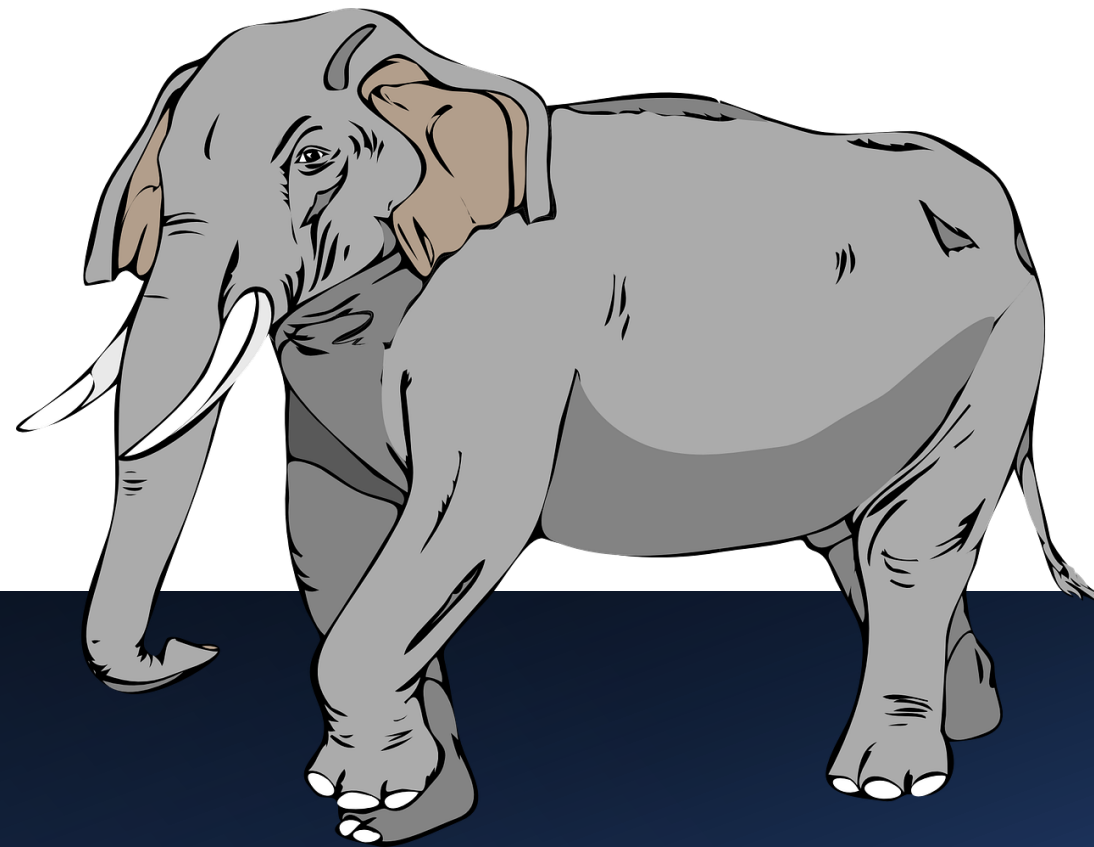


# PHP

**Jean-Claude AZIAHA**



# SOMMAIRE

## **Partie A : L'environnement PHP**

- 1- Qu'est-ce que PHP ?
- 2- Quelle est son utilité ?
- 3- Quel est son principe de fonctionnement ?
- 4- Quels seront nos outils de travail ?
- 5- Comment procéder à son installation ?

# SOMMAIRE

## **Partie B : Les bases du langage PHP**

- 1- L'emplacement du code PHP
- 2- Les variables et les différents types
- 3- Les variables de type scalaire
- 4- Les instructions d'affichage
- 5- Les commentaires
- 6- Les variables de type composé

# SOMMAIRE

## **Partie B : Les bases du langage PHP**

- 7- Les opérateurs et les différentes opérations possibles en PHP
- 8- Les conditions
- 9- Les boucles
- 10- Les fonctions natives (proposées par PHP)
- 11- Création de fonctions pouvant retourner une valeur
- 12- Création de fonctions retournant rien comme valeur
- 13- Les fonctions et les types
- 14- La portée de variables

# SOMMAIRE

## Partie C : PHP et HTML

- 1 - Inclusion du contenu d'un fichier dans un autre fichier
- 2 - La variable super globale \$\_GET
- 3 - La variable super globale \$\_POST
- 4 - La variable super globale \$\_FILES
- 5 - La variable super globale \$\_COOKIE
- 6 - La variable super globale \$\_SERVER
- 7 - La variable super globale \$\_SESSION

# SOMMAIRE

## **Partie D : Travaux dirigés**

Afin de nous assurer que le cours soit bien compris, nous allons réaliser :

- Une série de questions-réponses
- Des travaux dirigés

# L'ENVIRONNEMENT PHP

# 1- Qu'est-ce que le PHP ?

**PHP** est un acronyme qui signifie en réalité HyperText PreProcessor.

C'est un **langage** de programmation créé par **Rasmus Lerdorf**.

Il est apparu en **1994** et conçu sur la base du langage C.

