tente d'afficher un objet avec un appel à *echo*, lors de la concaténation d'un objet et une chaîne de caractère,…).

La méthode __toString() doit obligatoirement retourner une chaîne de caractères.

La méthode magique __clone()

La méthode magique __clone() s'exécute dès que l'on crée un clone d'un objet. La méthode __clone qui sera appelée n'est pas celle de l'objet copié mais celle de l'objet vers lequel la copie est faite :

Exemples :

```
class Etudiant
{

    private $nom;
    private $prenom;
```

```php
    public function __construct($nom, $prenom)
    {
        $this->nom = $nom;
        $this->prenom = $prenom;
    }

    public function __toString()
    {
        return "Etudiant : " . $this->nom . " / " . $this->prenom . '<br>';
    }

    public function __clone()
    {
        $this->nom = 'NOM :' . $this->nom;
    }

    public function getNom()
    {
        return $this->nom;
    }

    public function getPrenom()
    {
        return $this->prenom;
    }

    public function setNom($nom)
    {
        $this->nom = $nom;
    }

    public function setPrenom($prenom)
    {
        $this->prenom = $prenom;
    }
}

//On instancie deux objets
$e1 = new Etudiant("BOUDAA", "Mohamed");
$e2 = new Etudiant("KARAMI", "Said");

//Ceci va appeler automatiquement la méthode __toString
echo $e1;

//Ceci va appeler automatiquement la méthode __clone de l'objet $e2
$e2 = clone $e1;

//Ceci va appeler automatiquement la méthode __toString sur $e2
echo $e2;

//Ceci va appeler automatiquement la méthode __toString sur $e1
echo $e1;
```

Le résultat de l'exécution est :

Etudiant : BOUDAA / Mohamed
Etudiant : NOM :BOUDAA / Mohamed
Etudiant : BOUDAA / Mohamed

La liste complète des méthodes magiques :

https://www.php.net/manual/fr/language.oop5.magic.php

19- Les classes abstraites

Une méthode abstraite est une méthode qui ne possède pas un corps et dont seule la signature (c'est-à-dire le nom et les paramètres) est déclarée.

Dès qu'une classe possède une méthode abstraite, il va falloir la déclarer comme classes abstraite.

Une classe abstraite est une classe non instanciable.

Syntaxe :

```
abstract class NomClass {

    //.. attributs

    //méthodes abstraite
    public abstract  function nomMethode($param1, $param2, …);

    //autres méthodes abstraites ou non

}
```

20- Les interfaces

20.1. Définir une interface

Syntaxe

```
interface Nom_Interface{
    public function methode1();
      …
      public function methodeN();
}
```

Exemple :

```
interface IExecutable
{

    public function execute();
}
```

20.2. Implémenter une interface :

```
class Document implements  IExecutable
{
    private $nomFile;

    public function __construct($nomF){
        $this->nomFile = $nomF;
    }
       public function execute()
    {
            //Votre code ici
    }

}
```

20.3. Héritage entre interfaces

```
interface IOperable
{

    public  function  operate();
}
```

L'interface IPrintable hérite à la fois de deux interfaces IOperable et IExecutable. **Attention l'héritage multiple n'est pas autorisé entre les classes en PHP.**

```
interface IPrintable extends  IOperable , IExecutable{

    public function print();

    public function export();

}
```

20.4. Implémenter plusieurs interfaces

Dans ce cas il faut implémenter les méthodes de toutes les interfaces implémentées

```
class Document implements  IPrintable,  IExecutable
{
    private $nomFile;

    public function __construct($nomF){
        $this->nomFile = $nomF;
    }

    public function print()
    {
        echo '<p>'. $this->nomFile .'<p>';

    }

    public function export()
```

```
    {
        echo '<div>'. $this->nomFile .'<div>';
    }

    public function operate()
    {
      //Code...
    }

    public function execute()
   {
      //Code...
    }


}
```

21- Les closures (Les fonctions anonymes)

Les closures, représentant ce qu'on appelle des fonctions anonymes qui sont des fonctions particulières qui n'ont pas un nom. Elles représentent un moyen de créer des fonctions à la volée.

Si vous avez vu en langage C ou C++ les pointeurs de fonctions vous pouvez facilement faire la liaison avec les closures en PHP.

