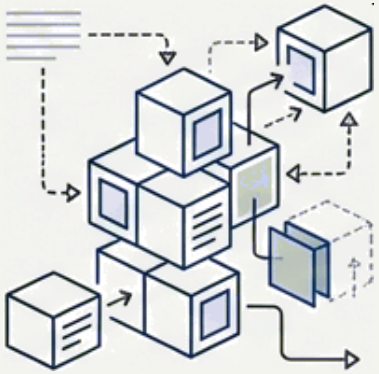
Deux Mondes, Deux Philosophies

Le Site Statique: La Brochure

- Contenu reste inchangé sauf modification manuelle.
- Facile à créer et rapide à héberger.
- Idéal pour les pages ne nécessitant pas de mises à jour fréquentes (sites vitrines, portfolios).

Le Site Dynamique: L'Organisme Vivant



- Pages générées 'à la volée' en fonction de la demande de l'utilisateur.
- Permet l'interactivité, la personnalisation et la gestion de contenu complexe.
- Répond aux actions des utilisateurs et stocke des données.

PHP : Le Cerveau

Le langage de programmation côté serveur. C'est l'intelligence du site : il exécute la logique, traite les formulaires, gère les sessions utilisateur et construit la page avant de l'envoyer.

MySQL : La Mémoire

Le système de gestion de base de données. Il stocke, organise et récupère toutes les informations (utilisateurs, produits, articles) sur demande de PHP.

Conversation Client-Serveur

1. Requête Client:
L'utilisateur demande une page php

2. Traitement Serveur:
Le serveur web APACHE reçoit la requête et sollicite l'interpreteur PHP

3. Dialogue avec la BD:
php execute le script et interroge la BD MySql si Necessaire

4. Réponse de la Base de Donnée:
MySql retourne les données demandées a PHP

5. Génération de la Page:
php utilise les données pour générer le code HTML final de la page

6. Envoi au Client:
Le serveur envoie la page HTML pure au navigateur du client.

NB:
le client ne reçoit jamais le code PHP, le résultat est uniquement **HTML** + **CSS** + **JS**.

Préparer son espace de travail

Editeur de Code

VsCode, DreamWeaver, Sublime

Pour écrire le code

Serveur WEB

APACHE, NGinx

Pour simuler un milieu d'hebergement et interpréter le requetes.

Interpréteur PHP

Php.exe

L'executeur des scripts php

SGBD

VsCode, DreamWeaver, Sublime

Le system qui gere votre BD

La Solution intégrée

WAMP

Pour éviter une installation manuelle et complexe de chaque composant, nous utilisons un paquetage intégré. **XAMPP** est la solution la plus populaire.

- **Apache** (le serveur Web)
- **MariaDB** (un remplaçant de MySQL pour la base de données)
- **PHP** (l'interpréteur de langage)
- **Perl** (un autre langage de script)

*Inclut également **phpMyAdmin**, une interface web pour gérer facilement vos bases de données.*

La Solution intégrée

XAMPP

Pour éviter une installation manuelle et complexe de chaque composant, nous utilisons un paquetage intégré. **XAMPP** est la solution la plus populaire.

- **Apache** (le serveur Web)
- **MariaDB** (un remplaçant de MySQL pour la base de données)
- **PHP** (l'interpréteur de langage)
- **Perl** (un autre langage de script)

*Inclut également **phpMyAdmin**, une interface web pour gérer facilement vos bases de données.*

Exécution d'un fichier *.php*

WAMP

un fichier ***php*** doit être place dans le dossier racine de votre serveur web local pour être accessible via un navigateur.
(souvent ***www***, ***htdocs*** ou ***wwwroot*** selon votre installation comme ***WAMP***, ***XAMPP*** , ***MAMP***)

Emplacement :
C:\wamp\www*mon_fichier.php*

Accès :
Pour l'exécuter, vous devrez alors taper son URL dans votre navigateur,
par exemple :
http://localhost/mon_fichier.php ou
http://127.0.0.1/mon_fichier.php

