PHP

- Maj : 30/03/2025 -

Variables : $var var_dump($var)echo "<HTML>"; Opérateur de concaténation : .
Type variable : gettype($var)

cast int vers float avec forçage 2 décimales :

```
$res = number_format(floatval($val_Int),2) ;
```

Opérateurs logiques :

`$a and $b  And (Et)`	`$a or $b  Or (Ou)`	`$a xor $b  XOR, Ou exclusif`
`! $a  Not (Non)`	`$a && $b  And (Et)`	`$a || $b  Or (Ou)`

Déclaration de tableau :

```php
<?php
$tab = array("val1","val2");
$tab2 = array(array("val1","val2"), array("val3","val4"));
echo $tab2[1][2];
?>
```

Opérations sur les tableaux :

count() et sizeof() : retournent la taille d'un tableau.
sort() trie les éléments du + petit au + grand
rsort() : du + grand au + petit
in_array() : vérifie si une valeur est présente dans un tableau
array_rand() : extrait une ou plusieurs valeurs d'un tableau au hasard
current() : retourne la valeur courante de l'élément du tableau où se trouve le pointeur.

```php
<?php
foreach ($legume as $valeur){
      echo $valeur, '<br>' ;
}
// avec de l'associatif :
foreach ($identite as $cle => $valeur){
      echo $cle, ' : ', $valeur, '<br>' ;
}
?>
```

Tableau à 2 dimensions :

```php
<?php
$chanteurs = array(
    array('prenom'=>'Roger', 'nom'=>'WATERS'),
    array('prenom'=>'Sid', 'nom'=>'BARRETT')
    );
foreach($apprenants as $ligne){        // Lecture de chaque ligne du tableau
        // Lecture de chaque colonne
        foreach($ligne as $cle=>$valeur){           // Affichage
             echo $cle.': '.$valeur;
             echo '<br>';
             }
      }
?>
```

Attention, ici, on utilise la même variable **$ligne** dans les 2 boucles, c'est elle qui va faire le lien ligne/colonnes.

Syntaxe NowDoc :	**Syntaxe HereDoc :**
`$foo = 'bar';`	`$foo = 'bar';`
`echo <<<'EOT'`	`echo <<<EOT`
`Hello $foo`	`Hello $foo`
`Goodbye!`	`Goodbye!`
`EOT;`	`EOT;`

Fonctions sur String :

Fonctions	Exemple	Résultat	Description
nl2br	nl2br("M \n D")	M D	insère un à chaque nouvelle ligne
strcmp	strcmp ("Moussa","Moise")	12	compare 2 chaînes de caractères.
strlen	strlen("Moussa")	6	retourne le nombre de caractère de la chaîne
strpos	strpos("Moussa",'s')	3	renvoie la position de la première occurrence d'un caractère
strrev	strrev("Moussa")	assuoM	inverse une chaîne
strtolower	strolower("MoussaDiop")	moussadiop	convertit les majuscules en minuscules
strtoupper	strtoupper("Moussa")	MOUSSA	convertit les minuscules en majuscules
trim	trim(" Moussa ")	Moussa	Supprime les espaces en début et fin de chaîne
echo, print	echo ("Bonjour")	1.35	Afficher une chaîne de caractères
ucfirst	ucfirst("moussa")	Moussa	Transforme le premier caractère en majuscule
substr	substr("moussa",2,3)	uss	Sous-chaîne avec offset + longueur

Tabulation :

```php
<?php echo "<pre>A\tB</pre>"; ?>
```

Fonction is_null()

Vérifie si une variable est nulle.
Attention : si c'est vrai, la fonction retourne 1, mais ne retourne rien dans le cas contraire.