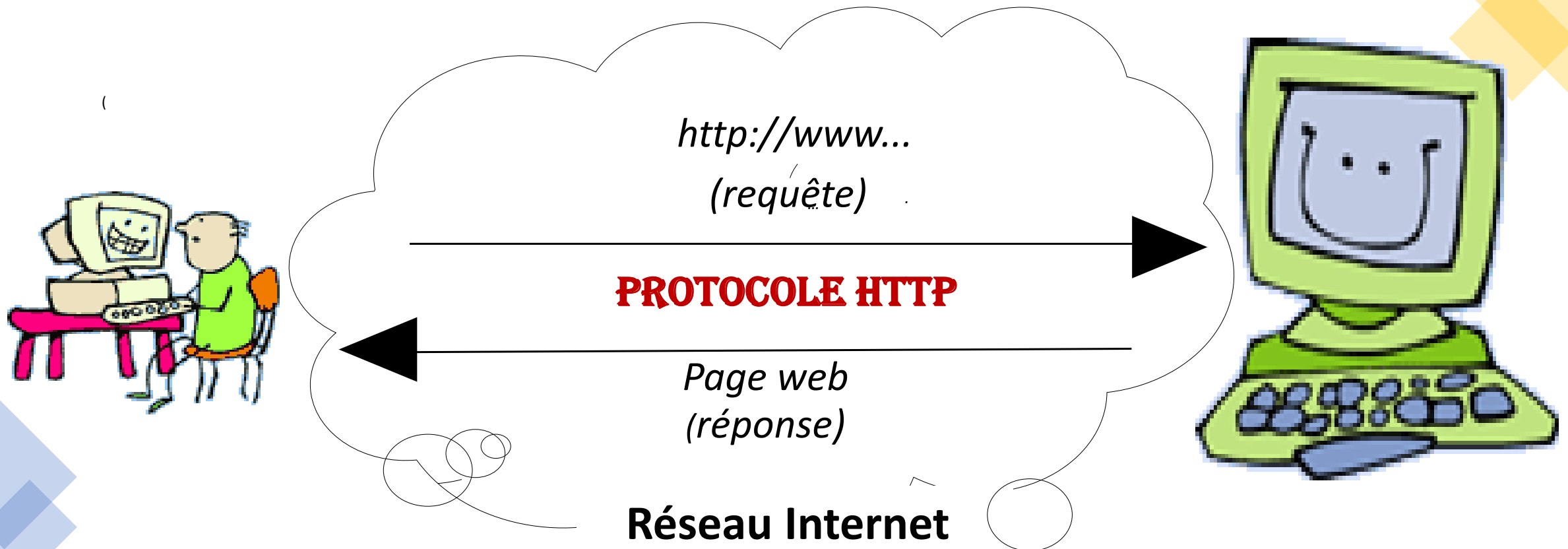


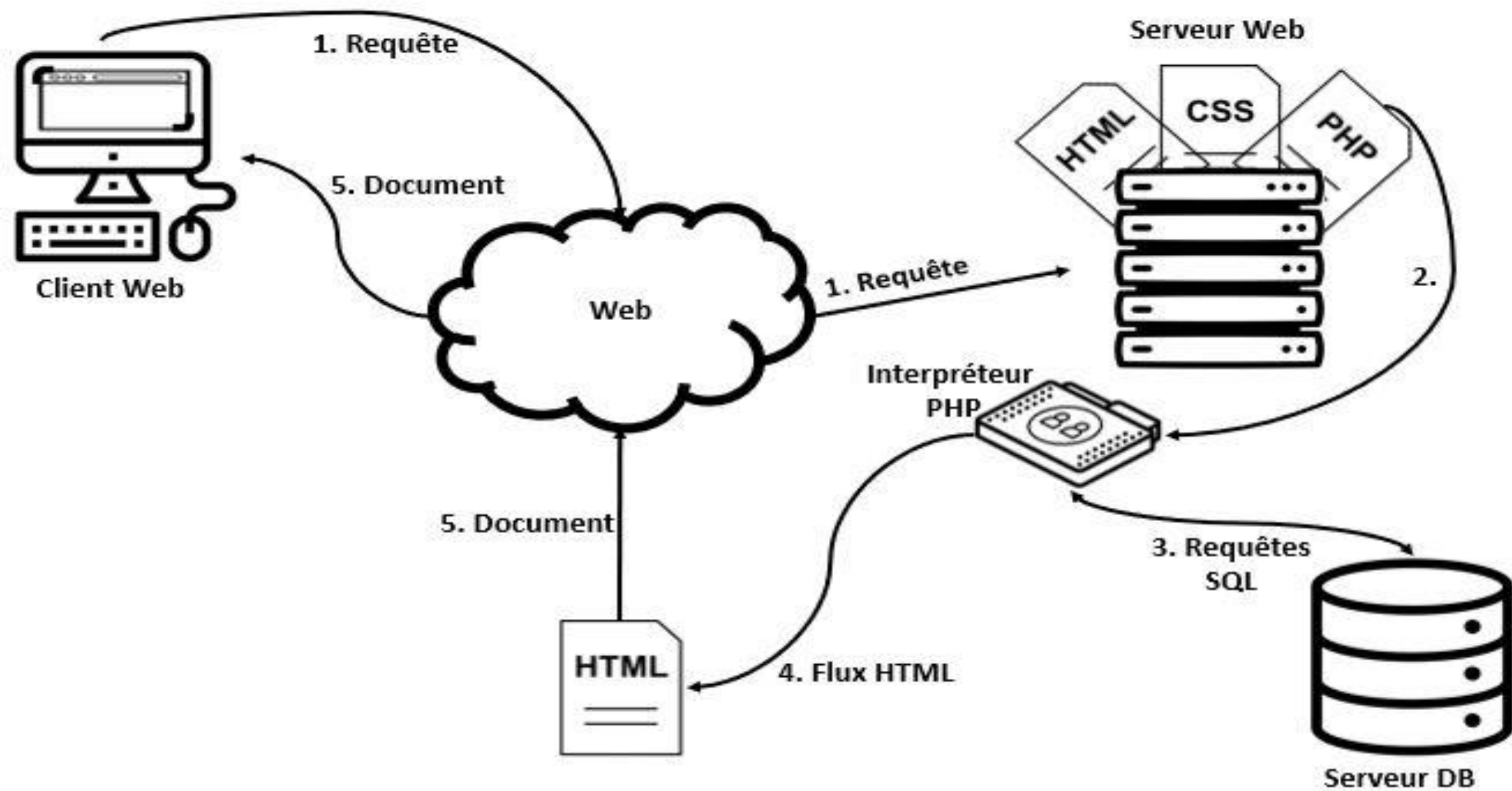
## 2- Quelle est son utilité ?

**PHP** permet de **créer des sites web dynamiques**.

- **Un site web statique:** Son contenu est généralement fixe et ne change pas fréquemment. Les pages sont généralement codées en dur en HTML et présentent les mêmes informations à tous les utilisateurs. Les modifications nécessitent généralement une intervention manuelle dans le code source.
- **Un site web dynamique:** Son contenu est généré en fonction des actions de l'utilisateur ou de données stockées dans une base de données. Le contenu peut donc varier en fonction de l'utilisateur, de la date, localisation...