Syntaxe *php*

```php
<?php // Début du bloc de code Php


// Les variables commence toujours par $
$message = "Bonjour";
// chaque instruction se termine par un point-virgule;
echo $message . « <br> »;
// Pomme echo est une commande utilisée pour afficher
// du texte, des variables ou du HTML vers le
navigateur.


?> // Fin du bloc
```

Règles d'or:

1. ***Encapsulation***:
   le code php est encapsule dans les balises **<?php… ?>**
2. ***Variables***:
   les variables sont déclarées avec un signe dollar **$**
3. ***Terminaisons***:
   chaque instruction se termine par un point-virgule **;**
4. ***Sensibilité a la casse***:
   PHP est sensibles a la case pour les variables ($nom *et* $NOM *sont différents)*

Types de Données fondamentaux

Nombre

123
3.14

- Nombres entiers, positifs ou negatives
- Reels avec virgule

```
$annee = 2026;
```

```
$moyenne = 14.67;
```

Chaine de caracteres

“ ”

Suite de caractères utilisée pour stocker du texte

```
$titre = “Mo premier projet”;
```

Boolean

true
false

Représente une valeur de vérité :

- **vrai** (true),
  faux (false)

```
$active = false;
```

Tableau

[ ]

Liste d'éléments où chaque élément est associé à un index (commençant par 0)

```
$couleur = array(
“blanc” , “jaune” ,
”rouge” );
```

Nombres

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

- **int** (entiers)
  float (nombres à virgule flottante)

Les opérateurs arithmetiques

```
+
-
*
/
%
```

*en PHP lorsqu'on compare un nombre (**int**) à une chaîne de caractères numérique (**string**) avec un opérateur de comparaison <, il convertit automatiquement la chaîne en nombre avant d'effectuer le calcul:*

- *L'opération devient donc : 5 < 6.*
- *Puisque 5 est supérieur à 6, le résultat est **true***

Les opérateurs de comparaison

```
==
<>
>
>=
<
<=
```

Fonction	Description	Exemple
`abs(x)`	Retourne la valeur absolue d'un nombre.	`abs(-15)` → `15`
`sqrt(x)`	Calcule la racine carrée d'un nombre.	`sqrt(16)` → `4`
`rand(min , max)`	Génère un nombre entier aléatoire entre deux bornes.	`rand(1,100)` → `56`
`round(x , precision)`	Arrondit un nombre à l'entier le plus proche ou à une précision définie.	`round(3.14159 , 2)` → `3.14`

Chaines de caractères

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.

```php
<?php

$prenom = "Amine";

$nom = "Tounsi";

$nom_complet = $prenom . " - " . $nom;

echo $nom_complet; // Affiche : Amine - Tounsi

?>
```

Chaines de caractères

Fonction	Description	Exemple
`chr(nbre)`	*retourne le caractère correspondant à un code ascii.*	`chr(65) → a`
`ord($c)`	*retourne le code ascii du premier caractère.*	`ord("a") → 65`
`strlen($chaine)`	*retourne la longueur d'une chaîne.*	`strlen("hello") → 5`
`substr($chaine , pos, nbr)`	*retourne une sous-chaîne d'une chaîne.*	`substr("hello", 1, 3) → ell`
`strpos($chaine , $c)`	*retourne la position de la 1ere occurrence d'une sous-chaîne.*	`strpos("hello", "e") → 1`
`strcmp($chaine1 , $chaine2)`	*Compare deux chaînes.*	`strcmp("apple", "banana") → -1`
`str_replace($chaine)`	*Remplace une sous-chaîne par une autre.*	`str_replace("World", "PHP", "Hello, World!") → Hello, PHP!`
`strtolower($chaine)`	*Convertit une chaîne en minuscules.*	`strtolower("HELLO") → hello`
`strtoupper($chaine)`	*Convertit une chaîne en majuscules.*	`strtoupper("hello") → HELLO`

Date/Heure

Fonction	Description	Exemple
`checkdate(mois, jour, année)`	Vérifie si une date est valide. (true/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`

Transtypage

Opérateur	Description	Exemple
`(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"]`

Booleen

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

==
<>
>
>=
<
<=

*en PHP lorsqu'on compare un nombre (**int**) à une chaîne de caractères numérique (**string**) avec un opérateur de comparaison <, il convertit automatiquement la chaîne en nombre avant d'effectuer le calcul:*

- *L'opération devient donc : 5 < 6.*
- *Puisque 5 est supérieur à 6, le résultat est **true***

Les opérateurs logiques

&&

||

!

```
(5 > 3) && (8 < 10) → true

5 == 2 || 4 > 3 → true

!(5 > 3) → flase
```

Tableau

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**.

