

PHP 8 et versions antérieures – Développement d'applications WEB

09/12/202

Glodie Tshimini

Glodie Tshimini

Consultant Formateur et Développeur Web depuis 2017

Certifié Professional Scrum Developer (PSD I)

Certifié 2AICONCEPT Expert Trainer

Certifié 2AICONCEPT IT Expert Trainer At POE

Email : student@tshimini.fr

OBJECTIFS PÉDAGOGIQUES

- Développer des pages Web dynamiques en PHP dans un environnement Internet / Intranet
- Avec une connexion à une base de données

PLAN DU COURS

PLAN

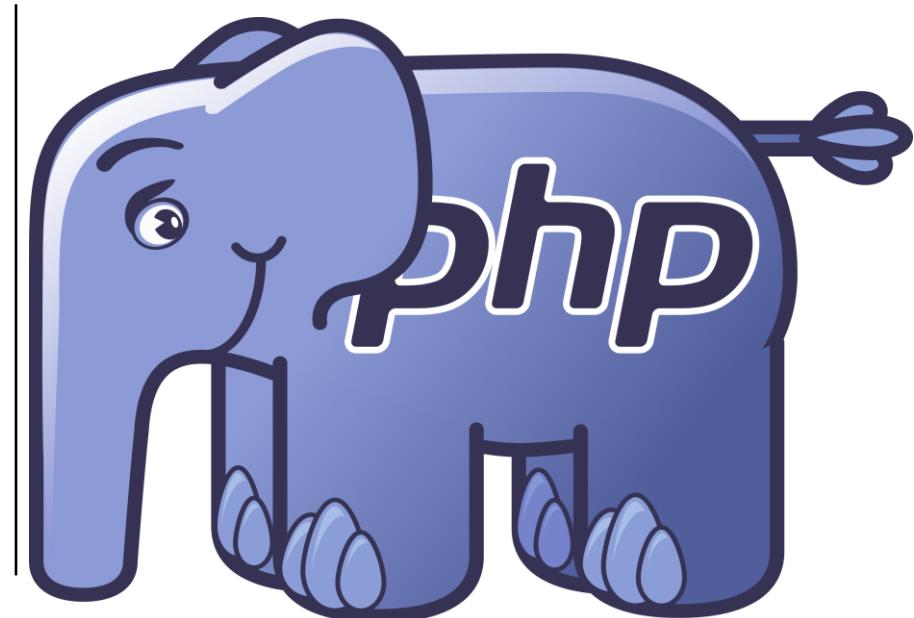
- I. Généralités
- II. Introduction à la programmation
- III. Programmation orientée objet
- IV. Gestion des formulaires
- V. Gestion des fichiers
- VI. Gestion d'une base de données
SQL
- VII. Gestion des sessions et des cookies
- VIII. Nouveautés PHP8

I. GÉNARALITÉS

Qu'est-ce que PHP ?

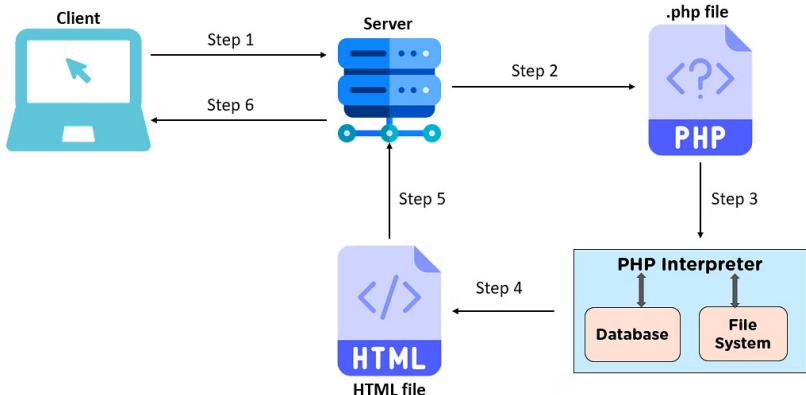
- PHP : à l'origine *Personal Home Page* puis *PHP Hypertext PreProcessor*
- Crée en **1995** par Rasmus Lerdorf
- Caractéristiques du langage
 - Interprété et langage de script
 - **Open Source**
 - Supporte la Programmation Orientée Objet (POO)
 - Initialement faiblement typé
- Dédié principalement aux applications Web
- Autres applications
 - Ligne de commandes
 - Applications graphiques
- Courbe d'apprentissage facile et rapide

[**Source image wikipedia**](#)



Architecture client/serveur

[Source image simplilearn](#)



- Client
 - Machine ou application demandeur de service
- Serveur
 - Machine ou application fournisseur de service qui met à disposition des ressources
- Communication s'effectue à travers le réseau Internet avec le protocole HTTP

Protocole HTTP

Requête

- Protocole dédié au Web
- Les méthodes
 - **GET** : demande d'une ressource
 - **POST** : envoie d'une ressource
 - **PUT** : demande de modification d'une ressource
 - **DELETE** : suppression d'une ressource

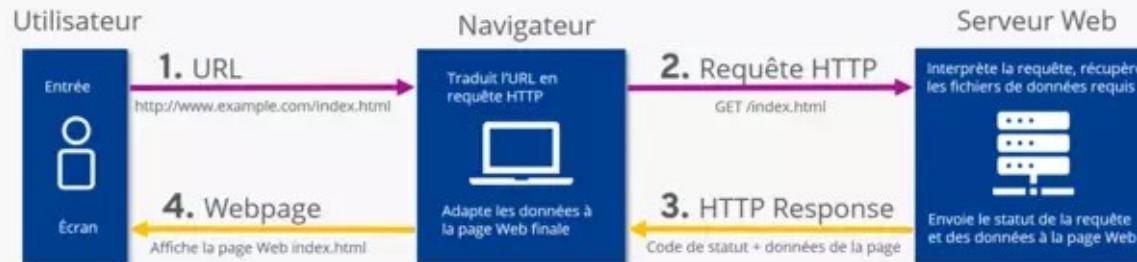
Réponse

Les réponses sont regroupées en cinq classes :

- **1xx** : les réponses informatives
- **2xx** : les réponses de succès
 - 200 OK
- **3xx** : les redirections
 - 304 NOT MODIFIED
- **4xx** : les erreurs du client
 - 403 FORBIDDEN
 - 404 NOT FOUND
- **5xx** : les erreurs du serveur
 - 500 INTERNAL ERROR

Protocole HTTP : Hypertext Transfer Protocol

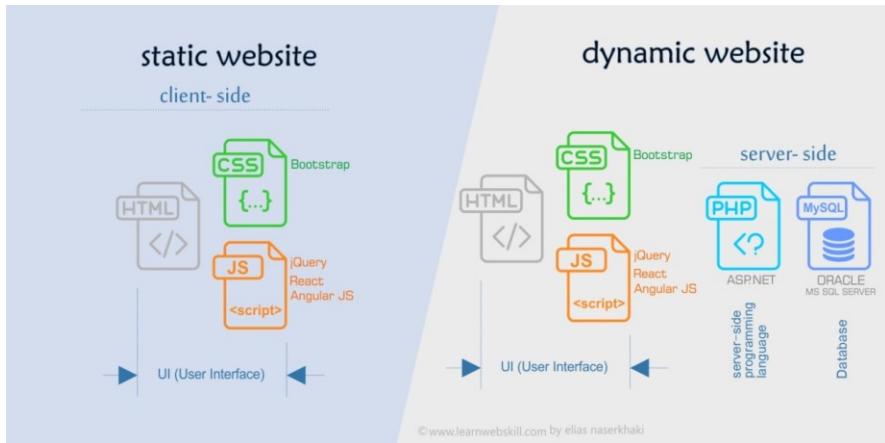
Processus de communication HTTP



IONOS

Sites statiques vs dynamiques

[Source image besolve.in](#)



- Sites statiques
 - HTML/CSS/JS
 - Même page pour tout le monde
 - Intervention humaine pour faire des modifications au niveau du contenu
- Sites dynamiques
 - HTML/CSS/JS
 - + un **langage côté serveur** comme PHP qui génère des pages HTML personnalisées
 - + une base de données

INSTALLATIONS

INSTALLATION

README.md

II. INTRODUCTION À LA PROGRAMMATION

Les bases

Balises

```
<?php // balise d'ouverture pour écrire du PHP
// Commentaire sur une ligne = ligne non interprétée
# Autre façon d'écrire un commentaire sur une ligne (adaptée aux annotations qu'on verra plus tard)
/**
 * Commentaire
 * Sur
 * plusieurs lignes
 */
?>
```

- Les commentaires
 - // Ceci est un commentaire sur une ligne (ligne non interprétée par PHP)
 - # Aussi un commentaire sur une ligne
 - /* Commentaire sur Plusieurs lignes */
- Les balises PHP
- Ouverture : <?php
- Fermeture : ?>
- Instructions
 - Chaque instruction (ordre au langage) doit se terminer par un point-virgule ;

Affichage

```
<?php  
echo '<h1>PHP</h1>';  
print '<p>hello world</p>';  
  
var_dump(  
    array('h1', 'h2', 'h3', 'h4', 'h5', 'h6')  
);  
print_r(array(1,2,3,4,5,6));  
?>
```

PHP

hello world

```
F:\formations'  
array (size=6)  
  0 => string 'h1' (Length=2)  
  1 => string 'h2' (Length=2)  
  2 => string 'h3' (Length=2)  
  3 => string 'h4' (Length=2)  
  4 => string 'h5' (Length=2)  
  5 => string 'h6' (Length=2)  
  
Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5 [5] => 6 )
```

- echo pour afficher du texte
- print est similaire à echo, cependant il est moins utilisé
- var_dump pour afficher du contenu plus complexe comme des objets ou des tableaux
- print_r similaire à var_dump

Variables, opérateurs et tableaux

Variables et constantes

```
<?php  
  
// Variable  
$firstName = 'Glodie';  
echo $firstName;  
echo '<br>';  
  
// Constante  
define('PI', 3.14);  
echo PI;
```

Glodie
3.14

- Une information stockée en mémoire
- Pour une variable, la donnée peut varier au cours de l'exécution
- Pour une constante, la donnée ne varie pas et elle est définie avec la fonction [define\(\)](#)
- Pour déclarer une variable, il faut :
 - Préfixer avec \$
 - Un *nom*
 - Un *opérateur d'affectation* =
 - Une *valeur*
- Bonnes pratiques de nommage des variables
 - En **anglais**
 - Pas d'espace entre les mots
 - Respecter les **conventions de nommage** (standards) de l'entreprise ou de la communauté

Convention de nommage

Les conventions ci-dessous varient en fonction des langages de programmation et des entreprises

camelCase

Générale-
ment (Génr)
pour les
variables et
fonctions

PascalCase

Génr. pour
les noms des
classes

UPPERCASE &
UPPER_SNAKE_
CASE

Génr. pour
les variables
globales et
constantes

kebab-case

Génr. pour
les noms des
ressources
Web, les
classes CSS,
etc.

snake_case

Génr. pour
les
variables,
fonctions et
les noms des
attributs des
tables en
base de
données

variables et types de données

Type	Ensemble des valeurs que peut prendre une variable
<u>string</u>	Chaines de caractères, autrement dit un ou plusieurs caractères alphanumériques
<u>int</u>	Nombres entiers
<u>float</u>	Nombres décimaux
<u>bool</u>	Boolean (booléen), <i>true</i> (vrai) ou <i>false</i> (faux)
<u>array</u>	Tableaux
<u>iterable</u>	Tableaux ou une instance de l'interface <i>Traversable</i>
<u>callable</u>	Fonctions de rappel
nom de classe / d'interface	Instances de la classe ou de l'interface donnée
self	Instances de la même classe
<u>object</u>	<i>Object</i> (objet), type générique pour tous les objets

Modification de types :caster une variable

```
$age = '32';
var_dump($age);
// transtypage
$age = intval($age);
var_dump($age);
```

```
F:\formations\                                \cours\demo\index.php:34:string '32' (Length=2)
F:\formations\                                \cours\demo\index.php:37:int 32
```