Fonction empty()

Vérifie si une variable est vide.
La fonction retourne false si la variable existe et n'est pas vide, sinon elle retourne true.
Exemples de variables vides :

0	0.0	"0"	""	NULL	FALSE	array()

Fonction isset()

Vérifie si une variable est vide, et aussi si elle est déclarée.

```php
$a = 0;
// True because $a is set
if (isset($a)) {
  echo "Variable 'a' is set.<br>";
}
```

```php
$b = null;
// False because $b is NULL
if (isset($b)) {
  echo "Variable 'b' is set.";
}
```

Quelques fonctions

exec

Exemple #1 Exemple avec exec() **exec($command, $output_tab, $retval)**

```php
<?php
// Affiche le nom d'utilisateur qui fait tourner le processus php/http
// (sur un système ayant "whoami" dans le chemin d'exécutables)
$output=null;
$retval=null;
exec('whoami', $output, $retval);
echo "Returned with status $retval and output:\n";
print_r($output);
?>
```

Résultat de l'exemple ci-dessus est similaire à :

```
Returned with status 0 and output:
Array
(
    [0] => cmb
)
```

$retval est un code retour (1 ou 0) : optionnel

$output : tableau de lignes retournées par la commande – les espaces de début et fin de chaîne comme \n ne sont pas inclus. L'argument $output est aussi facultatif.

Si `command` est vide ou contient des octets nuls, **exec()** lève désormais une exception ValueError.

Note :

print_r ( ) est une fonction intéressante, elle permet l'affichage d'une variable sous un format "human readable".

in_array()

in_array($aiguille, $botteDeFoin) vérifie que $aiguillle (type String, par exemple) est contenu dans le tableau (Array) $botteDeFoin

exemple :

```php
<?php
$os = array("Mac", "NT", "Linux");
if (in_array("Linux", $os)) {
echo "Got Linux !";
}
?>
```

PDO – Accès base de données

Instanciation d'un objet PDO

`$dbh=new PDO(DSN [, user [, pass [, options]]]);`

DSN : Data Source Name > nom_du_driver:syntaxe_spécifique_au_driver

`Ex : mysql:host=localhost;dbname=ma_base`

Fin de connexion : `$dbh=null ; ou unset($dbh) ;`

Connexion avec gestion d'erreur :

```php
<?php
try {
 $dbh = new PDO('mysql:host=h;dbname=db',$user, $pass) ;
 …
 $dbh = null ;
}
catch (PDOException $e) {
 echo "<p>Erreur: ".$e->getMessage() ;
 die() ;
}
?>
```

```php
<?php
try {
$dbh = new PDO('mysql:host=localhost; dbname=lafleur; charset=utf8','lafleur', 'secret');
$requete = "select * from produit where pdt_category='".$_GET['cat']."'";
$result = $dbh->query($requete);

$dbh=null;
}
catch (PDOexception $e) {
      echo "Erreur !".$e->getMessage();
      die();
}
?>
```

fetchall()

Récupération de toutes les lignes d'un jeu de résultat :

```php
<?php
$sth = $dbh->prepare("SELECT nom, couleur FROM fruit");
$sth->execute();

/* Récupération de toutes les lignes d'un jeu de résultats */
print("Récupération de toutes les lignes d'un jeu de résultats :\n");
$result = $sth->fetchAll();
print_r($result);
?>
```

Exemple #2 Récupération de toutes les valeurs d'une seule colonne depuis un jeu de résultats

L'exemple suivant montre comment retourner toutes les valeurs d'une seule colonne depuis un jeu de résultats, même si la requête SQL retourne plusieurs colonnes par lignes.

```
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();

/* Récupération de toutes les valeurs de la première colonne */
$result = $sth->fetchAll(PDO::FETCH_COLUMN, 0);
var_dump($result);
?>
```

```
=============
```

Exemple 1. Exécution d'une requête préparée avec des emplacements nommés

```
<?php
/* Exécution d'une requête préparée en liant des variables PHP */
$calories = 150;
$couleur = 'rouge';
$sth = $dbh->prepare('SELECT nom, couleur, calories
    FROM fruit
    WHERE calories < :calories AND couleur = :couleur');
$sth->bindParam(':calories', $calories, PDO_PARAM_INT);
$sth->bindParam(':couleur', $couleur, PDO_PARAM_STR, 12);
$sth->execute();
?>
```