// Déclarations

indexée

```php
$fruits = array("Pomme", "Banana", "Orange");
```

```php
$fruits = ["Pomme", "Banane", "Orange" ];
```

associatif

```php
$personne = array(
    "nom" => "Ali",
    "age" => 25,
    "ville" => "Tunis"
);
```

```php
$personne = [
    "nom" => "Ali",
    "age" => 25,
    "ville" => "Tunis"
];
```

// Accès aux éléments

```php
echo $fruits[1]; // affiche Banana
```

```php
$personne["nom"] = " karim ";
echo $personne["nom"]; // karim
echo $personne["ville"]; // Tunis
```

Tableau

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**.

// Déclarations

indexée

```php
$fruits = array("Pomme", "Banana", "Orange");
```

```php
$fruits = ["Pomme", "Banane", "Orange" ];
```

associatif

```php
$personne = array(
    "nom" => "Ali",
    "age" => 25,
    "ville" => "Tunis"
);
```

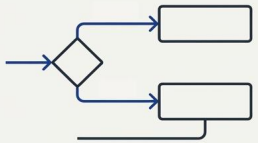
```php
$personne = [
    "nom" => "Ali",
    "age" => 25,
    "ville" => "Tunis"
];
```

// Parcours des éléments

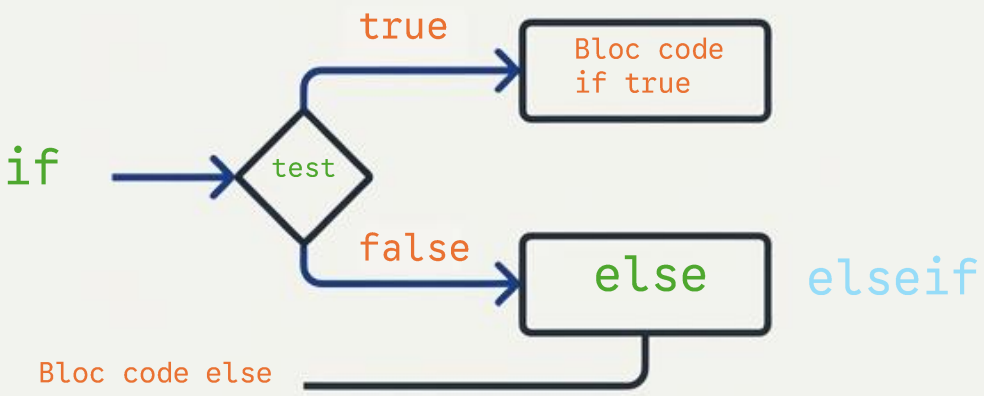
```php
for ($i = 0; $i < count($personne); $i++) {
echo $personne[$i];
}
```

Remarque
`count()` est une fonction qui retourne le nombre d'éléments d'un tableau.

```php
foreach ($t as $key => $val) {
echo "$key -> $val";
}
```