- Affecter un autre type à une variable
- Affectation via
 - Les parenthèses contenant le type souhaité juste avant la valeur
 - Les fonctions pour *caster*
 - [intval\(\)](#)
 - [floatval\(\)](#)
 - [strval\(\)](#)
 - [settype\(\)](#)

TABLEAUX



Tableaux indexés

```
$languages = [  
    'PHP',  
    'JavaScript',  
    'Python',  
    'C',  
    'C++',  
    'C#',  
    'JAVA'  
];  
  
var_dump($languages);  
  
array (size=7)  
  0 => string 'PHP' (length=3)  
  1 => string 'JavaScript' (Length=10)  
  2 => string 'Python' (Length=6)  
  3 => string 'C' (Length=1)  
  4 => string 'C++' (Length=3)  
  5 => string 'C#' (Length=2)  
  6 => string 'JAVA' (length=4)
```

- Une variable qui contient plusieurs valeurs
- Un tableau est construit avec une association entre une clé (position ou index ou indice) unique et une valeur
- Un tableau indexé est un tableau dont les clés sont numériques. On parle également de tableau indexé numérique
- Les valeurs
 - peuvent être de tout type
 - Sont séparées par des virgules
- Création d'un tableau via
 - La fonction [array\(\)](#)
 - Ou []
- L'index commence par 0 (premier élément du tableau)

Tableaux associatifs

```
$user = array(  
    'firstname' => 'Glodie',  
    'lastname' => 'Tshimini',  
    'age' => 32,  
    'country' => 'France'  
)  
  
var_dump($user);
```

```
F:\formations\coderbase\m2i\php8-18032024\cours\demo\index.php:58:  
array (size=4)  
    'firstname' => string 'Glodie' (Length=6)  
    'lastname' => string 'Tshimini' (Length=8)  
    'age' => int 32  
    'country' => string 'France' (Length=6)
```

- Similaire au tableau indexé, seule différence, les **clés** sont des **chaînes de caractères uniques**
- Avantages
 - Apporte plus de clarté sur les valeurs stockées
 - Plus pratique
- Création identique aux tableaux numériques vus précédemment

Opérateurs

|

OPÉRATEURS

```
$nb1 = 10;
$str1 = '10';

var_dump($nb1 + $str1); // 20
var_dump($nb1 >= intval($str1)); // true
var_dump($nb1 == $str1); // true
var_dump($nb1 === $str1); // false
$nb1++;
$nb1++;
var_dump($nb1); // 12
var_dump($nb1--); // 12
var_dump($nb1 *= 4); // 11 * 4 = 44
var_dump(1 === 1 && 1 < 5); // true
var_dump(1 != 1 || 5 > 4); // true
var_dump(!false); // true
var_dump (!!true); // true
```

- Opérateur d'**affectation** ou d'**assignation** : =
- Les opérateurs arithmétiques (**mathématiques**) : +, -, /, % (modulo = reste d'une division entière)
- Les opérateurs de **comparaison** : égalité (==), différence (!=), majeur (>) ou mineur (<)
- Les opérateurs **logiques** : AND (&), OR (||), NOT (!)
- Opérateur de **concaténation** : . (point)
- Opérateurs d'**incrémantation** : ++
- Opérateurs de **décrémentation** : --
- Opérateurs combinés : +=, -=, *=, .= (concatène les valeurs)

Démo mise en place d'une page web avec PHP

Structures de contrôle

Structure If..elseif..else

```
$age = 9;  
if($age >= 10 && $age < 18) {  
    echo 'Adolescent';  
} elseif( $age < 10 ) {  
    echo 'Enfant';  
} else {  
    echo 'Adulte';  
}
```

- Structure de test aussi appelée structure de condition, condition ou structure de contrôle
- Exécution d'une **seule instruction** de la structure parmi les différentes « branches » possibles
- **L'ordre des instructions à une importance**
- La première instruction qui vérifie la condition sera exécutée (pas les autres)
- **Une condition est évaluée à vrai ou faux**
- Les structures de contrôle peuvent s'imbriquer, en théorie, il n'y a pas de limites dans l'imbrication (code spaghetti), en pratique, le code est moins lisible et la solution moins optimisée

Structure switch

```
$city = 'Lyon';
✓ switch($city) {
✓   case 'paris':
|   echo 'France';
break;
case 'madrid':
case 'barcelona':
|   echo 'Espagne';
break;
default:
|   echo 'Autre';
break;
}
```

- Identique à *if*, plus lisible pour des vérifications à effectuer sur le contenu d'une chaîne de caractères
- L'instruction **break** permet de **sortir de la structure conditionnelle** lorsqu'un cas est évalué à **true** et que les instructions présentent entre case et break ont été exécutées
- Plus performant au niveau de l'exécution du programme

Opérateur ternaire

- Notation raccourcie d'un *if..else*
- Les ternaires peuvent être imbriquées, cependant le code est beaucoup moins lisible

```
$age = '17';
echo $age > 18 ? 'Majeur' : 'Mineur';
var_dump($age === 17 ? 'Exactement 17 ans' : 'Un autre age');
```

Opérateurs de comparaison

Les opérateurs de comparaison permettent de comparer deux valeurs

Exemple	Nom	Résultat
<code>\$a == \$b</code>	Egal	<i>True</i> si \$a est égal à \$b sans tenir compte de leur type
<code>\$a === \$b</code>	Strictement égal	<i>True</i> si \$a est égal à \$b et qu'ils sont de même type.
<code>\$a != \$b</code>	Différent	<i>True</i> si \$a est différent de \$b après le transtypage
<code>\$a !== \$b</code>	Différent	<i>True</i> si \$a est différent de \$b ou bien s'ils ne sont pas du même type.
<code>\$a < \$b</code>	Plus petit que	<i>True</i> si \$a est strictement plus petit que \$b
<code>\$a > \$b</code>	Plus grand	<i>True</i> si \$a est strictement plus grand que \$b
<code>\$a <= \$b</code>	Inférieur ou égal	<i>True</i> si \$a est plus petit ou égal à \$b
<code>\$a >= \$b</code>	Supérieur ou égal	<i>True</i> si \$a est plus grand ou égal à \$b
<code>\$a <=> \$b</code>	Combiné	Un entier inférieur, égal ou supérieur à zéro lorsque \$a est inférieur, égal, ou supérieur à \$b respectivement - Disponible à partir de PHP 7

Opérateurs logiques

Comparer et combiner l'évaluation de plusieurs conditions

Opération	Description
[Expr1] and [Expr2]	<i>True</i> si [Expr1] et [Expr2] sont vraies
[Expr1] && [Expr2]	Idem (&& est une autre notation de AND)
[Expr1] or [Expr2]	<i>True</i> si [Expr1] ou [Expr2] sont vraies
[Expr1] [Expr2]	Idem (est une autre notation de OR)
[Expr1] xor [Expr2]	<i>True</i> si [Expr1] ou [Expr2] sont vraies mais pas les deux en même temps
! [Expr1]	<i>True</i> si [Expr1] est non vraie

Structures itératives

Structures itératives (boucles)

- Parcourir un tableau d'éléments en exécutant un bloc de code tant qu'une certaine condition est vérifiée
 1. **FOR** : **compteur ; condition ; incrementation du compteur** {*les instructions à effectuer*};
Boucle adaptée lorsqu'on connaît le nombre d'itération à effectuer
 2. **WHILE** : **while (*condition*)** {*les instructions à effectuer et l'incrémentation selon les cas d'usage*};
Boucle adaptée pour parcourir des éléments lorsqu'on ignore le nombre d'itération à effectuer
 3. **DO WHILE** : **do** {*les instructions à effectuer et l'incrémentation selon les cas d'usage*} **while** {*condition*};
Similaire à while sauf qu'il s'exécute au moins une fois
- Les boucles peuvent **s'imbriquer**, c'est utile notamment pour parcourir un tableau multidimensionnel

Opérateurs d'incrémentation et de décrémentation

Utilisés principalement dans les boucles pour éviter d'itérer sur le même élément ou une boucle infinie (sans fin)

Opérateur	Description
<code>\$a++</code>	Retourne la valeur de <code>\$a</code> puis augmente la valeur de <code>\$a</code> de 1
<code>+\$a</code>	Augmente la valeur de <code>\$a</code> de 1 puis retourne la nouvelle valeur de <code>\$a</code>
<code>\$a--</code>	Retourne la valeur de <code>\$a</code> puis diminue la valeur de <code>\$a</code> de 1
<code>--\$a</code>	Diminue la valeur de <code>\$a</code> de 1 puis retourne la nouvelle valeur de <code>\$a</code>

Structure itérative for

```
for($i = 0; $i < 10; $i++) {  
    echo $i. " : i love PHP<br>";  
}
```

0 : i love PHP
1 : i love PHP
2 : i love PHP
3 : i love PHP
4 : i love PHP
5 : i love PHP
6 : i love PHP
7 : i love PHP
8 : i love PHP
9 : i love PHP

- Traduit par « pour » de 1 à compteur
- 1. Un compteur initialise le début de la boucle.
- 2. Une condition de sortie (évaluation de la condition).
- 3. Incrémentation du compteur (changement de la valeur du compteur).

Structure itérative foreach

```
$numbers = [10,20,30,40,50];
// notation sans tenir compte de la clé
foreach($numbers as $number) {
    echo $number.'  
';
}
echo '-----'.'  
';
// notation clé => valeur
foreach($numbers as $index => $value) {
    echo $index.':'.$value.'  
';
}
```

10
20
30
40
50

0:10
1:20
2:30
3:40
4:50

- PHP propose une déclinaison de la boucle for, plus adaptée aux tableaux associatifs
- Traduit par « pour chaque »
- 2 notations possibles
 1. En tenant compte de la clé et valeur
 2. En prenant en considération uniquement la valeur

Structure itérative while

```
$numbers = [10,20,30,40,50,60];
$find = 50;
$i = 0;
while($numbers[$i] != $find) {
    echo $numbers[$i].'\n\'est pas le nombre cherché!<br>';
    $i++;
}
echo 'On sort de la boucle!';
```

10 n'est pas le nombre cherché!
20 n'est pas le nombre cherché!
30 n'est pas le nombre cherché!
40 n'est pas le nombre cherché!
On sort de la boucle!

- Traduit par « tant que »
- S'exécute tant qu'une condition est respectée.
- Généralement, on utilise un **opérateur d'incrémentation** ou une **condition** dont l'évaluation faudra **faux** pour sortir de la boucle et éviter une boucle infinie.

Structure itérative do while

```
$numbers = [10,20,30,40,50,60];
$find = 50;
$i = 0;
do {
    echo $numbers[$i]. ' n\'est pas le nombre cherché!<br>';
    $i++;
}while($numbers[$i] == $find);
echo 'On sort de la première boucle do while!';
echo '<br>-----<br>';
do {
    echo 'Exécution au moins une fois<br>';
} while(false);
echo 'On sort de la deuxième boucle do while!<br>';
```

10 n'est pas le nombre cherché!

On sort de la première boucle do while!

Exécution au moins une fois

On sort de la deuxième boucle do while!

