- Si le fichier demandé contient du code PHP, le serveur web transmet ce fichier à l'interpréteur PHP pour traitement.

3. **Exécution du script PHP :**

- Le script PHP peut :
  - ✓ Récupérer des données depuis une base de données (MySQL dans notre cas).
  - ✓ Traiter les données envoyées par l'utilisateur (via un formulaire, une requête ,...).
  - ✓ Effectuer des calculs ou des traitements sur les données.
- PHP génère ensuite un fichier HTML contenant du contenu dynamique basé sur les résultats des traitements.

4. **Réponse envoyée au navigateur :**

- Une fois le script PHP exécuté, le serveur web renvoie le résultat (un fichier HTML avec le contenu généré dynamiquement) au navigateur de l'utilisateur.
- Le navigateur interprète ce fichier HTML et affiche le contenu.

III. Environnement de travail

XAMPP Lite est une version légère de XAMPP, un environnement de développement local qui regroupe tous les outils nécessaires pour exécuter des scripts PHP et interagir avec des bases de données MySQL sur un serveur local. Il est composé principalement de :

- ❖ **Apache** : Serveur web qui exécute les fichiers PHP.
- ❖ **PHP** : Langage de programmation côté serveur pour générer des pages dynamiques.
- ❖ **MySQL** : Système de gestion de bases de données relationnelles.
- ❖ **PhpMyAdmin** : Interface graphique pour gérer les bases de données MySQL.

IV. Les base d'un script PHP

Un script PHP est composé de différents éléments structurés pour exécuter des instructions côté serveur. Voici les principaux composants d'un script PHP :

1. Déclaration du code PHP :

Le code PHP doit être placé entre les balises de délimitation suivantes :

```php
<?php
//Code PHP ici
echo 'bonjour' ;
?>
```

La balise d'ouverture <?php indique au serveur que tout ce qui suit doit être interprété comme du PHP. La balise de fermeture ?> est utilisée pour terminer le script.

Remarques :

- Le **//** est utiliser pour faire des commentaires. Les commentaires permettent de documenter le code.
- Les instructions PHP terminent généralement par un point-virgule ;
- **echo** Affiche une ou plusieurs expressions séparer par des points

2. Variables :

Les variables en PHP commencent par le symbole **$** et servent à stocker des données :

```php
<?php
$nom = "Mohamed"; // Variable qui contient une chaîne de caractères
$age =    25;  //   Variable  qui  contient  un   entier
?>
```

3. typage et fonctions prédéfinies :

a. Les nombres

En PHP, les nombres sont des types de données utilisés pour effectuer des calculs mathématiques. On distingue deux principaux types de nombres :

1. **int** (entiers)
2. **float** (nombres à virgule flottante)

Opérations arithmétiques de base :

Opération	Opérateur	Exemple	Résultat
Addition	+	5 + 3	8
Soustraction	-	5 - 3	2
Multiplication	*	5 * 3	15
Division	/	6 / 2	3
Modulo	%	7 % 3	1 (reste)

Fonctions mathématiques :

Fonction	Description	Syntaxe	Exemple	Résultat
abs()	Retourne la valeur absolue d'un nombre.	abs(nombre)	abs(-15)	15
sqrt()	Calcule la racine carrée d'un nombre.	sqrt(nombre)	sqrt(16)	4
rand()	Génère un nombre entier aléatoire entre deux bornes.	rand(min, max)	rand(1, 100)	Un nombre entre 1 et 100
round()	Arrondit un nombre à l'entier le plus proche ou à une précision définie.	round(nombre, précision)	round(3.14159, 2)	3.14

b. Les chaînes de caractères en PHP

En PHP, les chaînes de caractères (ou **strings**) sont des séquences de caractères. Vous pouvez effectuer plusieurs opérations sur les chaînes de caractères.

La concaténation de chaînes permet de combiner plusieurs chaînes de caractères en une seule. En PHP, on utilise l'opérateur **"."** pour cela.