Syntaxe

```
function()
{
    echo 'Hello world !';
};
```

Une closure est en fait un objet. En réalité, il s'agit d'une instance de la classe **Closure.**

Pour vérifier ceci nous pouvons exécuter le code suivant :

```
//On définit une closure

$maFonction = function()
{
    echo 'Hello world !';
};
// On découvre ici qu'il s'agit bien d'un objet de type Closure.
var_dump($maFonction);
```

Le résultat de l'exécution est :

```
object(Closure)#1 (0) { }
```

La classe Closure possède une méthode magique nommée__*invoke*. Cette méthode est invoquée lorsque l'on se sert de notre objet comme une fonction.

Exemple :

```
$maFonction = function()
{
    echo 'Hello world !';
};

$maFonction(); // Affiche « Hello world ! »
```

Les closures sont principalement utilisées en tant que fonctions callback (fonctions de rappels). Les fonctions de rappels sont des fonctions demandées par d'autres fonctions pour effectuer des tâches spécifiques (Fonctions admettant comme paramètre des fonctions)

Exemple:

```
//Fonction prenant comme paramètre une fonction et un tableau

function appliquerUnTraitement($callback, $array)
{
    for ($i = 0; $i < count($array); $i ++) {
        $array[$i] = $callback($array[$i]);
    }
    return $array;
}

//Si nous voulons par exemple appliquer un traitement qui transforme un tableau
//d'entiers en puissance 2
//On passe en paramètre une closure calculant la puissance et un tableau

$res = appliquerUnTraitement(function ($n) {
    return $n * $n;
}, [
    1,
    2,
    3
]);

foreach ($res as $it) {
    echo $it . '<br>';
}
```

Le résultat de l'exécution est :

```
1
4
9
```

22- Auto-chargement des classes

De nombreux développeurs qui écrivent des applications orientées objet créent un fichier source par définition de classe. Un des plus gros inconvénients de cette méthode est d'avoir à écrire une longue liste d'inclusions de fichier de classes au début de chaque script : une inclusion par

classe. Pour résoudre ce problème PHP propose un moyen pour charger les classes automatiquement avec la fonction *spl_autoload_register()*

Exemple :

On suppose que nous avons défini deux classes dans des fichiers séparés Ingenieur.class.php et Technicien.class.php

```php
spl_autoload_register(function ($ClassName) {
    require_once $ClassName . '.class.php';
});

class Personne
{

    function test()
    {
        $i = new Ingenieur();
        $t = new Technicien();
    }
}

$p = new Personne();
$p->test();
```

23- Espaces de noms

Les espaces de noms (*namespaces*) sont des qualificatifs qui résolvent deux problèmes différents :

- Ils permettent une meilleure organisation des classes.
- Ils permettent d'utiliser le même nom pour plus d'une classe Par exemple, vous pouvez avoir une classe qui décrivent un tableau HTML nommée Table et une autre classe nommée aussi Table qui décrit une table à manger. Les espaces de noms peuvent être utilisés pour organiser les classes en deux groupes différents tout en empêchant également les deux classes Table et Table d'être mélangées.

Les espaces de noms sont comme une structure de répertoires virtuels pour vos classes.

Tout code qui suit une déclaration d'espace de noms fonctionne à l'intérieur de l'espace de noms, de sorte que les classes qui appartiennent à l'espace de noms peuvent être instanciées sans aucun qualificatif. Pour accéder aux classes depuis l'extérieur d'un espace de noms, la classe doit avoir l'espace de noms qui lui est attaché.

23.1. Déclaré un espace de noms :

On utilise la syntaxe : `namespace nom_de_espace_de_nom;`
Exemple 1 :

```php
<?php
// Déclaration du namespace.
namespace nom_de_espace_de_nom;
```

```php
// Toutes les constantes, fonctions et classes
// déclarées ici feront partie du namespace nom_de_espace_de_nom.
?>
```

Exemple 2:

```php
<?php
namespace nom_de_espace_de_nom;

// Déclaration du namespace.

// Toutes les constantes, fonctions et classes
// déclarées ici feront partie du namespace nom_de_espace_de_nom.
function test()
{
    echo "function test";
}

namespace nom_de_espace_de_nom2;

echo \nom_de_espace_de_nom\test();

?>
```

Exemple 3 : Déclaration d'une classe dans l'espace de nom « ensah »

```php
namespace ensah;

class Etudiant
{

}
```