Exemple 2. Exécution d'une requête préparée avec des marques de positionnement

```
<?php
/* Exécution d'une requête préparée en liant des variables PHP */
$calories = 150;
$couleur = 'rouge';
$sth = $dbh->prepare('SELECT nom, couleur, calories
    FROM fruit
    WHERE calories < ? AND couleur = ?');
$sth->bindParam(1, $calories, PDO_PARAM_INT);
$sth->bindParam(2, $couleur, PDO_PARAM_STR, 12);
$sth->execute();
?>
```

Autre exemple de requête préparée (Site "Lafleur") :

```php
<?php

$dbh = new PDO("mysql:host=localhost; dbname=$base; charset=utf8", $user, $pass);

// requête préparée

$req_prep = "SELECT clt_code,clt_motPasse from clientconnu WHERE clt_code = :cc AND
clt_motPasse = :cp;";

$st = $dbh->prepare($req_prep);

echo "<fieldset><Legend>TEST</legend>";
echo "codeclient : ".$_GET['codeclient']." passclient : ".$_GET['passclient']."<br>";
echo "</fieldset>";

    $st->bindParam(':cc', $_SESSION['codeclient']);
    $st->bindParam(':cp', $_SESSION['passclient']);
    $st->execute();
    $result = $st->fetchAll();


foreach ($result as $row){
   $taille2+=1; // incrémentation compteur
            }

if (isset($result) && $taille2>0){
// si le codeclient + motPasse correspondent à un enregistrement en BDD
   echo "<br>Commande autorisée : vous avez été authentifié(e).<br><hr>";
...
}

?>
```

Liste de paramètres pour bindParam(param, var) :

`param`

Identifiant.
Pour une requête préparée utilisant des marqueurs nommés, ce sera le nom du paramètre sous la forme ***:name***.
Pour une requête préparée utilisant les marqueurs interrogatifs, ce sera la position indexée +1 du paramètre.

`var`

Nom de la variable PHP à lier au paramètre de la requête SQL.

Insertion sans requête préparée

```php
<?php
$servername='localhost';
$username='root';
$password='';
$dbname = "crud";
try {
    $first_name = $_POST['first_name'];
    $last_name = $_POST['last_name'];
    $city_name = $_POST['city_name'];
    $email = $_POST['email'];
    date_default_timezone_set("Asia/Calcutta");
    $insertdate = date("Y-m-d H:i:s");
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);

/* set the PDO error mode to exception */
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   $sql = "INSERT INTO employee (first_name,last_name,city_name,email,datetime)
    VALUES ('$first_name', '$last_name','$city_name','$email','$insertdate')";
    $conn->exec($sql);
    echo "New record created successfully";
    }
catch(PDOException $e)
    {
        echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
?>
```

Exemple de page HTML / PHP avec gestion mot de passe

```
<!DOCTYPE HTML>
<html>
<head><title>Exercice3 - php</title><Meta charset="UTF-8"></head>
<body>
<form method="POST" action="#">
Nom : <input type="text" name="le_nom">
Mot de passe : <input type="password" name="le_mdp">
<input type="submit" name="bouton" value="envoyer">
</form>
<?php
if (isset($_POST['le_mdp']) && isset($_POST['le_nom'])){
      if ($_POST['bouton'] == 'envoyer') {
            if (empty($_POST['le_mdp'])||empty($_POST['le_nom'])){
                  echo "<br>Au moins un champ vide ! Verboten !";
                  }
            $nomCasse = trim(strtolower($_POST['le_nom']), " ");
            $mdpCasse = trim(strtolower($_POST['le_mdp']), " ");
            if (($nomCasse=="greta") && ($mdpCasse== 'greta')) {
                  echo "<br>Mot de passe ok : bienvenue.<br>";
      //echo "<H2>Nom après modif de la casse : $nomCasse - MDP : $mdpCasse </H2>";
                                                      }
            else {
                  echo "<br>Accès refusé pour l'utilisateur ".$_POST['le_nom']." .";
                                                      }
    }
}
?>
</body>
</html>
```

Inclure un script PHP externe à partir d'une page Web

L'extension de la page appelante doit être *.php

Lors de l'utilisation de **include**, en cas d'impossibilité d'accéder à la page appelée, PHP renvoie un simple avertissement.
Dans le cas de **require**, une erreur fatale sera retournée + arrêt du script.