- Traduit par « faire...tant que » ou « répéter...tant que »
- Similaire à la boucle *While*
- A la différence que la **condition est évalué à la fin** donc les instructions sont **exécutées au moins une fois** peu importe le résultat (*true* ou *false*) de la condition

Break et continue

```
echo '<br>';
for($i = 0; $i < 1000000; $i++) {
    echo $i.'<br>';
    if($i === 5) break;
}
echo '-----<br>';
$i = 5;
while($i > 0) {
    $i--;
    echo $i.'<br>';
    if($i === 3) continue;
}
```

0
1
2
3
4
5

4
3
2
1
0

- L'instruction *break* permet de quitter la structure itérative sans parcourir tous les éléments
- L'instruction *continue* permet de sauter le tour d'un l'élément courant et passer à l'élément suivant du tableau

Fonction

|

Définition

- On utilise des fonctions pour **regrouper des instructions et les appeler sur demande** : chaque fois qu'on a besoin de ces instructions, il suffira d'appeler la fonction au lieu de répéter toutes les instructions.
- Pour accomplir ce rôle, le **cycle de vie d'une fonction comprend 2 phases** :
 1. Une première phase dans laquelle la fonction est **déclarée**
 - On définit à ce stade toutes les instructions qui doivent être groupées pour obtenir le résultat souhaité ainsi que les paramètres nécessaires (variables fournies depuis l'extérieur de la fonction).
 - Dans les paramètres nécessaires, les paramètres optionnels (possèdent une valeur par défaut affectée dans la définition de la fonction) qui doivent être déclarés après les paramètres obligatoires.
 2. Une deuxième phase dans laquelle la fonction est appelée puis **exécutée**, en respectant sa signature (déclaration) et qui peut être répétée une ou plusieurs fois.
- Il existe deux types de fonctions
 - Natives (fournies par PHP)
 - Utilitaires (créer par vous-même ou fourni par une bibliothèque créée par une personne tierce)

Exemples

```
// Définition (déclaration)
function sum($nb1, $nb2) {
| return $nb1 + $nb2;
}
function helloWorld() {
| echo 'Hello world';
}
function hi($name) {
| echo 'Bonjour '.$name;
}
function bye($firstName, $lastName = 'NC') {
| echo 'Aurevoir '.$lastName.' '.$firstName;
}
```

```
// Appel (exécution)
echo sum(10,15);
echo '<br>';
$result = sum(5.7,4.3);
echo $result;
echo '<br>';
helloWorld();
echo '<br>';
hi('Rosa');
echo '<br>';
bye('Rosa');
echo '<br>';
bye('John', 'Doe');
echo '<br>';
```

25

10

Hello world

Bonjour Rosa

Aurevoir NC Rosa

Aurevoir Doe John

Fonctions utiles pour manipuler des nombres

- [abs\(\)](#) : retourne la valeur absolue d'un nombre
- [pow\(\)](#) : retourne un nombre élevé à une puissance d'un autre passé en paramètre
- [sqrt\(\)](#) : retourne la racine carrée d'un nombre
- [min\(\)](#) et [max\(\)](#) : renvoie la plus petite/grande valeur d'un tableau ou d'un ensemble d'arguments
- [floor\(\)](#) et [ceil\(\)](#) : retourne l'entier inférieur/supérieur d'un nombre
- [round\(\)](#) : arrondit un nombre à virgule flottante
- [number_format\(\)](#) : formate un nombre pour l'affichage
- [rand\(\)](#) : retourne un nombre aléatoire

Fonctions utiles dédiées à la manipulation des chaînes de caractères

- [strlen\(\)](#) : calcule la taille d'une chaîne
- [strtolower\(\)](#) : renvoie une chaîne en minuscules
- [strtoupper\(\)](#) : renvoie une chaîne en majuscules
- [ucfirst\(\)](#) : met le premier caractère en majuscule
- [ucwords\(\)](#) : met en majuscule la première lettre de tous les mots
- [strcmp\(\)](#) : comparaison binaire de chaînes
- [strcasecmp\(\)](#) : comparaison insensible à la casse de chaînes binaires
- [sprintf\(\)](#) : retourne une chaîne formatée
- [substr\(\)](#) : retourne un segment de chaîne
- [strpos\(\)](#) : cherche la position de la première occurrence dans une chaîne
- [str_replace\(\)](#) : remplace toutes les occurrences dans une chaîne
- [trim\(\)](#) : supprime les espaces (ou d'autres caractères) en début et fin de chaîne

Fonctions utiles pour les dates et les heures

- [checkdate\(\)](#) : valide une date grégorienne
- [date\(\)](#) : formate une date/heure locale
- [getdate\(\)](#) : retourne un tableau associatif contenant les informations de date et d'heure du timestamp
- [time\(\)](#) : retourne le timestamp UNIX actuel (en secondes depuis le 1^{er} janvier 1970 à 00h00mn00s)
- [mktime\(\)](#) : retourne le timestamp UNIX d'une date
- [setlocale\(\)](#) : modifie les informations de localisation
- [strtotime\(\)](#) : transforme un texte anglais en timestamp

Fonctions utiles pour les tableaux

FIFO, un premier entré, premier sorti

- array_push() : place un ou plusieurs éléments à la fin d'un tableau
- array_pop() : extrait la dernière valeur d'un tableau et la retourne

```
$names = ['alpha','fatima','eric','sarah','aya'];
array_push($names, 'antoine', 'julie');
var_dump(count($names));
array_pop($names);
var_dump($names);
```

```
F:\formations\                                \cours\demo\index.php:188:int  7
F:\formations\                                \cours\demo\index.php:190:
array (size=6)
  0 => string 'alpha' (Length=5)
  1 => string 'fatima' (Length=6)
  2 => string 'eric' (Length=4)
  3 => string 'sarah' (Length=5)
  4 => string 'aya' (Length=3)
  5 => string 'antoine' (Length=7)
```

LIFO, dernier arrivé, premier sorti

- array_unshift() : place un ou plusieurs éléments au début d'un tableau
- array_shift() : extrait la première valeur d'un tableau et la retourne

```
$names = ['alpha','fatima','eric','sarah','aya'];
array_unshift($names, 'antoine', 'julie');
var_dump(count($names));
array_shift($names);
var_dump($names);
```

```
:\\formations\                                \cours\demo\index.php:188:int
:\\formations\                                \cours\demo\index.php:190:
array (size=6)
  0 => string 'julie' (Length=5)
  1 => string 'alpha' (Length=5)
  2 => string 'fatima' (Length=6)
  3 => string 'eric' (Length=4)
  4 => string 'sarah' (Length=5)
  5 => string 'aya' (Length=3)
```

Fonctions utiles tri des tableaux

```
$names = ['alpha','fatima','eric','sarah','aya'];
$namesCopy = $names;
sort($names);
asort($namesCopy);
var_dump($names);
var_dump($namesCopy);
```

```
F:\formations\                                \cours\demo\index.php:196:
array (size=5)
  0 => string 'alpha' (Length=5)
  1 => string 'aya' (Length=3)
  2 => string 'eric' (Length=4)
  3 => string 'fatima' (Length=6)
  4 => string 'sarah' (Length=5)

F:\formations\                                \cours\demo\index.php:197:
array (size=5)
  0 => string 'alpha' (Length=5)
  4 => string 'aya' (Length=3)
  2 => string 'eric' (Length=4)
  1 => string 'fatima' (Length=6)
  3 => string 'sarah' (Length=5)
```

- [sort\(\)](#) : trie un tableau
- [rsort\(\)](#) : trie un tableau en ordre inverse
- [ksort\(\)](#) : trie un tableau suivant les clés
- [krsort\(\)](#) : trie un tableau en sens inverse et suivant les clés
- [asort\(\)](#) : trie un tableau et conserve l'association des index
- [arsort\(\)](#) : trie un tableau en ordre inverse et conserve l'association des index

Autres fonctions associées aux tableaux

- [count\(\)](#) ou [sizeof\(\)](#) : compte tous les éléments d'un tableau ou d'un objet
- [in_array\(\)](#) : indique si une valeur appartient à un tableau
- [array_key_exists\(\)](#) : vérifie si une clé existe dans un tableau
- [array_merge\(\)](#) : fusionne plusieurs tableaux en un seul
- [array_search\(\)](#) : recherche dans un tableau la clé associée à la première valeur
- [unset\(\\$array\[\\$i\]\)](#) : pour supprimer le i^{ème} élément d'un tableau
- [unset\(\\$array\)](#) : pour supprimer le tableau
- [implode\(\)](#) : rassemble les éléments d'un tableau en une chaîne
- [explode\(\)](#) : scinde une chaîne de caractères en segments

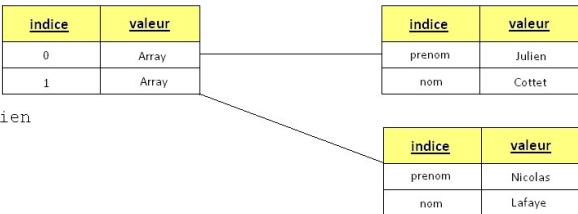
Tableaux multidimensionnels

TABLEAUX MULTIDIMENSIONNELS

Source image eprojet

Tableau Multidimensionnel

```
Array
(
    [0] => Array
    (
        [prenom] => Julien
        [nom] => Cottet
    )
    [1] => Array
    (
        [prenom] => Nicolas
        [nom] => Lafaye
    )
)
```



- Un tableau dont les éléments sont d'autres tableaux.
- Il n'y a pas de limite au niveau du nombre des dimensions, cependant au-delà de 2 dimensions, c'est plus en plus difficile pour les humains de visualiser les informations.
- On parle généralement de tableau à N dimensions, avec N le nombre des dimensions.
- Il faut des *boucles imbriquées* pour parcourir un tableau multidimensionnel

Exemple d'un tableau multidimensionnel

```
$persons = [
    [
        'name' => 'Charles',
        'age' => 32,
        'hobbies' => ['cuisine', 'code', 'finance']
    ],
    [
        'name' => 'John',
        'age' => 37,
        'hobbies' => ['math', 'course', 'trading']
    ],
    [
        'name' => 'Marie',
        'age' => 28,
        'hobbies' => ['mma', 'littérature', 'economie']
    ]
];

var_dump($persons);
```

```
array (size=3)
  0 =>
    array (size=3)
      'name' => string 'Charles' (Length=7)
      'age' => int 32
      'hobbies' =>
        array (size=3)
          0 => string 'cuisine' (Length=7)
          1 => string 'code' (Length=4)
          2 => string 'finance' (Length=7)
  1 =>
    array (size=3)
      'name' => string 'John' (Length=4)
      'age' => int 37
      'hobbies' =>
        array (size=3)
          0 => string 'math' (Length=4)
          1 => string 'course' (Length=6)
          2 => string 'trading' (Length=7)
  2 =>
    array (size=3)
      'name' => string 'Marie' (Length=5)
      'age' => int 28
      'hobbies' =>
        array (size=3)
          0 => string 'mma' (Length=3)
          1 => string 'littérature' (Length=12)
          2 => string 'economie' (Length=8)
```

III. Programmation Orientée Objet

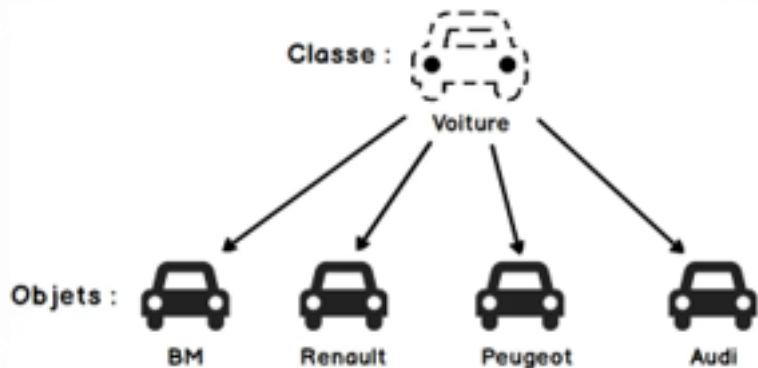
Approche objet

Approche orientée objet

- L'approche **procédurale** consiste à résoudre un problème informatique de manière **séquentielle** avec une suite d'instructions à exécuter et **l'utilisation des fonctions**.
- L'approche **objet** demande une réflexion plus poussée pour concevoir et développer une solution **réutilisable** et **évolutif** (maintenable). De plus, elle garantit une **protection** des données que l'on verra un peu plus tard lorsqu'on abordera la notion **d'encapsulation**.
- Les objets peuvent facilement être modélisés avec les diagrammes de classes et d'objets d'UML (langage graphique de modélisation des systèmes d'information)
- Elle utilise des **objets** qui vont **collaborer** pour résoudre un problème.
- En informatique, un objet est une **entité** qui possède un ensemble **d'attributs** déterminant ses caractéristiques propres et un ensemble de **méthodes** déterminant son comportement.