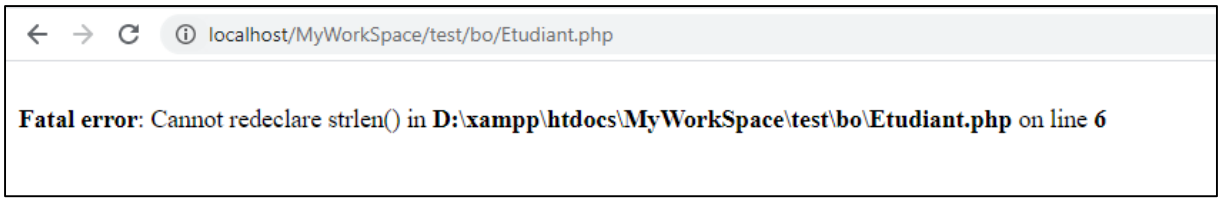
Exemple 3 : redéfinition d'une classe php existante dans un autre espace de noms

Strlen est une fonction PHP qui retourne la taille d'une chaîne de caractères. Si nous définissons une fonction avec le même nom comme dans le code suivant nous allons obtenir une erreur à l'exécution :

```php
<?php

function strlen()
{
    echo 'How are you ENSAH?';
}

strlen();

?>
```

Résultat :

Erreur à l'exécution



Nous pouvons éviter cette erreur on définissant cette fonction dans un autre espace de noms comme dans le code suivant :

```php
<?php
namespace test_namespace;

function strlen()
{
    echo 'How are you ENSAH?';
}

strlen();

?>
```

Résultat :

How are you ENSAH?

23.2. Accéder au nom du namespace :

On peut utiliser la constante magique : __NAMESPACE__

```php
namespace ensah;


echo __NAMESPACE__;
```

résultat :

ensah

23.3. Utilisation des éléments d'un espace de noms :

Pour utiliser une classe « Etudiant » déclarée dans l'espace de nom « ensah » on utilise la syntaxe suivante :

```php
$etd = new \ensah\Etudiant () ;
```

Si vous voulez appeler une fonction du namespace global dans ce cas il suffit tout simplement de faire précéder le nom de la fonction ou de la classe d'un backslash (\)

Exemple :

```php
<?php

namespace ensah;
```

```php
function strlen(){
    return -1;
}
echo "strlen du namsepace ensah: ".  strlen("Boudaa");
echo '<br>';
echo "strlen du namsepace global: ".  \strlen("Boudaa");
```

Résultat :

strlen du namsepace ensah: -1
strlen du namsepace global: 6

23.4. Utilisation des alias

Il peut être utile de donner un alias à un espace de noms ou à une classe pour faciliter l'écriture. Ceci est fait avec le mot-clé « use »:

Exemple :

```php
namespace bo;
require_once 'test.php';

use ensah\scolarite as es;

es\test();
```

Exemple :

```php
Donnez un alias à une classe :
use ensah\scolarite\Etudiant as E;
$etd = new E();
```

23.5. Les sous espaces de noms

Pour plus d'organisation, il est possible d'avoir des espaces de noms imbriqués :

Exemple 1:

```php
namespace ensah\scolarite;
function test()
{
    echo "Je suis un test";
}
```

Exemple 2:

```php
namespace ensah\scolarite;

class Etudiant
{
```

```
}
```

Utilisation de la classe Etudiant :

```
$etd = new \ensah\scolarite\Etudiant ();
```

Exemple 3:

```
<?php
    use A\B\C\D\E\F as nsF, G\H\I\J\K\L as nsL;
?>
```

23.6. Plusieurs espaces de noms dans un même fichier

La Déclaration de plusieurs namespaces dans un même fichier est possible mais elle est déconseillée.

```
<?php
namespace test;

class Etudiant
{

}

namespace bo;

use \test\Etudiant;

new Etudiant();
```

23.7. Appels relatif et absolu

Exemple 1:

```
<?php
    namespace A\B
    {
        function getNamespace()
        {
            echo __NAMESPACE__;
        }
    }

    namespace A
    {
        B\getNamespace(); // Appel de façon relative.
        \A\B\getNamespace(); // Appel de façon absolue.
    }
?>
```

Exemple 2:

```
<?php
namespace test\testA;

class Etudiant
```

```
{
}

namespace test;

new testA\Etudiant(); // Utilisation relative

new \test\testA\Etudiant(); //Utilisation absolue
```