### 3- Quel est son principe de fonctionnement ?





### 3- Quel est son principe de fonctionnement ?

**PHP** est donc un langage qui s'exécute du côté serveur ou back-end.

Grâce au protocole de transport HTTP, il :

- ❖ Reçoit la requête du client
- ❖ Réalise le traitement
- ❖ Lui retourne la réponse correspondante

## 4- Quels seront nos outils de travail

Pour coder en PHP, nous avons besoin :

- **Apache** pour simuler un serveur local
- Du **code source de PHP** afin de s'en servir pour coder en php
- **MySQL**, système de gestion de base de données relationnelle
- Editeur de code (**Visual Studio Code...**)
- Navigateur : (**Chrome...**)

## 5- Comment procéder à son installation ?

Pour installer Apache, PHP et MySQL nous avons besoin de l'un de ces logiciels à choisir en fonction du système d'exploitation

- **Wamp** pour Windows: <https://www.wampserver.com/>
- **Xampp** pour Windows: <https://www.apachefriends.org/fr/index.html>
- **Mamp server** pour Mac: <https://www.mamp.info/en/downloads/>
- **Lamp server** pour Linux : <https://doc.ubuntu-fr.org/lamp>

## 5- Comment procéder à son installation ?

Pour installer Visual Studio Code : <https://code.visualstudio.com/>

Pour installer Google Chrome : <https://www.google.com/chrome/>

# LES BASES DU LANGUAGE PHP



# 1- L'emplacement du code PHP

Pour Wamp server, se placer dans le dossier **C:\wamp64\www**

Pour Xampp server, se placer dans le dossier **C:\xampp\htdocs**

Pour Mamp server, se placer dans le dossier **htdocs**

Pour un début, il est indispensable que tout site créé en local en php, soit à cet emplacement.

# 1- L'emplacement du code PHP

- **Dans** le dossier **www** ou **htdocs**, créons un **nouveau dossier** du nom **"learn\_php"**.
- **Ouvrir** ce dossier vide **dans** notre **éditeur** de code.
- Dans ce dossier, créons notre premier fichier de nom **"index.php"**.
- **Dans l'index.php**, commençons par mettre : **<?php ?>**

# 1- L'emplacement du code PHP

**NB**

Il est indispensable que tout code php soit dans un fichier dont l'extension est **".php"**.

Dans le fichier, pour que du code php soit interprété, il est indispensable qu'il soit entouré de **<?php ?>**

**PRATIQUONS !**

## 2- Les variables et les différents types

Une variable est une zone mémoire identifiée par un nom qui contient une valeur lisible et modifiable dans le programme.

Les **variables** ne servent à **stocker** une information que **temporairement**. C'est-à-dire durant l'exécution du script.

Le type d'une variable en PHP dépend du type de la valeur lui est affectée.

## 2- Les variables et les différents types

### Les règles de nommage

Déclarer une variable revient à lui donner un nom.

Ce nom doit toujours commencer par un \$, suivi d'un mélange ou non de lettres et de chiffres.

Exemple : \$code , \$nombre1

Le nom des variables est sensible à la casse : **\$code** et **\$Code** sont vues par **PHP** comme deux variables différentes.

## 2- Les variables et les différents types

Le nom d'une variable doit **commencer par** une lettre **majuscule** ou **minuscule** ou un **\_**  
**mais pas un chiffre.**

La suite du nom d'une variable peut comporter des lettres, des chiffres et le caractère **\_**

(les espaces, le trait d'union ainsi que les caractères spéciaux comme @, &, ^, \$... ne sont pas autorisés).

## 2- Les variables et les différents types

Une valeur peut être affectée à une variable grâce à l'opérateur d'affectation =

Exemple : **\$code = 1 ;**

Nous venons donc **d'initialiser** la variable **\$code**;

Autrement dit, nous avons **déclaré** **\$code** puis nous lui avons **affectée** la valeur **1** qui est un entier positif.



## 2- Les variables et les différents types

PHP propose **quatre types scalaires** (c'est-à-dire des données ne pouvant contenir qu'une seule valeur à la fois), **deux types composés** (pouvant contenir plusieurs valeurs à la fois) et **deux types spéciaux** :

### A-Types scalaires :

- nombre entier
- nombre à virgule
- chaîne de caractères
- booléen

### B-Types composés :

- tableau
- objet

### C-Types spéciaux :

- null
- resource

## 3- Les variables de type scalaire

### Le type entier (integer)

Le type entier permet de stocker en mémoire, un chiffre ou un nombre entier. Il peut être positif ou négatif.

#### Exemple :

```
$chiffre      = 3;
```

```
$code_postal  = 77000;
```

```
$temperature  = -1;
```

### 3- Les variables de type scalaire

#### Le type flottant (float ou double)

Le type flottant permet de stocker en mémoire, un chiffre ou un nombre décimal. Il peut être positif ou négatif.

#### Exemple :

```
$chiffre = 3.3;
```

```
$resultat = -10.5;
```

## 3- Les variables de type scalaire

### Le type chaîne de caractères (string)

Une chaîne de caractères comme son nom l'indique est une séquence de plusieurs caractères. Ses caractères peuvent être des lettres, des chiffres, des caractères spéciaux.

Pour que PHP reconnaisse une chaîne de caractères, il faut que cette dernière soit **entourée de doubles griffes** ou **simples griffes**

(Penser à échapper l'apostrophe avec l'antislash \ ).

## 3- Les variables de type scalaire

### Le type chaîne de caractères (string)

Exemple :

```
$plat = "Lasagnes";
```

```
$phrase0 = "Hello! C'est Julie.";
```

```
$phrase1 = 'Hello! C\'est Julie.';
```

## 4- Les instructions d'affichage

```
$prenom = "JC";
```

```
echo $prenom;
```

Cette instruction affiche uniquement le contenu d'une variable de type scalaire.

```
var_dump($prenom);      print_r($prenom);
```

Ces instructions permettent d'afficher le contenu d'une variable de type scalaire, composé et spécial.

### 3- Les variables de type scalaire

**Le type boolean:** Ce type de variable ne peut contenir que **true** ou **false** comme valeur.

Exemple :

```
$test_covid = false;
```

```
$testGrippe = true;
```

## 5- Les commentaires

Un commentaire PHP à 3 principales fonctions :

- **Documentation** : Les commentaires peuvent servir à documenter votre code, en expliquant son fonctionnement, ses objectifs, ses entrées et ses sorties, ainsi que d'autres informations importantes. Une documentation bien écrite facilite la compréhension du code par d'autres développeurs et par vous-même à l'avenir.
- **Débogage** : Les commentaires peuvent être utiles lors du débogage en vous permettant de désactiver temporairement certaines parties du code sans les supprimer réellement. Cela peut être utile pour isoler un problème ou tester différentes sections du code.