Variantes :

include_once : un fichier ne pourra être inclus qu'une seule fois, contrairement à include
Idem pour **require_once**

```
<body>
     <h2>les 3 parties</h2>
<?php
     print "Partie 1 avec include\n" ;
     include "part1.php" ;
     print "Partie 2 avec require\n" ;
     require "part2.php" ;
     print "Troisième partie\n" ;
     include_once "part1.php" ; require_once "part2.php" ;
?>
...
```

Inverser l'ordre des lignes d'un fichier

```php
<?php
$file = 'myFile.txt';
$file_contents = array_reverse(file($file));
foreach($file_contents as $line){
echo $line;
}
?>
```

POO / constructeur / CAST objet >>> tableau

```php
<?php

class Car {
  public $color;
  public $model;
  public function __construct($color, $model) {
    $this->color = $color;
    $this->model = $model;
  }
  public function message() {
    return "My car is a " . $this->color . " " . $this->model . "!";
  }
}

$myCar = new Car("red", "Volvo");

$myCar = (array) $myCar;
var_dump($myCar);

?>
```

GESTION DES DATES

```php
<?php date_default_timezone_set('UTC'); ?>
```

Classe DateTime :

```php
<?php $madate = new DateTime('now') ; ?>
<?php $madate2 = new DateTime('2021-05-26') ; ?>
```

Conversion DateTime => String :

```php
<?php $madateStr = $madate->format('d-m-Y H:i:s') ; ?>
```

Conversion DateTime object => TimeStamp :

```php
<?php $madateStamp = $madate->getTimestamp() ; ?>
```

Conversion TimeStamp => DateTime object:

```php
<?php
use DateTimeImmutable;
$timestamp = '2678901034';
$datetime = DateTimeImmutable::createFromFormat('Y-m-d H:i:s', $timestamp);
?>
```

ou avec **getdate()** : attention : retourne un tableau.

```php
<?php $gd = getdate($timestamp)['year'];  ?>
```

['seconds'] => 0, ['minutes'] => 0, ['hours'] => 0, ['mday'] => 21, ['wday'] => 6, ['mon'] => 5, ['year'] => 2011

Conversion String => TimeStamp

```php
<?php $d_str = strtotime("2010/06/23"); ?>
```

Le Format ISO :

```php
<?php
$date = '2024-02-25';
$dateTime = new DateTime($date);
echo 'La date '.$date.' convertie au format Français donne '.$dateTime->format('d/m/Y');
?>
```

Ajouter / soustraire du temps :

Il faut instancier un objet DateInterval. => le paramètre commence par la lettre P, suivi d'un nombre d'années, mois, heures, min…

```php
<?php
$i1 = new DateInterval('P1Y'); // intervalle = 1 an
$i2 = new DateInterval('P1M3D'); // intervalle = 1 mois, 3 jours
$date = new DateTime('2024-02-25');
echo "date (pre) : ".$date->format('d/m/Y').PHP_EOL;
$newDate = $date->add($i1);
echo 'nouvelle date  (add) : '.$newDate->format('d/m/Y').PHP_EOL;
$newDate = $date->sub($i2);
echo 'nouvelle date (sub) : '.$newDate->format('d/m/Y').PHP_EOL;
?>
```

Résultat :

```
date (pre) : 25/02/2024
nouvelle date  (add) : 25/02/2025
nouvelle date (sub) : 22/01/2025
```

Attention : on voit que la dernière date a été obtenue en retranchant 1mois et 3 jours à la date précédente (celle qui avait reçue l'incrément +1 an)
$date est donc une instance dont la valeur est modifiée au fur et à mesure des traitements – Il ne s'agit pas d'une constante date immutable définie à l'origine ($date = **new** DateTime('2024-02-25');)