Classe et objet

[Source image waytolearnx.com](http://waytolearnx.com)



- La **classe** est le modèle permettant de créer un ou plusieurs **objets**.
- On dit alors qu'un **objet** est une instance d'une classe.
- Une **classe** possède :
 1. Un nom
 2. Des attributs (des données)
 3. Des comportements (des actions)

Exemples des instances d'une classe

```
class Car {  
}  
  
$bm = new Car();  
$renault = new Car();  
$peugeot = new Car();  
$audi = new Car();  
  
var_dump($bm, $renault, $peugeot, $audi);
```

F:\formations\
object(Car)[1]

\cours\demo\index.php:228:

F:\formations\
object(Car)[2]

\cours\demo\index.php:228:

F:\formations\
object(Car)[3]

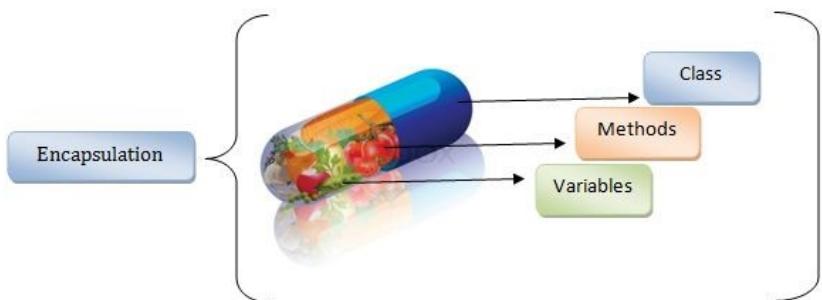
\cours\demo\index.php:228:

F:\formations\
object(Car)[4]

\cours\demo\index.php:228:

Encapsulation

[**Source image logicmojo**](#)



- Capacité d'une classe à contrôler l'accès à ses attributs et comportements.
- Grâce à l'encapsulation, un objet est autonome et indépendant
- **3 niveaux d'encapsulation :**
 - 1. Privé** : attributs et/ou méthodes accessibles uniquement par l'objet lui-même
 - 2. Protégé** : accessibles par l'objet lui-même et ses descendants (classes filles)
 - 3. Public** : accessibles par tout le monde
- De façon générale, les attributs d'un objet sont privés et les méthodes publiques.
- En cas de violation de l'accessibilité d'un attribut ou d'une méthode PHP déclenche une erreur fatale.

Exemple des classes en PHP

Attributs

```
class Car {  
    private $brand;  
    private $engine;  
    private $color;  
}
```

```
$bm = new Car();
```

```
var_dump($bm);
```

```
object(Car)[1]  
private 'brand' => null  
private 'engine' => null  
private 'color' => null
```

Méthodes

```
class Car {  
    private $brand;  
    private $engine;  
    private $color;  
  
    public function start() {  
        $this->engine = true;  
        return 'started';  
    }  
  
    public function stop() {  
        $this->engine = !$this->engine;  
        return 'stopped';  
    }  
}
```

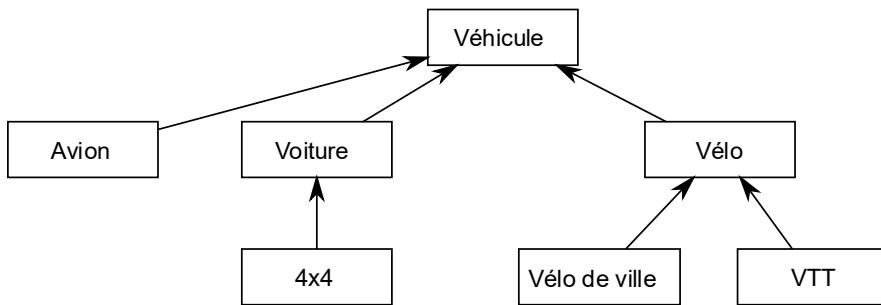
```
$bm = new Car();  
$bm->start();  
var_dump($bm);  
$bm->stop();  
var_dump($bm);
```

```
F:\formations\  
object(Car)[1]  
private 'brand' => null  
private 'engine' => boolean true  
private 'color' => null
```

```
F:\formations\  
object(Car)[1]  
private 'brand' => null  
private 'engine' => boolean false  
private 'color' => null
```

Héritage

Source de l'image perso-esiee



- Une **Voiture** est *classe* donc un *modèle*, lui-même créé à partir d'un autre modèle le **Véhicule**.
- Donc on peut dire qu'une **Voiture** est un **Véhicule**.
- Un **VTT** hérite des attributs et des comportements **publics** et **protégés** d'un **Vélo** qui lui-même hérite du **Véhicule**
- Autrement dit, un **VTT** hérite des caractéristiques d'un Vélo et d'un Véhicule
- Un Véhicule regroupe les caractéristiques communes des Avion, Voiture et Vélo
- Dans le cadre de l'héritage, les attributs deviennent protégés uniquement si elles doivent être accessibles par les classes filles

Exemple de l'héritage

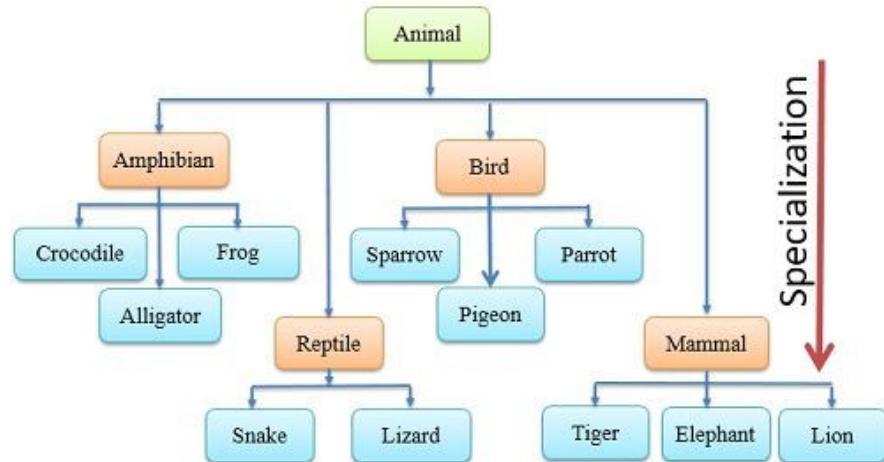
Mot-clé `extends` pour l'héritage

```
class Vehicule {  
    protected $brand;  
    protected $engine;  
    protected $color;  
    public function start() {}  
    public function stop() {}  
}  
  
class Car extends Vehicule {  
    private $registrationNumber;  
    private $year;  
    private function getVin() {}  
}  
  
class Bicycle extends Vehicule {  
    private $speedNumbers;  
    private $materials;  
}  
  
class Vtt extends Bicycle {  
}  
  
$audi = new Car();  
$vtt = new Vtt();  
var_dump($audi, $vtt);
```

```
F:\formations\object(Car)[1]          \cours\demo\index.php:281:  
private 'registrationNumber' => null  
private 'year' => null  
protected 'brand' => null  
protected 'engine' => null  
protected 'color' => null  
  
F:\formations\object(Vtt)[2]          \cours\demo\index.php:281:  
private 'speedNumbers' (Bicycle) => null  
private 'materials' (Bicycle) => null  
protected 'brand' => null  
protected 'engine' => null  
protected 'color' => null
```

Généralisation et spécialisation

[Source image letsstudytogether](#)



- D'une part, un *Serpent* est une *spécialisation* d'un *Reptile*.
- D'autre part, un Reptile est une généralisation d'un Serpent, Lézard, etc.
- La classe *Reptile* est appelée **classe mère** ou **superclasse**, car elles possèdent des caractéristiques et comportements communs à un *Serpent*, *Lézard*, etc.
- *Reptile*, *Mammifères*, *Oiseaux*, *Amphibien* sont des spécialisations de la classe *Animal*. Elles sont appelées **sous-classe** ou **classes filles**.
- Utilisation du mot-clé *extends* pour étendre une classe mère

Exemple de l'utilisation des attributs et méthodes parentes

parent:: permet d'accéder aux attributs ou méthodes de la classe parente

```
class Vehicule {  
    protected $brand;  
    protected $engine;  
    protected $color;  
    public function start() {  
        $this->engine = true;  
    }  
    public function stop() {}  
}  
  
class Bicycle extends Vehicule {  
    private $speedNumbers;  
    private $materials;  
}  
  
class Vtt extends Bicycle {  
    public function start() {  
        parent::start();  
        return 'Action sur les pédales ok';  
    }  
}  
  
$vtt = new Vtt();  
echo $vtt->start();  
var_dump($vtt);
```

Action sur les pédales ok

```
object(Vtt)[1]  
private 'speedNumbers' (Bicycle) => null  
private 'materials' (Bicycle) => null  
protected 'brand' => null  
protected 'engine' => boolean true  
protected 'color' => null
```

Classes abstraites et interfaces

Classe abstraite

```
class Vehicule {  
    protected $brand;  
    protected $engine;  
    protected $color;  
    public function start() {  
        $this->engine = true;  
    }  
    public function stop() {}  
}  
  
abstract class Bicycle extends Vehicule {  
    protected array $features;  
    public abstract function setFeatures(array $features);  
}  
  
class Vtt extends Bicycle {  
    public function setFeatures(array $features)  
    {  
        $this->features = $features;  
    }  
}  
  
$vtt = new Vtt();  
$vtt->setFeatures(['léger', 'grandes roues', 'guidon droit', 'vitesse']);  
var_dump($vtt);  
// ne peut pas être instancié => erreur fatale = le script s'arrête  
$bicycle = new Bicycle();  
var_dump($bicycle);
```

```
object(Vtt)[1]  
    protected array 'features' =>  
        array (size=4)  
            0 => string 'léger' (Length=6)  
            1 => string 'grandes roues' (Length=13)  
            2 => string 'guidon droit' (Length=12)  
            3 => string 'vitesse' (Length=7)  
    protected 'brand' => null  
    protected 'engine' => null  
    protected 'color' => null
```

(!) Fatal error: Uncaught Error: Cannot instantiate abstract class

Bicycle in F:

\formations\██████████ cours\demo\index.php on line
281

(!) Error: Cannot instantiate abstract class Bicycle in F:

\formations\██████████ cours\demo\index.php on line
281

Call Stack

#	Time	Memory	Function	Location
1	0.0014	404368	{main}()	...\index.php:0

Interface

- Une structure **similaire à une classe abstraite**, à la différence que **toutes les méthodes sont abstraites**
- C'est un **contrat** à remplir, c'est-à-dire une classe qui implémente (utilise) une interface doit définir toutes les méthodes de l'interface
- Des objets qui n'ont pas des caractéristiques communes (n'héritant pas des mêmes classes) peuvent implémenter une interface pour étendre leurs fonctionnalités.
- **Plus flexible** qu'une classe abstraite et plus facile pour la maintenance et l'évolution

```
interface crud {  
    function create(mixed $element) : void;  
    function read() : mixed;  
    function update(mixed $element) : bool;  
    function delete(mixed $element) : bool;  
}
```

```
class Customer implements crud {}      'Customer' does not implement methods 'create', 'read', 'update', 'delete'  
class Product implements crud {}       'Product' does not implement methods 'create', 'read', 'update', 'delete'  
class Cart implements crud {}         'Cart' does not implement methods 'create', 'read', 'update', 'delete'  
class Payment implements crud {}       'Payment' does not implement methods 'create', 'read', 'update', 'delete'  
class Delivery implements crud {}     'Delivery' does not implement methods 'create', 'read', 'update', 'delete'  
class Address implements crud {}      'Address' does not implement methods 'create', 'read', 'update', 'delete'  
class Category implements crud {}     'Category' does not implement methods 'create', 'read', 'update', 'delete'
```

Exemples de l'implémentation d'une interface

```
interface features {
    function setFeatures(array $features);
}

class Vtt implements features {
    private array $features;
    public function setFeatures(array $features)
    {
        $this->features = $features;
        return $this->features;
    }
}

class Feature {
    public array $list;
    public function add(string $element = '') : array {
        $this->list[] = $element;
        return $this->list;
    }
}

class Product implements features {
    private array $list;
    public function setFeatures(array $features)
    {
        $feature = new Feature();
        foreach($features as $element) {
            $feature->add($element);
        }
        $this->list = $feature->list;
        return $this->list;
    }
}
```