Exemple de concaténation :

```php
<?php
$prenom = "Amine";
$nom = "Tounsi";
$nom_complet = $prenom . " " . $nom; echo $nom_complet;

// Affiche : Amine Tounsi
?>
```

Fonctions prédéfinies pour les chaînes de caractères

Fonction	Description	Exemple	Résultat
chr()	Retourne le caractère correspondant à un code ASCII.	`chr(65)`	A
ord()	Retourne le code ASCII du premier caractère.	`ord("A")`	65
strlen()	Retourne la longueur d'une chaîne.	`strlen("Hello")`	5
substr()	Retourne une sous-chaîne d'une chaîne.	`substr("Hello", 1, 3)`	ell
strpos()	Retourne la position de la première occurrence d'une sous-chaîne.	`strpos("Hello", "e")`	1

Fonction	Description	Exemple	Résultat
strcmp()	Compare deux chaînes.	`strcmp("apple", "banana")`	-1
str_replace()	Remplace une sous-chaîne par une autre.	`str_replace("World", "PHP","Hello, World!")`	Hello, PHP!
strtolower()	Convertit une chaîne en minuscules.	`strtolower("HELLO")`	hello
strtoupper()	Convertit une chaîne en majuscules.	`strtoupper("hello")`	HELLO
trim()	Supprime les espaces au début et à la fin d'une chaîne.	`trim(" Hello ")`	Hello

c. Les booléens :

En PHP, les **booléens** sont utilisés pour représenter deux valeurs possibles : true (vrai) et false (faux). Ils sont principalement utilisés dans les conditions et les boucles.

Les opérateurs de comparaison

Opérateur	Description	Exemple	Résultat
==	Vérifie si deux valeurs sont égales (sans tenir compte du type).	5 == "5"	true
<>	Vérifie si deux valeurs ne sont **pas** égales.	5 <> 3	true
>	Vérifie si la première valeur est **plus grande** que la deuxième.	5 > 3	true
<	Vérifie si la première valeur est **plus petite** que la deuxième.	5 < 3	false
>=	Vérifie si la première valeur est **plus grande ou égale** à la deuxième.	5 >= 5	true
<=	Vérifie si la première valeur est **plus petite ou égale** à la deuxième.	3 <= 5	true

Les opérateurs logiques

Opérateur	Description	Exemple	Résultat
&&	**ET logique** : Retourne true si **les deux** expressions sont vraies.	(5 > 3) && (8 < 10)	true
||	**OU logique :** Retourne true si **l'un de deux** expression est vraies.	5 == 2 || 4> 3	true
!	**NON logique** : Inverse la valeur booléenne d'une expression.	!(5 > 3)	false

d. Fonctions des date/heure :

En PHP, les fonctions de **date/heure** sont utilisées pour manipuler et formater les dates et heures. Voici un aperçu des principales fonctions pour travailler avec la date et l'heure :

Fonction	Description	Exemple	Résultat
checkdate(mois, jour, année)	Vérifie si une date est valide. Retourne **true** si oui, sinon **false**.	checkdate(2, 29, 2023)	false (2023 n'est pas bissextile)
date(format [, timestamp])	Retourne une date formatée selon le format indiqué.	date("d/m/Y")	19/08/2025(varie selon la date système)
time()	Retourne le timestamp Unix courant (nombre de secondes depuis 01/01/1970).	time()	1766208904 (varie selon la date système)
strtotime(string [, timestamp])	Convertit une chaîne en **timestamp Unix**.	strtotime("2025-08-19")	1755571200

e. Les tableaux :

En PHP, les **tableaux** sont des structures de données permettant de stocker plusieurs valeurs sous une seule variable. Il existe deux types principaux de tableaux : les **tableaux indexés** et les **tableaux associatifs**.