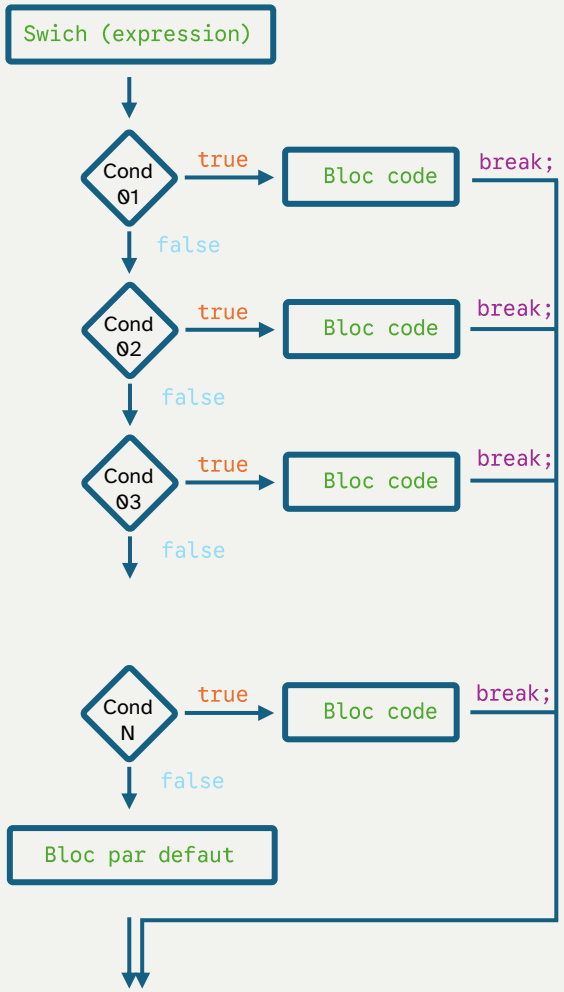
Structures de contrôles conditionnelles




```
if($x > 0){
 echo "Positif";
}
```

```
if($x > 0){
echo "Positif";
} else {
echo  "Negatif";
}
```

```
if($x > 0){
echo "Positif";
} elseif($x < 0){
echo "Négatif";
} else {
echo "Zéro";
}
```

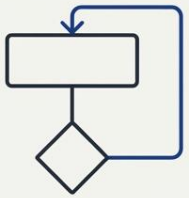



```
switch($mois){

case 1: echo "Janvier"; break;
case 2: echo "Février"; break;
case 3 : echo « Mars"; break;

default: echo "Autre mois";

}
```



Structures de contrôles iteratives

for

exécute un bloc un nombre déterminé de fois

```
for($i=0; $i<=5; $i++){
 echo $i;
}
```

while

exécute un bloc tant que la condition est vraie

```
$i=0;
while($i<=5){
 echo $i;
 $i++;
}
```

Do … while

Exécute un bloc au moins une fois puis répète tant que la condition est vraie

```
$i=0;
do {
  echo $i;
  $i++;
}
while($i<=5);
```

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.

```php
<?php

function addition($a, $b){
 return $a + $b;
}
echo addition(4, 6); // Affiche  10

?>
```

```php
function nomDeLaFonction(param1, param2, ...) {

  // Instructions à exécuter
  return valeur; // optionnel

}
```

Exemples Boucle *for*

```php
<?php

$prenoms = ["Oussema", "Salim" , "Mehdi", "Ali"];
for ($i=0; $i< count($prenoms); $i++ ){
    echo $prenoms[$i]."<br>";
}

?>
```

```php
<?php

$notes=[10,"5",13];
$moy=0;
for ($j=0; $j< count($notes); $j++ ){
    $moy += (int)$notes[$j];
}
$moy = $moy / count($notes);
echo $moy;

?>
```

Exemples Boucle *While & Do While*

```php
<?php
$i = 1; // Initialisation du compteur


while ($i <= 5) { // Condition : tant que $i est inférieur ou égal à 5
    echo $i; // Affiche la valeur de $i
    $i++;    // Incrémentation: augmente $i de 1 à chaque tour de boucle
}
// Sortie : 12345
?>
```

```php
<?php
$i = 1; // Initialisation du compteur

do {
    echo $i; // Affiche la valeur de $i
    $i++;    // Incrémentation
} while ($i <= 5); // Condition : répète tant que $i est inférieur ou égal à 5
// Sortie : 12345
?>
```

Exemples Boucle *While & Do While*

```php
<?php
$i = 10; // La condition ($i < 5) sera fausse dès le début
echo "--- Exemple While ---" . "<br>";
while ($i < 5) {
    echo $i; // Ce code ne sera jamais exécuté
    $i++;
}
echo "Fin de la boucle while" . "<br>";


$i = 10; // Réinitialisation de $i
echo "--- Exemple Do-While ---" . "<br>";
do {
    echo $i; // Ce code s'exécutera une fois (affiche 10)
    $i++;
} while ($i < 5); // La condition est testée ici et est fausse, donc la boucle s'arrête
echo "Fin de la boucle do-while" . "<br>";

?>
```