## 5- Les commentaires

- **Communication** : Les commentaires sont un moyen de communication entre les membres d'une équipe de développement. Ils permettent aux développeurs de communiquer des informations importantes sur le code, les modifications apportées, ou les décisions de conception.

## 5- Les commentaires

Voici la syntaxe :

// Commentaire monoligne

/\*

\* Commentaire

\* multiligne

\*/

## 6- Les variables de type composé

### Les tableaux (array)

Les **tableaux** ou array en anglais sont des variables avec la **particularité** de pouvoir contenir **plusieurs valeurs** à la fois.

Une variable de type tableau peut être définie explicitement grâce à la fonction **array()** ou implicitement en utilisant une notation à crochets **[]**.

## 6- Les variables de type composé

### Les tableaux (array)

En PHP, un tableau est donc une liste d'éléments, ordonnée sous forme de couples **clé/valeur**.

La **clé** peut être de type entier ou de type chaîne de caractères.

La **valeur** associée à la clé peut être de n'importe quel type, et notamment de type tableau. Dans ce cas, le tableau est dit **multidimensionnel**.

## 6- Les variables de type composé

### Les tableaux numériques

Pour créer ce tableau numérique, il faut écrire :

```
$tableau = array("Jean", "Robert", "Paul", "Joe", "Alain");
```

**Ou bien**

```
$tableau = ["Jean", "Robert", "Paul", "Joe", "Alain"];
```

Clé	Valeur
0	Jean
1	Robert
2	Paul
3	Joe
4	Alain

## 6- Les variables de type composé

### Les tableaux numériques

Pour récupérer directement la valeur d'un tableau grâce à son indice :

```
echo $tableau[0];           // Ce code affiche la valeur associée à l'indice 0
```

**Attention** : Les indices d'un tableau commencent toujours par 0

Pour remplacer la valeur que possède un indice par une autre :

```
$tableau[indice] = nouvelle_valeur;
```

**PRATIQUONS !**

## 6- Les variables de type composé

### Les tableaux numériques

Il est possible de créer un tableau vide et de le remplir de cette façon :

```
1  <?php
2      $tab = array();
3      $tab[0] = 'Jean-Claude';
4      $tab[1] = 'Samba';
5      $tab[2] = 'Sophie';
6      $tab[3] = 'Mounia';
7
8      print_r($tab);
9
10  ?>
```



## 6- Les variables de type composé

### Les tableaux associatifs

On parle de tableau associatif lorsqu'on choisit de redéfinir les clés d'un tableau.

```
$tab = [  
    "A1"    => "Jean",  
    "B4"    => "Robert",  
    3       => "Paul",  
    "Toto"  => "Joe",  
    "H"     => "Alain"  
];  
var_dump($tab);
```

Clé	Valeur
A1	Jean
B4	Robert
3	Paul
Toto	Joe
H	Alain

```
var_dump($tab);
```

```
};
```

## 6- Les variables de type composé

### Les tableaux associatifs à doubles dimensions

Le principe est d'inclure un tableau dans un autre sous forme de valeur.

```
$tab = [  
    "jours"    => ["Lundi", "Mardi", "Mercredi"],  
    "mois"     => ["Janvier", "Février", "Mars"],  
    "années"   => [1999, 2000, 2001],  
];
```

## 6- Les variables de type composé

### Les tableaux associatifs à doubles dimensions

Le principe est d'inclure un tableau dans un autre sous forme de valeur.

```
1  <?php
2  $villes_bresil = ["Sao Polo", "Brasilia"];
3  $villes_france = ["Paris", "Nice"];
4
5  $pays = [];
6
7  $pays['Bresil'] = $villes_bresil;
8  $pays['France'] = $villes_france;
9
10 print_r($pays);
11
12 ?>
```

**PRATIQUONS !**

# 7- Les opérateurs et les différentes opérations possibles en php

## Les opérateurs arithmétiques :

- L'opérateur se nomme + pour l'addition
- L'opérateur se nomme – pour la soustraction
- L'opérateur se nomme \* pour la multiplication
- L'opérateur se nomme / pour la division
- L'opérateur se nomme % pour le modulo (Le reste de la division entière)
- L'opérateur se nomme \*\* pour la l'exponentiation
- Les opérateurs d'incrémentation se nomment ++ et +=
- Les opérateurs de décrémentation se nomment -- et -=

## 7- Les opérateurs et les différentes opérations possibles en php

```
1  <?php
2      $nombre1 = 2;
3      $nombre2 = 2;
4
5      echo $nombre1 + $nombre2 . "\n";
6      echo $nombre1 - $nombre2 . "\n";
7      echo $nombre1 * $nombre2 . "\n";
8      echo $nombre1 / $nombre2 . "\n";
9      echo $nombre1 % $nombre2 . "\n";
10     echo $nombre1 ** $nombre2 . "\n";
11
12  ?>
```

```
4
0
4
1
0
4
```

```
}>
```

```
}>
```

## 7- Les opérateurs et les différentes opérations possibles en php

```
1 <?php
2     $nombre = 5;
3     $nombre++;
4     echo $nombre . "\n"; //6
5
6     $nombre--;
7     echo $nombre . "\n"; //5
8 ?>
```

```
1 <?php
2     $code = 25;
3
4     $code += 3;
5     echo $code . "\n"; //28
6
7     $code -= 20;
8     echo $code . "\n"; //8
9 ?>
```

# 7- Les opérateurs et les différentes opérations possibles en php

## - La concaténation

Le principe de la concaténation est de coller plusieurs chaînes de caractères différentes pour n'en faire qu'une seule. Son opérateur est le `.`

```
1  <?php
2      $chaine1 = "Hello";
3      $chaine2 = "World";
4
5      $concat = $chaine1 . ' ' . $chaine2;
6
7      echo $concat;           // Hello World
8  ?>
```