Exemple 1 tableau indexé :

```php
<?php
$fruits = array("Pomme", "Banane", "Orange");
echo $fruits[0]; // Pomme
echo $fruits[2]; // Orange
}
?>
```

Exemple 2 tableau associatif :

```php
<?php
$personne = array(
    "nom" => "Ali",
    "age" => 25,
    "ville" => "Tunis"
);
// Accès aux éléments
echo $personne["nom"];    // Ali
echo $personne["ville"]; // Tunis
?>
```

Exemple 3 fonction count :

La fonction count() retourne le nombre d'éléments d'un tableau.

```php
<?php
$fruits = ["Pomme", "Banane", "Orange"];
echo count($fruits); // 3
?>
```

4. Structures de contrôles conditionnelles :

Les structures de contrôle conditionnelles permettent d'exécuter un bloc de code uniquement si une condition est vraie, ou de choisir entre plusieurs blocs selon différentes conditions.

Structure	Description	Exemple
if	Exécute un bloc de code si la condition est vraie	`if($x > 0){ echo "Positif"; }`
if...else	Exécute un bloc si la condition est vraie, sinon un autre bloc	`if($x > 0){ echo "Positif"; } else { echo "Non positif"; }`
if...elseif...else	Permet de tester plusieurs conditions successives	`if($x > 0){ echo "Positif"; } elseif($x < 0){ echo "Négatif"; } else { echo "Zéro"; }`
switch	Compare une expression à plusieurs valeurs possibles	`switch($mois){ case 1: echo "Janvier"; break; case 2: echo "Février"; break; default: echo "Autre mois"; }`

5. Structures itératives :

Les structures itératives permettent **d'exécuter un bloc de code plusieurs fois**, soit un nombre déterminé de fois, soit tant qu'une **condition est vraie**.

Structure	Description	Exemple
for	Exécute un bloc un nombre déterminé de fois	`for($i=1; $i<=5; $i++){ echo $i." "; }`
while	Exécute un bloc tant que la condition est vraie	`$i=1; while($i<=5){ echo $i." "; $i++; }`
do...while	Exécute un bloc au moins une fois puis répète tant que la condition est vraie	`$i=1; do{ echo $i." "; $i++; } while($i<=5);`

6. Les fonctions :

Une **fonction** en PHP est un bloc de code réutilisable qui peut **prendre des paramètres** et **retourner une valeur**. Les fonctions permettent de structurer le code et d'éviter les répétitions.

Syntaxe d'une fonction :

```
function nomDeLaFonction(param1, param2, ...) {
    // Instructions à exécuter
    return valeur; // optionnel
}
```

Exemple :

```
<?php
function addition($a, $b){
    return $a + $b;
}
echo addition(4, 6); // Affiche 10
?>
```

7. Opérateurs de transtypage :

En PHP, les **opérateurs de transtypage** permettent de **convertir explicitement une variable d'un type à un autre**, par exemple transformer une chaîne en entier ou un entier en booléen.

Opérateur	Description	Exemple	Résultat
(int)	Convertit une variable en entier	$x = (int) "123";	123
(float)	Convertit une variable en réel (float)	$x = (float) "12.3";	12.3
(string)	Convertit une variable en chaîne	$x = (string) 123;	"123"
(bool)	Convertit une variable en booléen	$x = (bool) 0;	false
(array)	Convertit une variable en tableau	$x = (array) "texte";	["texte"]

8. Fonctions diverses :

a. die()

La fonction `die()` arrête immédiatement l'exécution du script et peut afficher un message d'erreur. Elle est souvent utilisée pour gérer les erreurs ou vérifier qu'une condition critique est remplie avant de continuer.

Exemple :

```
<?php
$nom = "";
if($nom == ""){
    die("Erreur : le nom est vide !");
} ?>
```

⇨ Dans cet exemple, si $nom est vide, le script s'arrête et affiche le message d'erreur.

b. isset()

La fonction **isset()** permet de vérifier si une variable existe et n'est pas nulle. Elle retourne **true** si la variable existe et **false** sinon.

Exemple :

```php
<?php
if(isset($age)){
    echo "La variable age existe et vaut $age";
} else {
    echo "La variable age n'existe pas";
}
?>
```

c. require()

La fonction **require()** permet d'inclure un fichier PHP dans le script. Si le fichier n'existe pas ou ne peut pas être chargé, le script s'arrête avec une erreur fatale. C'est utile pour séparer le code en plusieurs fichiers (ex : fonctions, configuration, templates).