Avec instanciation d'un objet immutable :

```php
<?php
$i1 = new DateInterval('P1Y'); // intervalle = 1 an
$i2 = new DateInterval('P1M3D'); // intervalle = 1 mois, 3 jours
$dateImmu = new DateTimeImmutable('2024-02-25');
echo "date (pre) : ".$dateImmu->format('d/m/Y').PHP_EOL;
$newDate = $dateImmu->add($i1);
echo 'nouvelle date  (add) : '.$newDate->format('d/m/Y').PHP_EOL;
$newDate = $dateImmu->sub($i2);
echo 'nouvelle date (sub) : '.$newDate->format('d/m/Y').PHP_EOL;
?>
```

Ici, on utilise l'objet DateTimeImmutable, qui permet au contraire de conserver la date de départ servant de base aux opérations d'addition et de soustraction.

```
date (pre) : 25/02/2024
nouvelle date  (add) : 25/02/2025
nouvelle date (sub) : 22/01/2024
```

L'année après soustraction reste bien 2024.

Objets en PHP

```php
<?php

$objet = new Utilisateur('exemple_nom', 'mot_de_passe');
$objet->nom = "Daniel";
$objet->motpass = "pastaga";
$objet->enregistrer_utilisateur();
Utilisateur::bienvenue(); // appel possible directement par la classe car méthode static


class Utilisateur {

public $nom, $motpass; // déclaration de propriétés
const FRANCAIS = 1; // déclaration de constante

//constructeur
function __construct($param1, $param2){
     echo "Dans le constructeur, avec param1=".$param1." et param2=".$param2."<br>";
}

function donnePass() { // méthode exemple
     return $this->motpass;
     }

static function bienvenue(){ // méthode "static"
     echo "<h2>BIENVENUE !</h2>";
     echo "constante : ".self::FRANCAIS;
     }

function enregistrer_utilisateur(){
     echo "Code pour l'enregistrement";
     }
}

?>
```

Note :

pas de $ devant la variable quand on fait référence à l'objet – ex : $objet->variable ou encore : self::variable

self::nom_variable => utilisation de "self" pour accéder à une propriété ou constante static

Le mot-clef $this est utilisé en référence à l'objet créé (instanciation de la classe)
$this->nom en référence à la propiété "nom" de l'objet, à l'intérieur de la classe.

:: 2 double points : opérateur de résolution de portée.

Méthode / propriété "static" : appartiennent à la classe, ne sont pas accessibles à partir d'une instance.

Portées : public (par défaut), protected (accessible aux méthode de la classe et de toutes ses sous-classes), private (accès uniquement par les méthodes de la classe)

Héritage

```php
<?php

$labonne = new Abonne('n1','p1');
$labonne->adresse = "Rue de l'asténosphère";

class Abonne extends Utilisateur {
     public $adresse;

     function afficheAdr(){
            return $this->adresse;
     }
}
echo "Retour de fonction afficheAdr() : ".$labonne->afficheAdr();

?>
```

- La classe possède le même constructeur hérité : obligation d'instancier avec 2 paramètres (dans cet exemple).
- Si on définit une méthode dans la classe héritée, qui porte le même nom dans la classe mère => surcharge.
- Dans le cas d'une surchage de méthode dans une classe "fille", on peut appeler la méthode de la classe mère en utilisant le mot-clé "parent"

exemple :
Une méthode afficheMachin() dans la classe mère et fille, appel de la méthode mère dans la classe fille :

```
parent::afficheMachin();
```

- Si présence d'un constructeur dans la classe fille, la méthode constructeur de la classe mère ne sera pas automatiquement appelée : pour le faire, il faudra d'abord utiliser une référence au parent :

```
function __construct(){
parent::__construct();
echo "Le reste du constructeur fils";
  }
```

Pour empêcher la surcharge d'une superclasse, on peut utiliser le mot-clef "final"

```
final function funcTest(){
     echo "Test";
}
```