# 7- Les opérateurs et les différentes opérations possibles en php

## - Conversion de chaîne en nombre

PHP est capable de convertir une chaîne en nombre (entier ou décimal).

```
index.php
1  <?php
2  echo '1 + "1" = ' . (1 + "1") . '<br/>';
3  echo '1 + "1.5" = ' . (1 + "1.5") . '<br/>';
4  echo '1 + 1abc = ' . (1 + "1abc") . '<br/>';
5  echo '1 + "1.5abcd" = ' . (1 + "1.5abcd") . '<br/>';
6  echo '1 + "1.5 abcd" = ' . (1 + "1.5 abcd") . '<br/>';
7  echo '1 + ".5" = ' . (1 + ".5") . '<br/>';
8  echo '1 + "-5" = ' . (1 + "-5") . '<br/>';
9  echo '1 + " \t\n\r 5" = ' . (1 + " \t\n\r5") . '<br/>';
10 echo '1 + "abc1" = ' . (1 + "abc1") . '<br/>';
11 ?>
```

```
11  >>
```

```
10  echo '1 + "abc1" = ' . (1 + "abc1") . '<br/>';
```

**PRATIQUONS !**

## 8- Les conditions

La structure de contrôle `if` permet une exécution conditionnelle d'instructions.

```
if ( condition ) {  
    // instructions;  
} else if ( condition ) {  
    // instructions;  
} else {  
    // instructions;  
}
```

## 8- Les conditions

Les conditions des clauses **if** et **elseif** sont testées séquentiellement.

**Si** la condition est **vraie alors** les **instructions** associées de la clause sont **exécutées**.

**Si aucune des conditions n'est vraie**, ce sont plutôt les **instructions** éventuelles définies dans la clause **else** qui seront **exécutées**.

## 8- Les conditions

Exemple :

```
1  <?php
2  $nom = "Zidane";
3
4  if($nom)
5  {
6      echo "Zidane existe";
7  } else {
8      echo "Zidane n'existe pas";
9  }
10
11 //Zidane existe
12 ?>
```

```
13 }>
14 //Zidane existe
```

## 8- Les conditions

### Exemple :

```
1  <?php
2  $ville = "Paris";
3
4  if($ville == "paris"){
5      echo "Nous sommes à paris";
6  } elseif($ville == "Paris"){
7      echo "Nous sommes à Paris";
8  } else {
9      echo "Nous sommes à Saint-Tropez";
10 }
11
12 // Nous sommes à Paris
13 ?>
```

```
13  >>
15  // nous sommes à paris
```

**PRATIQUONS !**

## 8- Les conditions

Les structures ternaires ont également pour rôle d'écrire des conditions.

Elles sont souvent utilisées lorsqu'on a envie que notre test tienne sur une seule ligne ou qu'il soit plus simple à lire.

Cette structure a pour syntaxe :

*test ? valeur à retourner si true : valeur à retourner si false*



## 8- Les conditions

### Exemple :

```
1  <?php
2
3  $heure = "12";
4
5  $message = ($heure==11? "Il est midi" : "Il n'est pas encore midi" );
6
7  echo $message;
8
9  // Il n'est pas encore midi
10 ?>
```

## 8- Les conditions

L'instruction **switch case** nous permet également de poser des conditions.

Elle est utile lorsque nous voulons étudier plusieurs cas possibles.

Cependant, le **switch case** possède une syntaxe un peu différente de celle des conditions qu'on a pu étudier jusqu'à présent.

De plus, cette instruction ne gère que l'égalité dans la comparaison. On ne peut donc pas tester les inégalités.

## 8- Les conditions

```
switch (variable) {  
    case 'value':  
        # code...  
        break;  
  
    default:  
        # code...  
        break;  
}
```

```
$var = 24;  
  
switch($var){  
    case 12 :  
        echo 'Il est midi';  
        break;  
  
    default:  
        echo 'Il est x heure?';  
        break;  
}
```

## 9- Les boucles

Une boucle est une structure de contrôle qui permet de répéter l'exécution d'une séquence d'instructions.

Elles sont très pratiques.

La boucle while (tant que) ;

La boucle do... while (faire... tant que) ;

La boucle for (pour) ;

La boucle foreach (pour chaque) ;

## 9- Les boucles

La boucle while (tant que )

```
1  <?php
2  //La boucle while
3  $x = 2;
4
5  while($x<10)
6  {
7      echo $x . "\n";
8      $x++;
9  }
10 ?>
```

2  
3  
4  
5  
6  
7  
8  
9

## 9- Les boucles

La boucle do while (faire, tant que)

```
1  <?php
2  //La boucle do while
3
4  $x = 2;
5  do{
6      $x++;
7      echo $x;
8  }
9  while($x>=10); //3
10
11  ?>
```

# 9- Les boucles

## La boucle for (pour)

```
1  <?php
2  //La boucle for
3
4  $tab = ["Bali", "Tokyo", "Abidjan"];
5
6  for($i=0; $i<=2; $i++)
7  {
8      echo "J'adore " . $tab[$i] . "\n";
9  }
10
11  ?>
```

J'adore Bali  
J'adore Tokyo  
J'adore Abidjan

## 9- Les boucles

La boucle foreach (pour chaque)

```
1  <?php
2  //La boucle foreach
3
4  $tab = ["Bali", "Tokyo", "Abidjan"];
5
6  foreach ($tab as $city) {
7      echo "J'adore ". $city . "\n";
8  }
9
10 ?>
```

```
J'adore Bali
J'adore Tokyo
J'adore Abidjan
```



**PRATIQUONS !**

## 10- Les fonctions natives

Les fonctions natives sont des fonctions qui sont déjà créées par PHP.

En tant que développeur, nous pouvons nous servir de ces fonctions pour effectuer des opérations bien précises de manière plus simple et plus rapide.

## 10- Les fonctions natives

**COUNT()** Pour déterminer la longueur d'un tableau.

```
<?php  
  
$tab = ['a', 'b', 'c'];  
  
echo count($tab); //3  
  
?>
```

```
<>
```

## 10- Les fonctions natives

**EXPLODE()** Cette fonction permet de découper une chaîne de caractères en utilisant un séparateur puis stocke les données dans un tableau.

```
<?php
$data = "1,2,3,4,5";

$tab = explode(",", $data);

print_r($tab);
?>
```

```
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)
```

## 10- Les fonctions natives

**IMPLode()** Cette fonction permet de regrouper les valeurs d'un tableau dans une chaîne en utilisant un séparateur.

```
<?php
$tab = ["C'est", "cool"];

echo implode(" ", $tab);
?>
```

C'est cool

```
>>
```

## 10- Les fonctions natives

**STR\_REPLACE()** Cette fonction remplace un morceau d'une chaîne de caractères par un autre.

```
<?php
$phrase = "J'adore la ville de Paris";

$phrase_modif = str_replace("Paris", "Nice", $phrase);

echo $phrase_modif; //J'adore la ville de Nice
?>
```

```
>>
```

```
echo $phrase_modif; //J'adore la ville de Nice
```