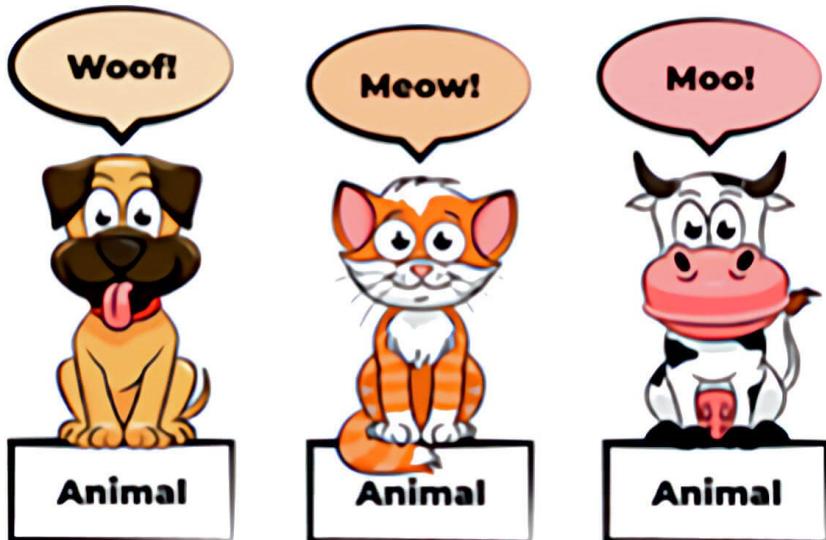
```
$vtt = new Vtt();
$phone = new Product();
$phone->setFeatures(['5g', 'tactile', 'appareil photo', 'gps']);
$vtt->setFeatures(['léger', 'grandes roues', 'guidon droit', 'vitesse']);
var_dump($vtt, $phone);
```

```
object(Vtt)[1]
private array 'features' =>
array (size=4)
    0 => string 'léger' (Length=6)
    1 => string 'grandes roues' (Length=13)
    2 => string 'guidon droit' (Length=12)
    3 => string 'vitesse' (Length=7)
```

```
object(Product)[2]
private array 'list' =>
array (size=4)
    0 => string '5g' (Length=2)
    1 => string 'tactile' (Length=7)
    2 => string 'appareil photo' (Length=14)
    3 => string 'gps' (Length=3)
```

Polymorphisme

polymorphisme animal



- Une méthode peut avoir plusieurs formes
- Autrement dit le comportement « *crier* » d'un *Animal* peut prendre plusieurs formes.
- Le polymorphisme se manifeste par une réécriture de la méthode dans la classe fille
- Une de manière d'implémenter le polymorphisme est d'utiliser les classes abstraites et les interfaces en rendant les méthodes plus abstraites au niveau des classes mères et plus spécifiques au niveau des classes filles

Exemples du polymorphisme

```
class Animal {
    protected function yell() {
        echo 'Yell';
    }
}
class Dog extends Animal {
    public function yell() {
        echo 'Wooooof!<br>';
    }
}
class Cat extends Animal {
    public function yell() {
        echo 'Meow!<br>';
    }
}
class Cow extends Animal {
    public function yell() {
        parent::yell();
        echo ' Moooooo!';
    }
}

$cat = new Cat();
$cat->yell();
(new Dog)->yell();
(new Cow())->yell();
```

Meow!
Wooooof!
Yell Moooooo!

Méthodes



Méthodes magiques

- Méthodes spéciales réservées à PHP pour effectuer certaines opérations sur un objet
- Ses méthodes commencent par __ (2 underscores (tiret du 8))
- Quelques exemples
 - __construct()
 - __destruct()
 - __tostring()
 - Etc.
- [Documentation sur les méthodes magiques](#)

Constructeur

```
class Product {  
    private string $name;  
    private string $desc;  
    private string $qty;  
    public function __construct(string $name, string $desc, int $qty) {  
        $this->name = $name;  
        $this->desc = $desc;  
        $this->qty = $qty;  
    }  
}  
  
$printer = new Product('Evolis', 'imprimante laser', 15);  
var_dump($printer);
```

```
object(Product)[1]  
private string 'name' => string 'Evolis' (Length=6)  
private string 'desc' => string 'imprimante laser' (Length=16)  
private string 'qty' => string '15' (Length=2)
```

- Un constructeur est la méthode magique permettant **d'initialiser un objet à sa création.** Autrement dit instancier une nouvelle instance d'une classe en lui fournissant les données nécessaires à travers cette méthode
- La méthode se nomme ***__construct()*** est automatiquement appelée lorsque on utilise le mot-clé new suivi du nom de la classe
- Le constructeur peut prendre 0, un ou plusieurs paramètres
- Le constructeur ne doit pas effectuer de la logique ou des tâches lourdes.

Destructeur

```
class Product {  
    private string $name;  
    private string $desc;  
    private string $qty;  
    public function __construct(string $name, string $desc, int $qty) { ... }  
  
    public function __destruct()  
    {  
        echo 'Objet détruit';  
    }  
}  
  
$printer = new Product('Evolis', 'imprimante laser', 15);  
unset($printer);  
var_dump($printer);
```

Objet détruit

(!) Warning: Undefined variable \$printer in F:\formations\cours\demo\index.php on line 346

Call Stack

#	Time	Memory	Function	Location
1	0.0004	366152	{main}()	...\\index.php:0

F:\formations\

cours\\demo\\index.php:346:null

- La méthode magique `__destruct()` permet d'effectuer certaines opérations juste avant la destruction de l'objet en libérant de l'espace mémoire par exemple (fermetures des ressources telles que fichier, base de données, etc.)
- Elle est automatiquement appelée :
 - à la fin du script PHP
 - ou à l'appel de la fonction `unset()` sur l'objet

Accesseurs et mutateurs

- Des méthodes publiques qui permettent de lire (accesseur) ou de modifier (mutateur) la valeur d'un attribut en respectant le principe de l'encapsulation
- Pour les accesseurs ou getters
 - La méthode est préfixée par « get » suivi du nom de la propriété en utilisant la convention de nommage camelCase
- Pour les mutateurs ou setters
 - Préfixée par « set » suivi du nom de la propriété toujours en camelCase

Exemples des accesseurs et mutateurs

```
class Product {
    public function __construct(private string $name, private string $desc, private int $qty) {}
    public function getName()
    {
        return $this->name;
    }
    public function setName(string $name)
    {
        $this->name = $name;
        return $this;
    }
    public function getDesc()
    {...}
    public function setDesc(string $desc)
    {...}
    public function getQty()
    {...}
    public function setQty(int $qty)
    {...}
}
```

```
object(Product)[1]
private string 'name' => string 'Evolis' (Length=6)
private string 'desc' => string 'imprimante laser' (Length=16)
private int 'qty' => int 20
```

(!) Fatal error: Uncaught Error: Cannot access private property
Product::\$qty in F:
\formations\██████████\cours\demo\index.php on line
363

(!) Error: Cannot access private property Product::\$qty in F:
\formations\██████████\cours\demo\index.php on line
363

Call Stack

#	Time	Memory	Function	Location
1	0.0011	366192	{main}()	...index.php:0

```
$printer = new Product('Evolis', 'imprimante laser', 15);
$printer->setQty(20);
var_dump($printer);
$printer->qty = 30; // erreur, la propriété privée qty ne peut-être modifiée que par l'intermédiaire de son setter
var_dump($printer); // non exécuté à cause de l'erreur fatale précédente
```

Méthodes

```
class Product {  
    public function __construct(private string $name, private string $desc, private int $qty) {}  
    public function isNew() {}  
    public function search() {}  
    public function getProducts() {}  
    public function getCategories() {}  
    public function convertPrice() {}  
    public function checkAccess() {}  
    public function checkQty() {}  
    public function duplicate() {}  
    public function getName() {}  
    { ...  
    }  
    public function setName(string $name) {}  
    { ...  
    }  
    public function getDesc() {}  
    { ...  
    }  
    public function setDesc(string $desc) {}  
    { ...  
    }  
    public function getQty() {}  
    { ...  
    }  
    public function setQty(int $qty) {}  
}
```

- En-dehors des accesseurs et mutateurs, une classe peut avoir d'autres méthodes
- Ses méthodes remplissent des objectifs divers et variés
- Par exemple
 - Propre fonctionnement de l'objet
 - Communication avec les autres objets
 - Utilitaire
 - Responsabilité
 - Etc.

Propriétés et méthodes statiques

```
class Circle {  
    /**  
     * propriété statique immuable (ne change pas)  
     * constante de classe  
     */  
    private const PI = 3.14159;  
    private static int $count = 0; // propriété statique variable  
    // méthode magique pour construire un objet  
    public function __construct(private float $diameter) {  
        self::$count++;  
    }  
    // méthode d'instance, propre à chaque objet  
    public function perimeter() {  
        return self::PI * $this->diameter;  
    }  
    // méthode statique = méthode de classe  
    public static function total() {  
        return self::$count;  
    }  
  
    $c1 = new Circle(20);  
    $c2 = new Circle(10);  
    $c3 = new Circle(5);  
    var_dump($c1, $c2, $c3, $c1->perimeter(), Circle::total());  
  
object(Circle)[1]  
    private float 'diameter' => float 20  
  
object(Circle)[2]  
    private float 'diameter' => float 10  
  
object(Circle)[3]  
    private float 'diameter' => float 5  
  
F:\formations\cours\demo\index.php:425:float 62.8318  
F:\formations\cours\demo\index.php:425:int 3
```

- Une propriété de classe ou méthode de classe fait appel à la propriété ou à la méthode **directement de la classe** sans devoir instancier un objet
- Autrement dit la propriété ou la méthode **ne dépend pas de l'objet** mais uniquement de la classe
- Propriétés ou méthodes de classe sont appelées propriétés ou méthodes **statiques**
- Autrement dit, elles seront appelées par l'intermédiaire de la classe, le contexte **\$this n'existe plus** dans ce cas de figure
- A la place le mot-clé **self** est utilisé

Gestion des erreurs

Qu'est-ce qu'une exception ?

```
class Math {  
    public function divide (float $nb1, float $nb2) {  
        return $nb1 / $nb2;  
    }  
}  
echo 'Avant l\'exception';  
$op = new Math();  
var_dump($op->divide(10, 0), $op);  
echo 'Ne sera jamais affiché à cause de l\'erreur fatale à la ligne précédente';
```

Avant l'exception

(!) Fatal error: Uncaught DivisionByZeroError: Division by zero
in F:\formations\cours\demo\index.php on line 430

(!) DivisionByZeroError: Division by zero in F:\formations\cours\demo\index.php on line 430

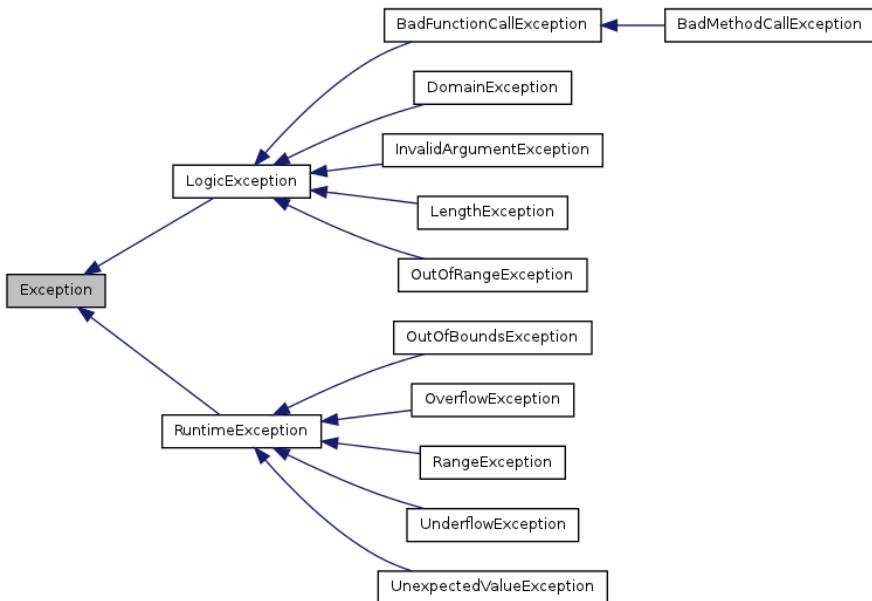
Call Stack

#	Time	Memory	Function	Location
1	0.0004	370272	{main}()	...\index.php:0
2	0.0005	370496	Math->divide(\$nb1 = 10, \$nb2 = 0)	...\index.php:435