Affichage

```
// --- Exemple While ---
// Fin de la boucle while
// --- Exemple Do-While ---
// 10Fin de la boucle do-while
```

Créer un tableau *indexé* a partir d'un un tableau HTML

Voici un exemple simple en PHP utilisant une boucle for pour transformer un tableau indexé en un tableau HTML, en itérant sur chaque élément pour créer une ligne (<tr>) contenant une cellule de données (<td>), idéal pour afficher des listes d'éléments comme des prénoms ou des produits.

```php
<?php
// 1. Définir un tableau indexé
$fruits =
array("Pomme", "Banane", "Orange", "Mangue");

// 2. Début du tableau HTML
echo "<table border='1' cellpadding='5'>";
echo "<tr><th>Fruits Disponibles</th></tr>";
// Optionnel: un en-tête

// 3. Boucle 'for' pour parcourir le tableau
for ($i = 0; $i < count($fruits); $i++) {
    // Afficher le début d'une ligne
    echo "<tr>";
    // Afficher la cellule avec la valeur du tableau
    echo "<td>" . $fruits[$i] . "</td>";
```

```php
// Afficher la fin de la ligne
    echo "</tr>";
}

// 4. Fin du tableau HTML
echo "</table>";
?>
```

Explication du code :

1. **`$fruits = array(...)`** : Crée un tableau avec des chaînes de caractères, chaque élément étant indexé numériquement (0, 1, 2, 3).
2. **`echo "<table border='1'>"`** : Commence la structure HTML du tableau, border='1' pour voir les bordures.
3. **`for ($i = 0; $i < count($fruits); $i++)`** :
   - *$i = 0 : Initialise un compteur i à 0.*
   - *$i < count($fruits) : La boucle continue tant que i est inférieur au nombre d'éléments dans le tableau (ici, 4).*
   - *$i++ : Incrémente i de 1 à chaque tour de boucle.*
4. **`echo "<td>" . $fruits[$i] . "</td>";`** : Affiche une cellule <td> contenant la valeur $fruits[$i] (ex: "Pomme", puis "Banane", etc.).
5. **`echo "</table>"`** : Ferme le tableau HTML à la fin de la boucle.

Ce script générera le code HTML suivant :

```html
<table border='1' cellpadding='5'>
  <tr><th>Fruits Disponibles</th></tr>
  <tr><td>Pomme</td></tr>
  <tr><td>Banane</td></tr>
  <tr><td>Orange</td></tr>
  <tr><td>Mangue</td></tr>
</table>
```

Fruits Disponibles
Pomme
Banane
Orange
Mangue

Fonctions diverses

`require()` Permet d'inclure et d'exécuter un autre fichier PHP.
C'est l'équivalent d'importer des modules ou des plans pré-dessinés (en-têtes, pieds de page, fonctions) dans votre projet principal pour une meilleure organisation et réutilisabilité.

```php
<?php

require("config.php");
echo "Fichier inclus avec succès";

?>
```

`isset()` permet de vérifier si une variable existe et n'est pas nulle. Elle retourne :

- **`true`** si la variable existe
- **`false`** sinon.

```php
if(isset($age)){
 echo "La variable existe et vaut $age";

} else {
 echo "La variable age n'existe pas";
}
```

`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.

```php
$nom = "";

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

Fonctions diverses *exemples*

`isset()` : vérifie si une variable existe et n'est pas NULL, par exemple pour traiter des données d'un formulaire :

on utilise souvent **`isset($_POST['champ_formulaire'])`** pour voir si le champ a été rempli avant de l'utiliser,

- retournant **TRUE** si défini *(même s'il est vide)*
- **FALSE** si non déclaré ou **NULL**