## 10- Les fonctions natives

**TRIM()** permet de supprimer les espaces vides en début et fin de caractères.

```
<?php
    $prenom = " Bertrand ";
    echo $prenom . "\n"; // Bertrand

    $prenom_modif = trim($prenom);

    echo $prenom_modif; //Bertrand
?>
```

```
>>
```

## 10- Les fonctions natives

**ISSET()** permet de tester l'existence d'une variable. Elle retourne true si la variable existe et false dans le cas contraire.

```
$ville = "Paris";  
if(isset($ville))  
{  
    echo "Paris existe"; //Paris existe  
} else {  
    echo "Paris n'existe pas";  
}
```



## 10- Les fonctions natives

**EMPTY()** La fonction **empty()** permet de tester si une variable est nulle ou pas. Elle retourne true si la valeur est nulle, false si la valeur contient des données.

```
<?php
$ville = "Paris";

if(empty($ville) )
{
    echo "Couvre-feu";
} else{
    echo "C'est le déconfinement";
}

//C'est le déconfinement
```

```
//C'est le déconfinement
```

## 10- Les fonctions natives

**TIME()** Retourne un nombre exprimé en secondes. Ce nombre représente le temps écoulé depuis le 1 Janvier 1970 GMT.

Ce nombre est également appelé timestamp UNIX.

Cette fonction constitue la base de calcul d'autres fonctions comme la fonction date() par exemple.

**DATE():** Cette fonction permet d'afficher la date enregistrée à un instant t.

Cette fonction prend en paramètres, le mode d'affichage de la date.

## 10- Les fonctions natives

```
<?php  
echo date('d.m.y');  
?>
```

```
;>
```

## 10- Les fonctions natives

Cette fonction date peut également prendre en paramètres le timestamp UNIX pour définir une autre date que la date du jour.

```
<?php
    $semaine_prochaine = time() + (7 * 24 * 60 * 60);
    $date = date("d-m-y", $semaine_prochaine);
    echo $date;
?>
```

```
>>
```

## 10- Les fonctions natives

**MKTIME()** Retourne le timestamp UNIX à partir d'une date passée en paramètre.

```
<?php
```

```
    echo mktime(0,0,0,3,3,1993);
```

```
?>
```

```
>>
```

## 10- Les fonctions natives

**CHECKDATE()** Cette fonction indique si une date est valide ou non. Elle tient compte des années bissextiles. Elle prend en paramètres le mois, le jour et l'année et renvoie vrai ou faux.

**\$valide = checkdate(\$mois, \$jour, \$annee)**

```
$date = checkdate(2, 31, 2021);  
  
if ($date) {  
    echo "Cette date est valide.";  
} else {  
    echo "Cette date n'est pas valide.";  
}  
//Cette date n'est pas valide.
```

```
\\Cetfe qate n'est pas valide.
```

## 10- Les fonctions natives

Nous avons vu jusqu'à maintenant des fonctions natives.

C'est-à-dire des fonctions qui sont déjà prévues pour nous par php.

Cependant, pour des actions bien spécifiques, il est possible de créer nos propres fonctions.

L'utilité de créer nos propres fonctions est d'écrire du code réutilisable.

Ce qui évite donc la répétition inutile.

## 11- Création des fonctions pouvant retourner une valeur

```
<?php
function calcul_du_poids($masse, $pesanteur)
{
    $poids = $masse * $pesanteur;
    return $poids;
}

// Calcul du poids d'un objet Obj1
echo calcul_du_poids(70, 9.81);

?>
```

```
;>
```

```
echo calcul_du_poids(70, 9.81);
```



## 11- Création des fonctions pouvant retourner une valeur

Le calcul du poids aurait très bien pu se faire directement dans le fichier et sans passer par une fonction.

Cependant, l'avantage de le faire à l'aide d'une fonction nous fait gagner du temps et de l'énergie. La fonction représente donc une formule que l'on peut utiliser pour réaliser autant d'opérations souhaitées.

Par conséquent, à partir de maintenant, à chaque fois que nous souhaiterons calculer le poids d'un autre objet, on n'a plus besoin de repasser une nouvelle fois l'opération mais simplement d'appeler la fonction en lui passant les nouveaux paramètres.

# 11- Création des fonctions pouvant retourner une valeur

```
<?php
function calcul_du_poids($masse, $pesanteur)
{
    $poids = $masse * $pesanteur;
    return $poids;
}

// Calcul du poids d'un objet Obj1
echo calcul_du_poids(70, 9.81) . "<br/>";

// Calcul du poids d'un objet Obj2
echo calcul_du_poids(35, 9.81) . "<br/>";

// Calcul du poids d'un objet Obj2
echo calcul_du_poids(22, 9.81) . "<br/>";

?>
```

```
?>
```

```
echo calcul_du_poids(22, 9.81) . "<br/>";
```

# 11- Création des fonctions pouvant retourner une valeur

Il est également possible de donner des valeurs par défaut aux paramètres des fonctions.

```
<?php

function saluer($prenom, $nom, $type_de_salutation = "Salut")
{
    $phrase = $type_de_salutation . " " . $prenom . " " . $nom . "\n";
    return $phrase;
}

echo saluer("John", "Doe") ;           //Salut John Doe
echo saluer("John", "Doe", "Bonjour"); //Bonjour John Doe
echo saluer("John", "Doe", "Bonsoir"); //Bonsoir John Doe
```

```
echo saluer("John", "Doe", "Bonsoir"); //Bonsoir John Doe
```

# 11- Création des fonctions pouvant retourner une valeur

En reprenant, l'exemple du calcul du poids, si on part sur le principe que l'intensité de la pesanteur est  $\simeq$  à  $9,81 \text{ N.kg}^{-1}$

```
<?php
function calcul_du_poids($masse, $pesanteur = 9.81)
{
    $poids = $masse * $pesanteur;
    return $poids;
}

// Calcul du poids d'un objet Obj1
echo calcul_du_poids(70) . "<br/>";

// Calcul du poids d'un objet Obj2
echo calcul_du_poids(35) . "<br/>";

// Calcul du poids d'un objet Obj2
echo calcul_du_poids(22) . "<br/>";

?>
```

```
?>
echo calcul_du_poids(35) . "<br/>";
// Calcul du poids d'un objet Obj2
echo calcul_du_poids(22) . "<br/>";
```

**PRATIQUONS !**

## 12- Création des fonctions pouvant ne rien retourner comme valeur

Il est possible de créer une fonction qui effectue des opérations et qui ne retourne rien à la fin.