- Une exception est une **erreur** survenue lors de l'exécution du script
- L'erreur selon sa criticité peut stopper la suite de l'exécution du script. On parle alors d'une **erreur fatale** dans le cas de l'interruption du script
- Les exceptions peuvent être **déclenchées** dans le programme par le développeur pour des **scénarios** ou **conditions** dont la finalité est une erreur, en utilisant le mot-clé ***throw***
- Une exception capturée et traitée évite que le programme s'arrête subitement

Différents types d'exception

Source image stackoverflow



- Les exceptions peuvent être mineures (notice ou warning) ou critiques (fatal)
- Les erreurs critiques, s'ils ne sont pas gérées (capturées) bloqueront la suite du script
- Toutes les classes filles des exceptions spécialisent une exception via l'héritage
- La classe `Exception` possède les méthodes suivantes parmi tant d'autres :
 - [`getMessage\(\)`](#) retourne le message d'erreur défini lors du lancement de l'exception
 - [`getCode\(\)`](#) retourne le code d'erreur défini lors du lancement de l'exception
 - [`getFile\(\)`](#) retourne le chemin du fichier depuis lequel l'exception a été lancée
 - [`getLine\(\)`](#) retourne la ligne du fichier depuis lequel l'exception a été lancée

Traitement des exceptions

```
class Math {  
    public function divide (float $nb1, float $nb2) {  
        return $nb1 / $nb2;  
    }  
}  
echo 'Avant 1\'exception';  
$op = new Math();  
try {  
    var_dump($op->divide(10, 0), $op);  
} catch(DivisionByZeroError $e) {  
    var_dump($e->getMessage());  
} finally {  
    echo 'Sera affiché peu importe le scénario (erreur ou non)';  
}
```

Avant l'exception

F:\formations\

cours\demo\index.php:438:string 'Division by zero'

Sera affiché peu importe le scénario (erreur ou non)

1. On entoure la méthode susceptible de déclencher une exception avec le *bloc try*
2. On capture l'exception dans le *bloc catch* pour la traiter
3. On peut éventuellement utiliser le bloc *finally* qui sera exécuté peu importe le scénario (erreur ou non)
 - Dans le programme pour déclencher une erreur comme vu précédemment, on utilisera le mot-clé *throw* suivi d'une instance d'un objet de type *Exception*

Exemple de l'utilisation des blocs try/catch/finally

```
class Customer {
    public function __construct(private int $age){}
    public function isAdult() {
        if($this->age < 18) {
            throw new Exception('Only for adult');
        }
    }
}
try {
    $user = new Customer(17);
    $user->isAdult();
} catch(Exception $e) {
    $page = 'http://localhost:9001/index.php?page=signup&error=';
    $page .= urlencode($e->getMessage());
    header('Location: '.$page); // redirection vers la page d'inscription avec le paramètre error qu'on peut récupérer avec $_GET
    die(); // die Interrompt le programme
} finally { // finally ne sera pas exécuté à case du die() précédent
    header("Location: board.php");
}
```

Édition Affichage Historique Marque-pages Outils Aide

Authentification



→ C ⌘



localhost:9001/index.php?page=signup&error=Only+for+adult

60 % ☆



Inscription

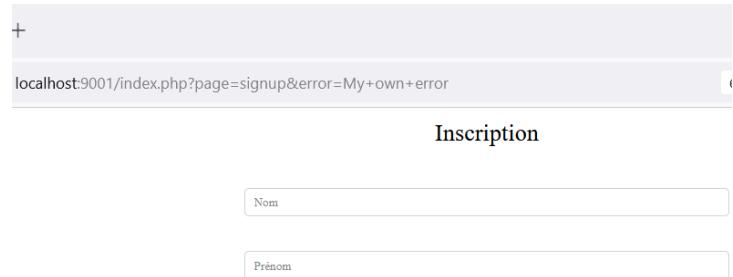
Nom

Prénom

Exemple d'une exception personnalisée

- On peut créer une classe qui étend la classe Exception ou ses sous-classes pour implémenter ses propres exceptions

```
class Customer {  
    public function __construct(private int $age){}  
    public function isAdult() {  
        if($this->age < 18) throw new MyException('Only for adult');  
    }  
}  
  
class MyException extends Exception {  
    private const MESSAGE = 'My+own+error';  
    private static $page = 'http://localhost:9001/index.php?page=signup&error=';  
    public function ban(string|null $page = null) : void {  
        /*valeur de page si elle n'est pas nulle(donnée en paramètre)  
         * sinon ça sera la valeur de la variable de classe $page concaténée avec la constante MESSAGE  
        */  
        $page = $page??self::$page.self::MESSAGE;  
        die(header('Location: '.$page));  
    }  
}  
  
$user = new Customer(17);  
try {$user->isAdult();}  
catch(MyException $e) {  
    $e->ban();  
} finally { // si MyException a lieu, finally ne sera pas exécuté à case du die() dans la méthode ban()  
}
```



Quelques types d'erreurs les plus rependues

Constante	Signification
E_ERROR	Affichage par défaut de l'erreur fatale avec interruption brusque de la suite du script
E_WARNING	Affichage par défaut de l'erreur d'alerte sans interrompre la suite du script
E_NOTICE	Pas d'affichage par défaut d'une potentielle erreur rencontré dans l'exécution du script
E_CORE_ERROR	Similaire à E_ERROR cependant générée par le code source de PHP
E_COMPILE_ERROR	Similaire à E_ERROR cependant générée par le moteur de PHP
E_ALL	Toutes les erreurs, alertes et notices supportées dans votre environnement d'exécution

[Documentation de la liste exhaustive de toutes les constantes prédéfinies des erreurs](#)

IV. Gestion des formulaires

Superglobales

Généralités

- Les **superglobales** sont des variables prédéfinies en PHP et disponibles partout quel que soit le contexte du script (fonction, classe, formulaire, etc.)
- Elles couvrent de nombreux domaines
 - Variables externes
 - Variables d'environnement PHP
 - Protocole HTTP
 - Etc.

- **\$GLOBALS** : référence toutes les variables disponibles dans un contexte global
- **\$ SERVER** : variables de serveur et d'exécution
- **\$ ENV** : variables d'environnement
- **\$ GET** : variables HTTP GET
- **\$ POST** : variables HTTP POST
- **\$ COOKIE** : cookies HTTP
- **\$ REQUEST** : variables de requête HTTP
- **\$ FILES** : variable de téléchargement de fichier via HTTP
- **\$ SESSION** : variables de session

Soumission d'un formulaire côté HTML

```
<form  
    method="POST"  
    action="form_submit.php"  
>  
    <input  
        type="email"  
        name="email"  
        placeholder="email"  
    >  
    <input  
        type="password"  
        name="password"  
        placeholder="mot de passe"  
    >  
    <input  
        type="submit"  
        value="Valider"  
    >  
</form>
```

- Dans la balise *form*, l'attribut *action* permet de spécifier le fichier PHP à qui les données seront envoyés pour le traitement
- Il convient de spécifier la *méthode HTTP* (par défaut *GET* ou *POST*)
- Les données seront récupérées et traitées dans le fichier grâce aux superglobales *\$_GET* et *\$_POST*
- Les attributs *name* des champs *input*, *textarea* permettront de récupérer les bonnes informations côté PHP

Traitement des données avec \$_GET

- Méthode par défaut pour transmettre des informations entre un client et un serveur
- La superglobale `$_GET` est un tableau associatif qui contient toutes les informations transmises sous forme de clé et valeur
 - La clé est le nom du paramètre
 - La valeur est l'information qui vient après le =

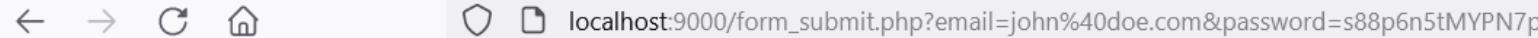
```
<form method="GET" action="form_submit.php">
  <input type="email" name="email" placeholder="email">
  <input type="password" name="password" placeholder="mot de passe">
  <input type="submit" value="Valider">
</form>
```

john@doe.com

.....

Valider

```
cours > demo > form_submit.php
1  <?php
2
3  var_dump($_GET);
```



```
F:\formations\
array (size=2)
  'email' => string 'john@doe.com' (Length=12)
  'password' => string 's88p6n5tMYPN7p' (Length=14)
```

Traitement des données avec \$_POST

- De même que \$_GET, la superglobale \$_POST est un tableau associatif contenant les informations transmises avec la méthode POST
- Adapté pour
 - Données volumineuses
 - Envoi des fichiers
 - Données confidentielles (qui ne seront pas visibles directement sur l'URL, cependant l'information peut facilement être vu par un utilisateur expérimenté)

```
<form method="POST" action="form_submit.php">
  <input type="email" name="email" placeholder="email">
  <input type="password" name="password" placeholder="mot de passe">
  <input type="submit" value="Valider">
</form>
```

The screenshot shows a web browser window with a form. The form has three input fields: an email field containing 'john@doe.com', a password field containing a redacted password (represented by a series of dots), and a submit button labeled 'Valider'. The browser's address bar displays the URL 'localhost:9000/form_submit.php'.

```
F:\formations\array (size=2)
  'email' => string 'john@doe.com' (length=12)
  'password' => string 's88p6n5tMYPN7p' (length=14)
```

Adresse email john@doe.com

```
<?php
var_dump($_POST);
echo 'Adresse email '.$_POST['email'];
```

Comparaison GET & POST

GET

- Transmission simple des informations via *l'URL*
- Les informations sont directement visibles depuis l'URL
- Directement *modifiables* toujours depuis l'URL
- Pratique pour transmettre des données dans un script
- Elles peuvent être **mises en cache**, enregistrées dans les favoris et/ou dans l'historique
- Utilisables en dehors des formulaires, c'est notamment pratique pour des tâches planifiées depuis un serveur
- Possible de revenir en arrière dans l'historique de navigation
- Les données sont **limitées en longueur**

POST

- Paramètres **non-visibles par un utilisateur novice**, elles restent lisibles dans les outils de [DevTools](#) notamment dans la partie réseau
- Les informations en POST ne sont pas mises en cache, favoris ou historique
- Reste modifiable par un utilisateur qui s'y connaît
- **Pas de restriction sur la longueur des données**

Traitement des fichiers envoyés avec `$_FILES`

- Tableau associatif à 2 dimensions
- Permet de récupérer dans un dossier temporaire les fichiers *uploadés* (chargés par l'utilisateur)
- Attention, côté HTML, il faut rajouter l'attribut `enctype="multipart/form-data"`

```
<form method="POST" action="form_submit.php" enctype="multipart/form-data">
  <input type="email" name="email" placeholder="email">
  <input type="password" name="password" placeholder="mot de passe">
  <label for="face">Photo de passeport</label>
  <input type="file" name="face" id="face">
  <input type="submit" value="Valider">
</form>
```

```
array (size=2)
  'email' => string 'john@doe.com' (Length=12)
  'password' => string '' (length=0)
```

```
array (size=1)
  'face' =>
    array (size=6)
      'name' => string '5f8c1d91989bd.jpeg' (Length=18)
      'full_path' => string '5f8c1d91989bd.jpeg' (Length=18)
      'type' => string 'image/jpeg' (Length=10)
      'tmp_name' => string 'D:\wamp\tmp\phpFFEE.tmp' (Length=23)
      'error' => int 0
      'size' => int 74399
```

cours > demo > `php form_submit.php`

```
1   <?php
2
3   var_dump($_POST, $_FILES);
```

john@doe.com

mot de passe

Photo de passeport 5f8c1d91989bd.jpeg

Fonctions dédiées aux formulaires

- Fonctions dédiées aux échanges entre le client et le serveur via un formulaire
- [addslashes\(\)](#) : retourne une chaîne après avoir échappé tous les caractères qui doivent l'être
- [htmlentities\(\)](#) et [htmlspecialchars\(\)](#) : convertit tous les caractères éligibles en entités HTML
- [getallheaders\(\)](#) : récupère tous les en-têtes de la requête HTTP
- [urlencode\(\)](#) et [urldecode\(\)](#) : encode/décode une chaîne encodée URL
- [base64_encode\(\)](#) et [base64_decode\(\)](#) : encode/décode une chaîne en MIME base64
- [filter_input\(\)](#) ; filtrer une variable externe (ex: vérifier qu'un email envoyé par le client est au bon format d'une adresse email conventionnelle)
- [json_encode\(\)](#) : retourne la représentation JSON d'une valeur
- [json_decode\(\)](#) : décode une chaîne JSON