Exemple :

```php
<?php
require("config.php"); // inclut le fichier config.php
echo "Fichier inclus avec succès";
?>
```

V. Principe de transmission de données entre des pages web

Lorsqu'une application web nécessite de transmettre des informations d'une page à une autre, PHP utilise des mécanismes adaptés. Les données peuvent être transmises via des **méthodes HTTP** (comme GET ou POST), et PHP les rend accessibles via des **variables superglobales** comme $_GET et $_POST.

1. La méthode GET :

La méthode **GET** envoie les données via l'**URL**. Elle est visible dans la barre d'adresse et convient pour des informations non sensibles.

Exemple :

Page html :

```html
<form action="traitement_get.php" method="get">
    Nom : <input type="text" name="nom">
    <input type="submit" value="Envoyer">
</form>
```

Si l'utilisateur saisit "Ali" dans le formulaire, l'URL devient :

traitement_get.php?nom=Ali

Page PHP :

```php
<?php
// Récupérer la valeur de la variable nom
$nom = $_GET['nom'];
echo "Bonjour, $nom !";
?>
```

2. La méthode POST :

La méthode **POST** envoie **les données dans le corps de la requête**, ce qui est plus sûr pour les informations sensibles (mot de passe, email...).

Exemple :

Page html :

```html
<form action="traitement_post.php" method="post">
    Nom : <input type="text" name="nom">
    <input type="submit" value="Envoyer">
</form>
```

Dans ce cas, la donnée **n'apparaît pas** dans l'URL.

Page PHP :

```php
<?php
// Récupérer la valeur de la variable nom
$nom = $_POST['nom'];
echo "Bonjour, $nom !";
?>
```

VI. Fonctions PHP pour MySQL

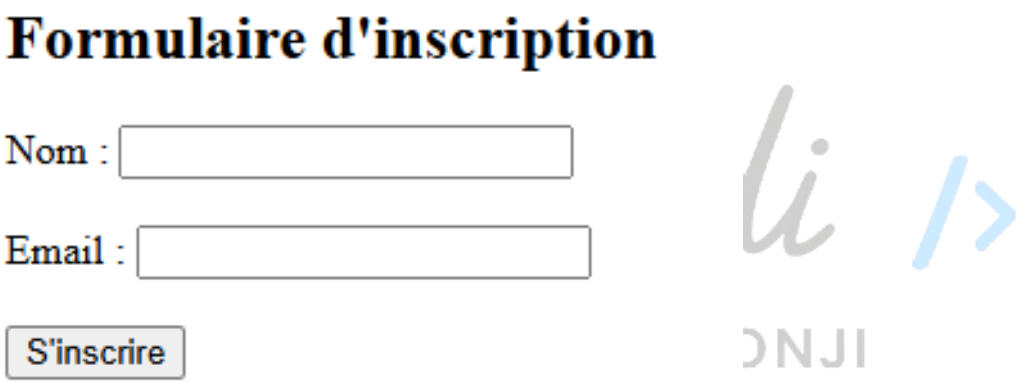
L'extension **MySQLi** fournit un ensemble de fonctions permettant d'interagir avec une base de données MySQL dans une interface procédurale. Voici un aperçu des principales fonctions, leur rôle et des exemples d'utilisation.