Dans ce cas, pour plus de clarté, il est également conseillé de mettre **:void**

```
<?php

function saluer() : void
{
    echo "Hello World!";
}

saluer();

?>
```

```
>>
```

```
saluer()?
```

**PRATIQUONS !**

## 13- Les fonctions et les types

Il est possible de définir le type des paramètres et de la valeur de retour d'une fonction. Cela permet d'avoir un code plus clair et plus précis.

```
<?php  
  
function somme(int $val1, int $val2):int  
{  
    return $val1 + $val2;  
}  
  
echo somme(1, 1); //2
```

```
echo somme(1, 1); //2
```



## 13- Les fonctions et les types

Ce code type les **paramètres** de la fonction **somme** en précisant qu'ils doivent être des nombres entiers.

Puis le **: int** signifie que la fonction est censée retourner une valeur de type int.

# 13- Les fonctions et les types

```
<?php
```

```
function somme(int $val1, int $val2):array  
{  
    return $val1 + $val2;  
}
```

```
echo somme(1, 1);
```

```
//Fatal error: Uncaught TypeError:
```

```
//Return value of somme() must be of the type array, int returned
```

```
\\βεfουεu λαgνε οf zοmmε() wηzε ρε οf fηε fηβε αηεαλ' ηuf uεfουεuεq
```

```
\\εαfεη εηου: ηucεηεμf ιλβεfουου:
```

```
εουη uεfουεuεq
```

**PRATIQUONS !**

## 14- La portée des variables

Cette notion est super importante car elle est source de beaucoup d'erreurs.

Les variables déclarées ou passées en paramètres d'une fonction ne sont reconnues que dans la fonction.

De la même façon, une variable déclarée en dehors de la fonction n'est pas reconnue dans la fonction.

## 14- La portée des variables

```
<?php
```

```
function afficher():void  
{  
    $nom = "JC";  
}
```

```
echo $nom;
```

```
//Notice: Undefined variable:
```

```
//nom in C:\wamp64\www\prof\index.php on line 8
```

```
//nom in C:\wamp64\www\prof\index.php on line 8
```

## 14- La portée des variables

```
$nom = "jc";

function afficher()
{
    return $nom;
}

echo afficher();
//Notice: Undefined variable: nom in C:\wamp64\www\prof\index.php on line 18
```

//Notice: Undefined variable: nom in C:\wamp64\www\prof\index.php on line 18

## 14- La portée des variables

Le mot-clé **global** permet de résoudre ce problème. Cela a pour effet de pouvoir utiliser la variable partout dans le code.

```
$nom = "jc";  
function afficher()  
{  
    global $nom;  
    return $nom;  
}  
echo afficher(); //jc
```

```
echo afficher(); //jc
```

**PRATIQUONS !**



**PHP ET HTML**

# 1- Inclusion du contenu d'un fichier dans un autre

L'instruction **include** permet d'inclure le contenu d'un fichier dans un autre.

L'objectif étant d'avoir du code réutilisable.

Le principe est donc de se positionner dans le fichier dans lequel l'on désire inclure un autre fichier.

Puis, l'inclure grâce à la fonction **include**.

En utilisant la fonction include, un fichier peut être inclu plusieurs fois sur une même page.

# 1- Inclusion du contenu d'un fichier dans un autre



The screenshot shows a code editor with two tabs: 'fichier1.php' (active) and 'fichier2.php'. The active tab contains the following PHP code:

```
fichier1.php
1  <?php
2
3      $ville1 = "Dubai";
4      $ville2 = "Lisbonne";
5
```



The screenshot shows the same code editor with the tabs swapped: 'fichier2.php' (active) and 'fichier1.php'. The active tab contains the following PHP code:

```
fichier2.php
1  <?php
2
3  include "fichier1.php";
4
5  echo $ville1 . " et " . $ville2; //Dubai et Lisbonne
6
```

# 1- Inclusion du contenu d'un fichier dans un autre

**NB:**

```
index.php
1  <?php
2      include "test.php";
3
4      echo "bim";
5  ?>
```

Si le fichier "**test.php**" à inclure **n'existe pas** ou que le chemin défini est erroné, **PHP** va quand même **exécuter** l'instruction de la **ligne 4**.

C'est-à-dire **signaler l'erreur** de la ligne 2 puis **afficher "bim"** à l'écran.

**PRATIQUONS !**

# 1- Inclusion du contenu d'un fichier dans un autre

L'instruction **include\_once** procède presque de la même manière que **Include**.

A la différence que celle-ci permet d'inclure le fichier une seule fois sur une même page.

L'objectif est donc d'éviter d'inclure le même fichier plusieurs fois.

# 1- Inclusion du contenu d'un fichier dans un autre

L'instruction **require** permet d'inclure le contenu d'un fichier dans un autre.

L'objectif étant d'avoir du code réutilisable.

Le principe est donc de se positionner dans le fichier dans lequel l'on désire inclure un autre fichier et réaliser cette action grâce à la fonction **require**.

En utilisant la fonction `require`, un fichier peut être inclu plusieurs fois sur une même page.

# 1- Inclusion du contenu d'un fichier dans un autre

**NB:**

```
index.php
1  <?php
2      require "test.php";
3
4      echo "bim";
5  ?>
```

**Si** le fichier "**test.php**" à inclure **n'existe pas** ou que le chemin défini est erroné, **PHP n'exécutera pas** l'instruction de la **ligne 4**.

C'est-à-dire l'affichage de "bim" à l'écran ne se fera pas.



# 1- Inclusion du contenu d'un fichier dans un autre

L'instruction **require\_once** procède presque de la même manière que **require**.

A la différence que celle-ci permet d'inclure le fichier une seule fois sur une même page.

L'objectif est donc d'éviter d'inclure le même fichier plusieurs fois.

**PRATIQUONS !**

## 2 - La variable super globale \$\_GET

\$\_GET est une variable superglobale de type tableau.