Démo traitement d'un formulaire

Inscription

ValiderConnexion

V. Gestion des fichiers

Inclusion des fichiers

- L'inclusion des fichiers permet de mutualiser du code dans un fichier que l'on peut inclure une ou plusieurs fois
 - Par exemple, pour les parties statiques telles que l'en-tête, menu, footer, etc.
- [Require\(\)](#) : inclus un fichier et rends toutes les variables PHP du fichier source dans le fichier de destination. Déclenche une erreur fatale lorsque le fichier n'est pas trouvé.
- [require_once\(\)](#) : similaire à *require* sauf qu'il n'inclut pas le fichier s'il a déjà été inclus précédemment.
- [include\(\)](#) : idem que *require* sauf qu'il n'y a pas d'erreur **fatale** lorsque le fichier n'est pas trouvé, il y a un *warning* qui ne bloque pas la suite de l'exécution de votre code.
- [include_once\(\)](#) : si le fichier est déjà inclus, il ne va pas l'inclure une deuxième fois mais retourner *true*

Exemple avec require

```
head.php
cours > demo > includes > head.php > div > h2
1  <div>
2    <h2>En tête</h2>
3    <p>LOGO</p>
4  </div>

menu.php
cours > demo > includes > menu.php > nav > h2
1  <nav>
2    <h2>Menu</h2>
3    <ul>
4      <li>Accueil</li>
5      <li>Contact</li>
6      <li>Inscription</li>
7    </ul>
8  </nav>

footer.php
cours > demo > includes > footer.php > div > h2
1  <div>
2    <h2>Pied de page</h2>
3    <nav>
4      <ul>
5        <li>Mentions légales</li>
6        <li>RGPD</li>
7        <li>Notre entreprise</li>
8      </ul>
9    </nav>
10   </div>
```

```
index.php
cours > demo > includes > index.php > html > body > header
2  <html lang="fr">
3    <head>
4      <body>
5        <header>
6          <?php include 'head.php'; ?>
7          <?php require '404.php'; ?>
8        </header>
9        <nav><?php include_once 'menu.php'; ?></nav>
10       <main><h2>Contenu principale</h2></main>
11       <footer>
12         <?php include 'menu.php'; ?>
13         <?php include_once 'footer.php'; ?>
14       </footer>
15     </body>
16   </html>
```

En tête

LOGO

(!)	Warning: require(404.php): Failed to open stream: No such file or directory in F:	formations	on line 11
Call Stack			
#	Time	Memory	Function
1	0.0021	357936	{main}()

(!)	Fatal error: Uncaught Error: Failed opening required '404.php' (include_path='.;C:\php\pear') in F:	formations	on line 11
Call Stack			

(!)	Error: Failed opening required '404.php' (include_path='.;C:\php\pear') in F:	formations	on line 11
Call Stack			
#	Time	Memory	Function
1	0.0021	357936	{main}()

Exemple avec include

```
head.php X ...
cours > demo > includes > head.php > div > h2
1  <div>
2    <h2>En tête</h2>
3    <p>LOGO</p>
4  </div>

index.php X ...
cours > demo > includes > index.php > html > body
2   <html lang="fr">
3     <body>
4       <header>
5         <?php include 'head.php'; ?>
6         <?php include '404.php'; ?>
7       </header>
8       <nav><?php include_once 'menu.php'; ?></nav>
9       <main><h2>Contenu principale</h2></main>
10      <footer>
11        <?php include 'menu.php'; ?>
12        <?php include_once 'footer.php'; ?>
13      </footer>
14    </body>
15  </html>

menu.php X ...
cours > demo > includes > menu.php > nav > h2
1  <nav>
2    <h2>Menu</h2>
3    <ul>
4      <li>Accueil</li>
5      <li>Contact</li>
6      <li>Inscription</li>
7    </ul>
8  </nav>

footer.php X ...
cours > demo > includes > footer.php > div > h2
1  <div>
2    <h2>Pied de page</h2>
3    <nav>
4      <ul>
5        <li>Mentions légales</li>
6        <li>RGPD</li>
7        <li>Notre entreprise</li>
8      </ul>
9    </nav>
10   </div>
```

En tête

LOGO

(!) Warning: include(404.php): Failed to open stream: No such file or directory in F:\formation\index.php on line 11				
Call Stack	#	Time	Memory	Function
	1	0.0016	357936	[main]()

(!) Warning: include(): Failed opening '404.php' for inclusion (include_path='.;C:\php\pear') in F:\formation\index.php on line 11				
Call Stack	#	Time	Memory	Function
	1	0.0016	357936	[main]()

Menu

- Accueil
- Contact
- Inscription

Contenu principale

Menu

- Accueil
- Contact
- Inscription

Pied de page

- Mentions légales
- RGPD
- Notre entreprise

Fonctions associées aux commandes shell

- [basename\(\)](#) : donne le nom d'un fichier
- [dirname\(\)](#) : donne le chemin d'un fichier
- [link\(\)](#) : crée un lien physique
- [symlink\(\)](#) : crée un lien symbolique
- [unlink\(\)](#) : détruit un lien
- [touch\(\)](#) : change la date de modification
- [chmod\(\)](#) : modifie les droits d'accès
- [chown\(\)](#) : modifie le propriétaire
- [chgrp\(\)](#) : modifie le groupe
- [mkdir\(\)](#) : crée un répertoire
- [rmdir\(\)](#) : détruit un répertoire
- [copy\(\)](#) : copie un fichier

Fonctions dédiées aux fichiers

- [fopen\(\)](#) : ouvre un fichier ou un URL
- [fclose\(\)](#) : ferme un fichier
- [fread\(\)](#) : lecture de fichier en mode binaire
- [fwrite\(\)](#) : écrit un fichier en mode binaire
- [file_get_contents\(\)](#) : retourne l'intégralité d'un fichier dans une chaîne de caractères
- [file_put_contents\(\)](#) : écrit des données dans un fichier
- [file_exists\(\)](#) : vérifie si une ressource (fichier ou un dossier) existe
- [unlink\(\)](#) : supprime un fichier
- [rename\(\)](#) : renomme une ressource
- [is_file\(\)](#) : vérifie que la ressource est un fichier
- [is_writable\(\)](#) : vérifie que le fichier est accessible en écriture

[Documentation de toutes les fonctions dédiées aux fichiers](#)

Démo manipulation des fichiers

VI. Gestion des bases de données SQL

Rappels base de données

- Une **base de données** (ou BDD) est un ensemble structuré de données enregistrées dans des tables
- Les manipulations fréquentes sur les données sont connus sous l'acronyme CRUD pour
 - CREATE : insertion des données dans une table
 - READ : lecture des données
 - UPDATE : mise à jour des données
 - DELETE : suppression des données
- Il y a plusieurs types de base de données, ici nous utiliserons les bases de données SQL manipulable à travers le langage SQL
- Contrairement au stockage des informations dans un fichier, les BDD sont plus en puissant, plus facile à manipuler et scalable (étendre la structure et le volume de stockage), etc.
- Il existe plusieurs SGBDR (moteurs ou « logiciels ») de base de données :
 - Oracle, Microsoft SQL Server, IBM DB2, Sybase, etc. destinés à gérer d'enormes volumes de données et pas gratuits
 - MySQL/MariaDB un peu moins performants que ceux cités ci-dessus mais gratuits - en particulier MySQL que l'on abordera dans ce cours

Démo PHPMYADMIN



Classe PDO

PDO (PHP Data Objects)

Connexion à une BDD avec PHP

- La classe PDO est dédiée à faire l'interface entre le programme PHP et une base de données
- Permet l'accès et la manipulation d'une base de données de manière très simple et facile
- PDO s'interface avec tout type de base de données (Oracle, MySQL, SQL Server, SQLite, etc.)

Documentations

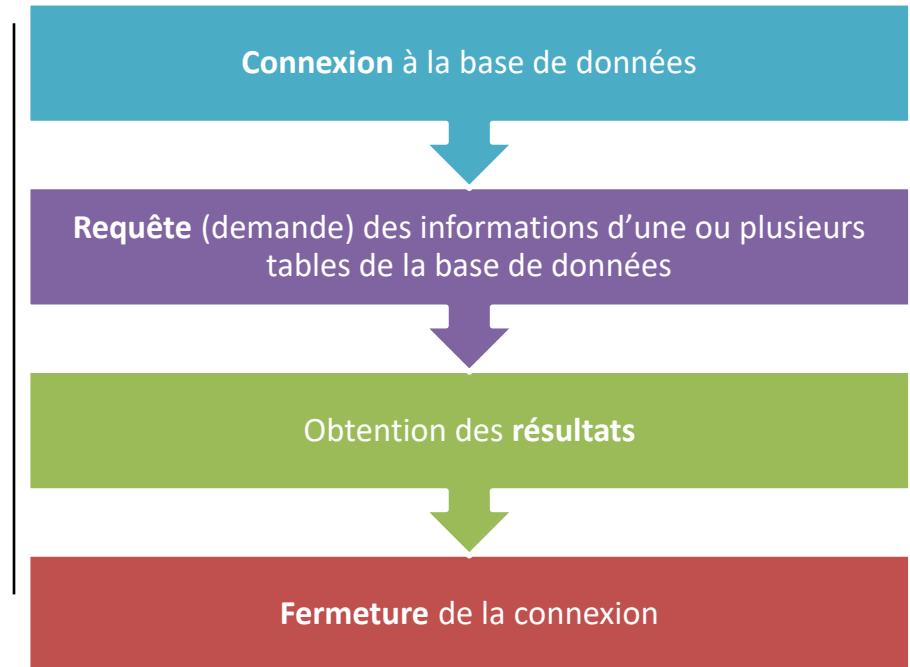
- [Connexion à la base de données](#)
- [Configuration de PDO](#)
- [Requêtes préparées](#)
- [Gestion des exceptions](#)
- [Toute la documentation](#)

Utilisation de la base de données

Connexion à la base de données (BDD)

- Nom ou **adresse IP** de l'hôte : adresse du serveur BDD.
En configuration locale, l'hôte se nomme **localhost** ou avec l'IP **127.0.0.1**
- **Port** : port de communication pour se connecter au serveur de BDD. Par défaut, le port est 3306 pour MySQL
- **Base de données** : nom de la BDD à laquelle on souhaite se connecter
- **Login ou nom de l'utilisateur** : utilisateur souhaitant et ayant les droits de se connecter à la BDD
- **Mot de passe de l'utilisateur** : mot de passe de l'utilisateur

Les étapes



Quelques méthodes de la classe PDO

- [__construct\(\)](#) : crée une instance PDO qui représente une connexion à la BDD
- [exec\(\)](#) : exécute une requête SQL et retourne le nombre de lignes affectées
- [prepare\(\)](#) : prépare une requête à l'exécution et retourne un objet
- [query\(\)](#) : exécute une requête SQL et retourne une instance [PDOStatement](#) manipulable pour récupérer les données
- [fetch\(\)](#) : à partir d'une instance de PDOStatement, lit séquentiellement un résultat des données retournées
- [fetchAll\(\)](#) : idem que fetch, à la différence qu'on peut lire tous les résultats sous forme d'un tableau et met fin à la connexion à la BDD
- [setAttribute\(\)/getAttribute\(\)](#) : configue/récupère un attribut de PDO
- [lastInsertId\(\)](#) : retourne l'identifiant de la dernière ligne insérée
- [quote\(\)](#) : protège une chaîne pour l'utiliser dans une requête SQL PDO
- [beginTransaction\(\)](#) : démarre une transaction
- [commit\(\)/rollBack\(\)](#) : valide/annule une transaction
- [getAvailableDrivers\(\)](#) : retourne la liste des pilotes PDO disponibles
- [errorCode\(\)](#) : retourne le SQLSTATE associé avec la dernière opération sur la BDD
- [errorInfo\(\)](#) : retourne les informations associées à l'erreur sur la dernière opération

Principaux formats des réponses des données via PDO

- PDO_FETCH_BOTH
 - Format par défaut
 - Résultats sous-forme d'un composé des résultats à la fois avec des indices numériques et en chaîne de caractères
- PDO_FETCH_ASSOC
 - Résultats sous-forme d'un tableau associatif
- PDO_FETCH_OBJ
 - Tableau associatif dont la valeur est un objet de type stdClass
- [Documentation sur toutes les modes de réponse](#)

Démo Mise en pratique PDO

Requêtes préparées

Pourquoi ?