23.8. Importation de classes

Nous allons voir une façon d'utiliser use qui permet d'importer des classes, et uniquement des classes (ça ne fonctionnera pas avec des constantes ou fonctions). Le but est d'importer la classe d'un certain namespace dans le namespace courant. Par exemple, dans le fichier F.ns.php, nous allons créer une classe toute bête :

```
<?php
    namespace A\B\C\D\E\F;

    class MaClasse
    {
        public function hello()
        {
            echo 'Hello world !';
        }
    }
?>
```

Pour importer la classe, il suffit de d'utiliser **use espace\de\noms\MaClasse**. Exemple :

```
<?php
    require 'F.ns.php';

    use A\B\C\D\E\F\MaClasse;

    $a = new MaClasse; // Se transforme en $a = new A\B\C\D\E\F\MaClasse.
    $a->hello();
?>
```

Avec alias

```
<?php
    require 'F.ns.php';

    use A\B\C\D\E\F\MaClasse as Hello;

    $a = new Hello; // Se transforme en $a = new A\B\C\D\E\F\MaClasse.
    $a->hello();
?>
```

24- Réflexion

La réflexion est une technique qui permet de récupérer des informations au moment d'exécution sur les classes et les objets. Par exemple :

- Obtenir des informations sur les attributs d'une classe
- Obtenir des informations sur les méthodes d'une classe
- Obtenir des informations sur les relations entre classes comme les classes mères, les interfaces implémentées,…
- Récupérer les valeurs des propriétés d'un objet
- …

Nous allons découvrir la réflexion avec PHP en pratique en analysant le programme ci-dessous :

```php
// On définit une classe Personne
class Personne
{
}

// On définit une classe Etudiant
class Etudiant extends Personne
{

    private $nom;

    private $prenom;

    public $email;

    const AGE_MAX = 100;

    public function __construct($nom, $prenom, $email)
    {
        $this->nom = $nom;
        $this->prenom = $prenom;
        $this->email = $email;

    }

    public function tester()
    {
        echo "tester", "<br>";
    }

    public function msg($msg)
    {
        echo $msg, "<br>";
    }
}

// Instantiation par reflexion
$classeEtudiant = new ReflectionClass('Etudiant');

// Vérifier si la classe a un attribut nommé "nom"
if ($classeEtudiant->hasProperty('nom')) {
    echo "Oui elle a une propriété nom", "<br>";
}
// Vérifier si la classe a un attribut nommé "age"
if ($classeEtudiant->hasProperty('age')) {
    echo "Oui elle a une propriété age", "<br>";
} else {
    echo "Non elle n'a pas une propriété age", "<br>";
}
```

```php
if ($classeEtudiant->hasMethod('tester')) {
    echo 'La classe Etudiant implémente une méthode tester()', '<br>';
}

if ($classeEtudiant->hasConstant('AGE_MAX')) {
    echo 'La classe Etudiant a une constante AGE_MAX', '<br>';
}

if ($parent = $classeEtudiant->getParentClass()) {
    echo 'La classe Etudiant a une classe mère nommée: ', $parent->getName(),
'<br>';
}

// On récupère une propriété de la classe
$attributNom = $classeEtudiant->getProperty('nom');

echo ($attributNom), '<br>';

// On récupère toutes les propriétés
$attributs = $classeEtudiant->getProperties();
foreach ($attributs as $att) {
    echo $att->getName(), '<br>';
}

$etd = new Etudiant('BOUDAA','Tarik','test@test.com');
// On récupère toutes les propriétés publique
$attributs = $classeEtudiant->getProperties(\ReflectionProperty::IS_PUBLIC);
foreach ($attributs as $attribut) {
    echo $attribut->getName(), ' => ', $attribut->getValue($etd), '<br>';
}


//Appel d'une fonction par réflexion
call_user_func_array(array(
    $etd,
    'msg'
), array("Message par réflexion"));
```

Résultat d'exécution :

Oui elle a une propriété nom
Non elle n'a pas une propriété age
La classe Etudiant implémente une méthode tester()
La classe Etudiant a une constante AGE_MAX
La classe Etudiant a une classe mère nommée: Personne
Property [private $nom]
nom
prenom
email
email => test@test.com
Message par réflexion