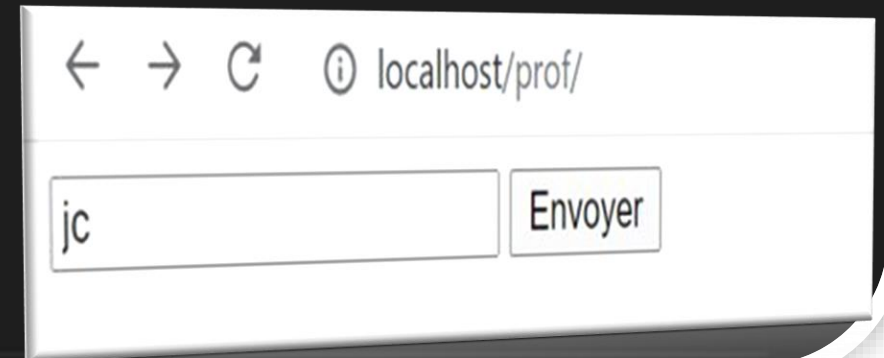
Ce tableau est rempli si et seulement si les données sont envoyées via la méthode GET.

Une fois rempli, \$\_GET contient des données sous forme de clé-valeur.

\$\_GET va donc être très intéressant et utile car cette variable va servir à transiter des données d'une page à une autre.

## 2 - La variable super globale \$\_GET

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <form method="get" action="backend.php">
    <input type="text" name="name"/>
    <input type="submit" name="send_button"/>
  </form>
</body>
</html>
```



← → ↻ ⓘ localhost/prof/

Nous avons un fichier index.php dans lequel se trouve un formulaire. En cliquant sur le bouton "envoyer", les données du formulaire sont envoyées vers le fichier "backend.php".

## 2 - La variable super globale \$\_GET

backend.php

```
1  <?php
2  echo $_GET["name"];
3  ?>
```

← → ↻ ⓘ localhost/prof/backend.php?name=jc&send\_button=Envoyer

jc

Dans backend.php, nous pouvons donc afficher à la valeur associée à la clé "name" de \$\_GET.

**PRATIQUONS !**

## 3 - La variable super globale \$\_POST

**\$\_POST** est une variable superglobale de type tableau.

Ce tableau est rempli si et seulement si les données sont envoyées via la méthode POST

Une fois rempli, \$\_POST contient des données sous forme de clé-valeur. \$\_POST va donc être très intéressant et utile car cette variable va servir à transiter des données d'une page à une autre.

Contrairement à la méthode GET, les données transmises ne sont pas visibles dans la barre d'url. Ce qui est plus sécurisant.

# 3 - La variable super globale \$\_POST

```
mp
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <form method="post" action="backend.php">
      <input type="text" name="name"/>
      <input type="submit" name="send_button"/>
    </form>
  </body>
</html>
```

← → ↻ ⓘ localhost/prof/

jc

Envoyer



### 3 - La variable super globale \$\_POST

```
backend.php  
1  <?php  
2  echo $_POST["name"];  
3  ?>
```

← → ↻ ⓘ localhost/prof/backend.php

jc

Dans backend.php, nous pouvons donc afficher la valeur associée à la clé "name" de \$\_POST.

**PRATIQUONS !**

## 4 - La variable super globale \$\_FILES

**\$\_FILES** est une variable superglobale.

C'est un tableau associatif contenant des informations sur les fichiers transmis au serveur web comme les images, document word, pdf...

Ce tableau contient le nom, le type, l'erreur et la taille du fichier envoyé.

Lorsque nous chargeons un fichier sur un site, ces fichiers sont donc stockés dans **\$\_FILES**.

# 4 - La variable super globale \$\_FILES

index.php

```
1  <?php
2      var_dump($_FILES);
3  ?>
4
5  <!DOCTYPE html>
6  <html lang="en">
7  <head>
8      <meta charset="UTF-8">
9      <meta http-equiv="X-UA-Compatible" content="IE=edge">
10     <meta name="viewport" content="width=device-width, initial-scale=1.0">
11     <title>Document</title>
12 </head>
13 <body>
14     <form method="post" enctype="multipart/form-data">
15         <input type="file" name="image" />
16         <input type="submit" />
17     </form>
18 </body>
19 </html>
```

20 <\μfwJ>

# 4 - La variable super globale \$\_FILES

← → ↻ ⓘ localhost/formation/procedural/index.php

C:\wamp64\www\formation\procedural\index.php:2:

```
array (size=1)
```

```
  'image' =>
```

```
    array (size=5)
```

```
      'name' => string 'France.png' (length=10)
```

```
      'type' => string 'image/png' (length=9)
```

```
      'tmp_name' => string 'C:\wamp64\tmp\php8D5C.tmp' (length=25)
```

```
      'error' => int 0
```

```
      'size' => int 329
```

Choisir un fichier

Aucun fichier choisi

Envoyer

**PRATIQUONS !**

## 5 - La variable super globale \$\_COOKIE

- Un cookie est un fichier contenant des informations stockées sur l'ordinateur du visiteur. Chaque navigateur utilise ses propres cookies. Ils peuvent être stockés pendant plusieurs mois et permettent par exemple de s'authentifier à chaque fois que l'on revient sur un site sur lequel l'on s'est inscrit.
- Ne jamais stocker d'informations critiques car ces fichiers sont facilement accessibles.

## 5 - La variable super globale \$\_COOKIE

- **\$\_COOKIE** est aussi un tableau superglobal donc il est valable sur toutes les pages de votre site.
- Pour créer un cookie, il faut utiliser l'instruction **setcookie()**:

```
setcookie ( string $name , string $value = "" , int $expires = 0 , string $path = "" , string $domain = "" , bool $secure = false , bool $httponly = false ) : bool
```



**PRATIQUONS !**

## 6 - La variable super globale `$_SERVER`

- `$_SERVER` est un tableau contenant des informations comme les entêtes, dossiers et chemins du script.
- Les entrées de ce tableau sont créées par le serveur web.
- `var_dump($_SERVER);`

## 7 - La variable super globale `$_SESSION`

- C' est un tableau associatif permettant de stocker n'importe quelle valeur propre à chaque utilisateur. Ce tableau sera valable dans toutes les pages PHP du site web et vous permet donc de transmettre des variables d'une page à l'autre. Attention de ne pas trop en abuser car si beaucoup de personnes se connectent à votre site, vous risquez de remplir la mémoire de votre serveur web.
- **`$_SESSION`** est la variable superglobale dans laquelle sont stockées les différentes sessions.
- `var_dump($_SESSION);`

**PRATIQUONS !**