- Ne jamais faire confiance aux données attendues par les utilisateurs et récoltées à partir d'un formulaire
- Les différentes failles de sécurité pouvant permettre à un utilisateur d'accéder à des informations qu'il n'a pas le droit ou d'effectuer des modifications dans la base de données sans l'accord de l'utilisateur
 - Injection SQL : profiter du langage SQL pour modifier la requête exécutée par votre programme
 - Faille CSRF (non abordée dans ce cours)
 - Etc.

Solution

- La requête préparée permet de contrer la faille d'injection SQL
- Les requêtes préparées sont
 - Plus performants, car stocké en mémoire
 - Plus sûre, car à chaque valeur reçue par le client, elle sera associée à la valeur de l'attribut attendu au niveau de la requête SQL

Zoom injection SQL

[Source image gnut.wordpress.com](http://gnuit.wordpress.com)

SQL Injection.

User-Id :

Password :

```
select * from Users where user_id= 'srinivas' and password = 'mypassword'
```

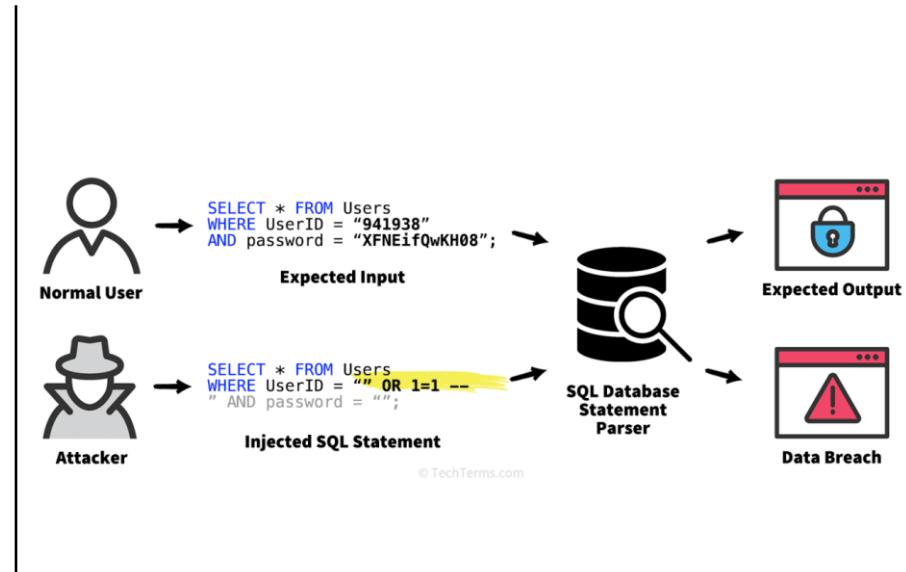
User-Id :

Password : /*--

```
select * from Users where user_id= '' OR 1= 1; /*' and password = '*/*'
```

9lessons.blogspot.com

[Source image E. KAAS Harrys-phills](http://E.KAAS.Harrys-phills)



Démo Mise en pratique requêtes préparées



VII. Gestion des sessions et cookies

Superglobale `$_SESSION`

- Le protocole HTTP est *stateless*, c'est-à-dire que les informations ne sont pas conservées d'une requête à une autre
- Une *session* permet de conserver les informations d'un utilisateur durant sa navigation entre les pages (de requête en requête)
- Les informations de session sont stockées côté serveur

Quelques fonctions dédiées aux sessions

- [`session_start\(\)`](#) : démarrer une session
- La superglobale `$_SESSION` permet d'accéder à la valeur stockée en session à partir d'une clé
- [`session_id\(\)`](#) : définir un identifiant unique de session
- [`session_unset\(\)`](#) : détruire toutes les variables d'une session
- [`session_destroy\(\)`](#) : détruire la session en cours

Superglobale `$_COOKIE`

- Petits fichiers stockés par le navigateur sur le disque dur d'un visiteur à la demande du serveur
- Fichier d'une taille n'excéder pas 4 ko en moyenne
- Les informations sont stockées côté client
- La superglobale `$_COOKIE` permet de récupérer une valeur à partir d'une clé
- La création d'un cookie s'effectue à l'aide de la méthode `setcookie()`
- [Documentation de la fonction setcookie](#)

Paramètres de la fonction <code>setcookie()</code>	Définition
Name	Nom du cookie
Value	Valeur
Expires	Date d'expiration en timestamp UNIX
Path	Disponibilité du cookie dans un répertoire et/ou sous-répertoire spécifique du serveur
Domain	Disponibilité du cookie sur un domaine ou sous-domaine spécifique
Secure	Connexion sécurisée ou non
Httponly	Utilisation du protocole HTTP ou HTTPS (interdire l'accès via des scripts en JavaScript)

Session VS cookie

	Session	Cookie
Stockage	Côté serveur	Côté client
Limite des données	Illimitée	< 4ko
Sécurité	Plus sécurisée par le stockage sur le serveur	Facile d'effectuer une modification côté client
Certains usages	<ul style="list-style-type: none">Garder des informations sensibles d'un utilisateur (les sessions ne sont pas infaillibles, par ex. faille fixation de session)Panier d'achatCache	<ul style="list-style-type: none">Garder des informations non-sensible d'un utilisateurGarder des données d'un formulaire (pré-remplissage)StatistiquesMarketing
Expiration	Fermeture du navigateur	Date d'expiration
Valeur des stockage	Tout type (string, objet, tableaux, etc.)	Uniquement des strings (chaîne de caractères). Les objets, tableaux, etc. peuvent être serialisés (transformé en string)

Démo Mise en pratique session et cookies

VIII. Nouveautés PHP8

Arguments nommés

Source image sensiolabs

```
<?php

function foo(string $p1 = '', string $p2 = '', string $p3 = '', string $p4 = '')
{
    echo $p1 . ' ' . $p2 . ' ' . $p3 . ' ' . $p4;
}

// avant PHP 8
foo(' ', ' ', ' ', 'bar');

// après PHP 8
foo(p4: 'bar');
```

- Donner plus de sens aux arguments
- Permet d'avoir un code plus propre et surtout plus lisible
- Avec les attributs nommés
 - On peut ne pas tenir compte de l'ordre de passage des arguments établi dans la définition de la fonction
 - Lors de l'appel, les arguments à passer doivent être nommés comme ils ont été définis dans la signature
- Facilite l'appel des méthodes qui possèdent plusieurs arguments en nommant uniquement les arguments auxquels on souhaite affecter une valeur

Attributs

Source image sensiolabs

```
// PHP 8.0
namespace App\Controller;

use Symfony\Component\Routing\Annotation\Route;

class HelloWorldController
{
    #[Route('/foo', methods: ["GET"])]
    public function __invoke()
    {
        echo "Hello world !";
    }
}
```

Exemple

- A l'aide des commentaires sur une méthode ou une classe, celle-ci accède à des nouvelles informations ou fonctionnalités
- Similaire aux annotations, en référence aux décorateurs ([design pattern \(patron de conception\)](#)) qui permettent de décorer un objet, c'est-à-dire de lui attribuer des comportements (fonctionnalités) supplémentaires

Propriétés contrôleur ([source image sensiolabs](#))

- Allégement de la notation pour déclarer les attributs d'une classe directement au niveau du contrôleur

```
<?php

class Address
{
    private string $street;
    private string $city;
    private string $country;

    public function __construct(string $street, string $city, string $country)
    {
        $this->street = $street;
        $this->city = $city;
        $this->country = $country;
    }
}
```

```
<?php

class Address
{
    public function __construct(
        private string $street,
        private string $city,
        private string $country,
    )
    {}

}
```

Type union

Source image sensiolabs

```
<?php

declare(strict_types=1);

function square(int|float $number): int|float
{
    return $number * $number;
}
```

- Permet d'accepter plusieurs types de valeur pour un paramètre donnée
- Pour cela, on utilise | (barre vertical obtenu sur Windows avec ALT + 6, en anglais pipe)

Instruction match

Source image sensiolabs

```
<?php

$foo = match(0) {
    0.0 => 'float has matched',
    0 => 'integer has matched',
    '0' => 'string has matched',
};

// print : integer has matched
echo $foo;

$foo = match(0.0) {
    0.0 => 'float has matched',
    0 => 'integer has matched',
    '0' => 'string has matched',
};

// print : float has matched
echo $foo;

$foo = match('0') {
    0.0 => 'float has matched',
    0 => 'integer has matched',
    '0' => 'string has matched',
};

// print : string has matched
echo $foo;
```

- Alternative au switch permettant d'avoir un code moins verbeux et plus lisible
- Comparaison plus stricte que le switch, c'est-à-dire comparaison avec la valeur et le type
- Le résultat d'un match peut être stocké dans une variable
- Syntaxe similaire au switch et à un tableau associatif avec
 - La partie clé qui correspond à la condition
 - La partie valeur, la valeur qui sera retournée si la condition est vraie
- Avec un switch, le cas par défaut est optionnel, avec un match, il faut spécifier toutes les conditions, dans le cas contraire, il y aura une erreur

Opérateur NULL-SAFE ([source image sensiolabs](#))

```
<?php

class Foo {
    private ?Bar $bar = null;

    public function getBar(): ?Bar
    {
        return $this->bar;
    }
}

class Bar {
    private string $baz = "someString";

    public function getBaz(): string
    {
        return $this->baz;
    }
}

$foo = new Foo;

// PHP 7.4
if (null !== $foo->getBar()) {
    echo $foo->getBar()->getBaz();
}

// PHP 8.0
echo $foo->getBar()?->getBaz();
```

- Chainer des méthodes sans vérifier que la méthode précédente renvoie une valeur non nulle

Nouvelles fonctions dédiées à la recherche d'un motif

- [str_contains\(\)](#) : vérifie qu'un motif ou sous-chaîne existe dans une autre chaîne de caractères
- [str_starts_with\(\)](#) : vérifie qu'une chaîne commence par un motif
- [str_ends_with\(\)](#) : vérifie qu'une chaîne se termine par un motif
- Ces nouvelles fonctionnent permettent de s'abstraire des expressions régulières avec l'utilisation de la fonction [preg_match\(\)](#) pour les cas de figure précédentes
- [json_validate\(\)](#) : vérifie qu'une chaîne contient un JSON valide, utilise moins de mémoire que la fonction [json_decode\(\)](#)

ANNEXE

Documentation PHP

- [Documentation PHP](#)
- [Documentation toutes les nouveautés PHP8](#)
- [Documentation nouveauté PHP8.1](#)
- [Documentation nouveauté PHP8.2](#)
- [Documentation nouveauté PHP8.3](#)
- [Documentation nouveauté PHP8.4](#)
- [Documentation nouveauté PHP8.5](#)

ALLER PLUS LOIN

- [PHP Standards Recommendations \(PSR\)](#)
- [Utilisation des Namespace](#)
- [Gestionnaire de dépendance de PHP avec Composer](#)
- [Architecture MVC](#)
- [ORM Doctrine](#)
- [Framework Symfony](#)
- [PHPUnit](#)

Merci de votre attention et participation
Glodie Tshimini