Fonction	Description	Exemple
mysqli_connect(host, user, password, db)	Établit une connexion à la base de données MySQL. On peut utiliser or die() pour afficher une erreur si la connexion échoue.	`$conn = mysqli_connect("localhost","root","","test") or die("Connexion échouée");`
mysqli_query(conn, requete)	Exécute une requête SQL sur la connexion donnée. On peut utiliser or die() pour afficher une erreur en cas d'échec.	`$res = mysqli_query($conn, "SELECT * FROM utilisateurs") or die(mysqli_error($conn));`
mysqli_error(conn)	Retourne le message d'erreur de la dernière requête SQL.	`echo mysqli_error($conn);`
mysqli_fetch_array(res)	Récupère une ligne de résultat sous forme de tableau associatif et/ou indexé.	`$row = mysqli_fetch_array($res);` `echo $row['nom'];`
mysqli_fetch_row(res)	Récupère une ligne de résultat sous forme de tableau indexé.	`$row = mysqli_fetch_row($res) ;` `echo $row[1];`
mysqli_num_rows(res)	Retourne le nombre de lignes dans le résultat d'une requête SELECT.	`echo mysqli_num_rows($res);`
mysqli_affected_rows(conn)	Retourne le nombre de lignes affectées par la dernière requête INSERT, UPDATE ou DELETE.	`echo mysqli_affected_rows($conn);`
mysqli_close(conn)	Ferme la connexion à la base de données.	`mysqli_close($conn);`

Exemple 1 : insertion des données d'un formulaire dans un base de données

Soit le formulaire html suivant :

```
<body>
    <h2>Formulaire d'inscription</h2>
    <form action="insertion.php" method="post">
        Nom : <input type="text" name="nom" required><br><br>
        Email : <input type="email" name="email" required><br><br>
        <input type="submit" value="S'inscrire">
    </form>
</body>
```



Partie php : insérer les données dans une table "**utilisateurs**" d'une base de données "**test**".

```php
<?php
// Connexion à la base de données
$conn = mysqli_connect("localhost", "root", "", "test") or
die("Connexion échouée") ;
// Récupération des données du formulaire
$nom = $_POST['nom'];
$email = $_POST['email'];
// Insertion dans la table
$sql = "INSERT INTO utilisateurs (nom, email) VALUES ('$nom','$email')";
if(mysqli_query($conn, $sql)){
    echo "Inscription réussie !";
} else {
    echo "Erreur : " . mysqli_error($conn);
}
// Fermeture de la connexion
mysqli_close($conn);
?>
```

Explications :

1. Le formulaire HTML envoie les données via **POST** à insertion.php.
2. La connexion à la base se fait avec mysqli_connect() et or die() pour gérer l'erreur.
3. La requête SQL INSERT ajoute les données dans la table utilisateurs.
4. mysqli_query() exécute la requête et mysqli_error() permet d'afficher l'erreur si nécessaire.
5. Enfin, mysqli_close() ferme la connexion à la base.

Exemple 2 afficher tous les utilisateurs dans un tableau :

```php
<?php
// Connexion à la base de données
$conn = mysqli_connect("localhost", "root", "", "test") or
die("Connexion échouée : ");
// Requête pour récupérer tous les utilisateurs
$sql = "SELECT * FROM utilisateurs";
$result = mysqli_query($conn, $sql) or die(mysqli_error($conn));
// Vérification s'il y a des résultats
if(mysqli_num_rows($result) > 0){
    echo "<table border='1' cellpadding='5'>";
    echo "<tr><th>ID</th><th>Nom</th><th>Email</th></tr>";
    // Parcours de chaque ligne avec mysqli_fetch_array
    while($row = mysqli_fetch_array($result)){
        echo "<tr>";
        echo "<td>" . $row['id'] . "</td>";
        echo "<td>" . $row['nom'] . "</td>";
        echo "<td>" . $row['email'] . "</td>";
```

```
        echo "</tr>";
    }
    echo "</table>";
} else {
    echo "Aucun utilisateur trouvé.";
}

// Fermeture de la connexion
mysqli_close($conn);
?>
```

Explications

1. **Connexion à la base** : mysqli_connect() avec gestion d'erreur grâce à or die().
2. **Requête SELECT** : SELECT * FROM utilisateurs pour récupérer tous les utilisateurs.
3. **Exécution de la requête** : mysqli_query() retourne un résultat.
4. **Vérification des résultats** : mysqli_num_rows() permet de savoir si la table contient des données.
5. **Affichage dans un tableau HTML** :
    - mysqli_fetch_array() récupère chaque ligne sous forme de **tableau associatif**.
    - On parcourt toutes les lignes avec une boucle while pour remplir le tableau.
6. **Fermeture de la connexion** : mysqli_close() libère les ressource