Exemple de classes avec héritage et constructeur :

```php
<?php
// Définition de la classe parente Animal
class Animal {
    // Propriétés de la classe Animal
    protected $nom;
    protected $age;

    // Constructeur de la classe Animal
    public function __construct($nom, $age) {
        $this->nom = $nom;
        $this->age = $age;
    }

    // Méthode pour afficher les informations de l'animal
    public function afficherInfos() {
        return "Nom: " . $this->nom . ", Age: " . $this->age;
    }
}

// Définition de la classe fille Chien qui hérite de Animal
class Chien extends Animal {
    // Propriété propre à la classe Chien
    private $race;

    // Constructeur de la classe Chien
    public function __construct($nom, $age, $race) {
        // Appel du constructeur de la classe parente
        parent::__construct($nom, $age);
        $this->race = $race;
    }

    // Méthode pour afficher les informations du chien
    public function afficherInfos() {
        // Appel de la méthode de la classe parente et ajout de la race
        return parent::afficherInfos() . ", Race: " . $this->race;
    }
}

// Exemple d'utilisation
$monChien = new Chien("Rex", 3, "Berger Allemand");
echo $monChien->afficherInfos(); // Affiche : Nom: Rex, Age: 3, Race: Berger Allemand
?>
```

Explications :

- **Classe `Animal`** :
    - Elle possède deux propriétés : `nom` et `age`.
    - Le constructeur initialise ces deux propriétés.
    - La méthode `afficherInfos` retourne une chaîne contenant le nom et l'âge de l'animal.
- **Classe `Chien`** :
    - Elle hérite de la classe `Animal`.
    - Elle possède une propriété supplémentaire : `race`.
    - Le constructeur de `Chien` appelle le constructeur de `Animal` pour initialiser `nom` et `age`, puis initialise `race`.
    - La méthode `afficherInfos` est redéfinie pour inclure la race du chien en plus des informations de l'animal.

serialize(), unserialize()

En PHP, les méthodes `serialize` et `unserialize` sont utilisées pour convertir des variables en une représentation sous forme de chaîne de caractères et vice versa. Voici comment elles sont utilisées dans le contexte des variables de session :

3. **Serialize** :
    - La fonction `serialize` convertit une variable PHP en une chaîne de caractères qui peut être stockée ou transmise facilement. Cela inclut les objets, les tableaux et les autres structures de données complexes.
    - Lorsque PHP stocke des variables de session, il utilise `serialize` pour convertir les données de session en une chaîne de caractères avant de les stocker sur le serveur (par exemple, dans un fichier ou une base de données).
4. **Unserialize** :
    - La fonction `unserialize` fait l'inverse de `serialize`. Elle prend une chaîne de caractères précédemment sérialisée et la reconvertit en une variable PHP.
    - Quand une session est récupérée, PHP utilise `unserialize` pour convertir les données stockées sous forme de chaîne de caractères en variables PHP utilisables.

Pourquoi utiliser `serialize` et `unserialize` avec les sessions ?

- **Stockage** : Les sessions doivent souvent être stockées sur le serveur entre les requêtes. `serialize` permet de convertir les données de session en un format qui peut être facilement stocké.
- **Transmission** : Si les données de session doivent être transmises (par exemple, dans un cookie), `serialize` permet de les convertir en une chaîne de caractères qui peut être envoyée.
- **Compatibilité** : Les données sérialisées peuvent être stockées dans différents formats (fichiers, bases de données, etc.) et peuvent être facilement récupérées et désérialisées pour être utilisées dans une autre requête.

Exemple d'utilisation

```php
<?php

// Stockage d'une variable de session

$_SESSION['user'] = array('name' => 'John', 'age' => 30);

// Sérialisation

$serialized = serialize($_SESSION['user']);

// Désérialisation

$user = unserialize($serialized);
```

En résumé, `serialize` et `unserialize` sont essentiels pour gérer les données de session en PHP, car elles permettent de convertir des structures de données complexes en chaînes de caractères pour le stockage et la transmission, puis de les restaurer à leur état d'origine lorsqu'elles sont nécessaires.