```php
// Exemple 1: Variable définie
$nom_utilisateur = "Alice";

if (isset($nom_utilisateur)) {
    echo "Bonjour, $nom_utilisateur !"; //
Affiche "Bonjour, Alice !"
} else {
    echo "Utilisateur non défini.";
}

echo "<br>"; // Saut de ligne
```

```php
// Exemple 2: Variable non définie
// (va déclencher le 'else')
if (isset($email)) {
    echo "L'email est : $email";
} else {
    echo "L'email n'est pas défini."; //
Affiche "L'email n'est pas défini."
}

echo "<br>";
```

```php
// Exemple 3: Variable définie mais avec une valeur NULL
// (traité comme non défini)
$age = NULL;

if (isset($age)) {
    echo "L'âge est : $age";
} else {
    echo "L'âge n'est pas défini (ou est NULL).";
// Affiche "L'âge n'est pas défini (ou est NULL)."
}

echo "<br>";
```

```php
// Exemple 4: Avec les données d'un formulaire (POST)
// Si un champ 'nom' d'un formulaire HTML est soumis
if (isset($_POST['nom'])) {
    echo "Le nom soumis est : " . $_POST['nom'];
} else {
    echo "Veuillez soumettre le formulaire.";
}
?>
```

Principe de transmission de données entre des pages web

Dans une application web, ***php*** transmet les informations d'une page à une autre, via des méthodes **HTTP**

GET

POST

puis ***php*** les rend accessibles via des variables superglobales comme

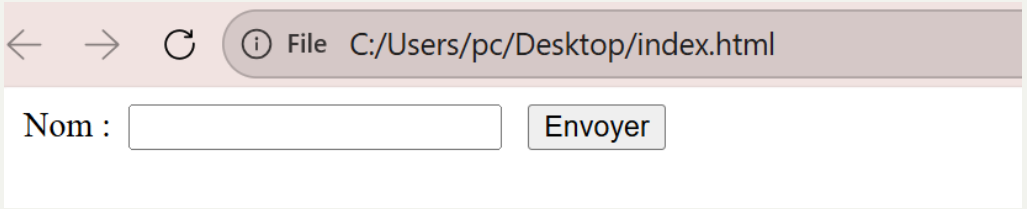
$_POST

$_GET

Principe de transmission de données entre des pages web

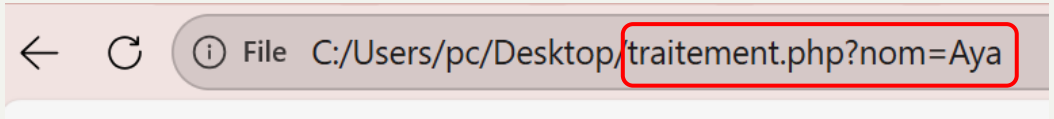
Méthode GET

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




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

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



traitement.*php*?nom=Aya

$_GET

Puis a travers la variables superglobales ***$_GET***, on peut récupérer la valeur de la variable nom

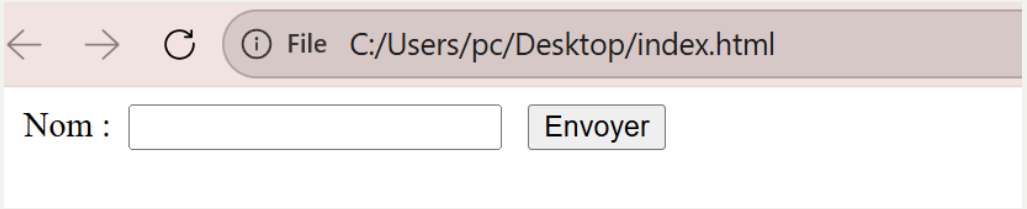
Tableau associatif des valeurs passées au script courant via les paramètres ***URL***

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

Principe de transmission de données entre des pages web

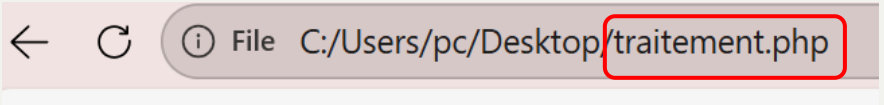
Méthode POST

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...).




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

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



***traitement**.php*

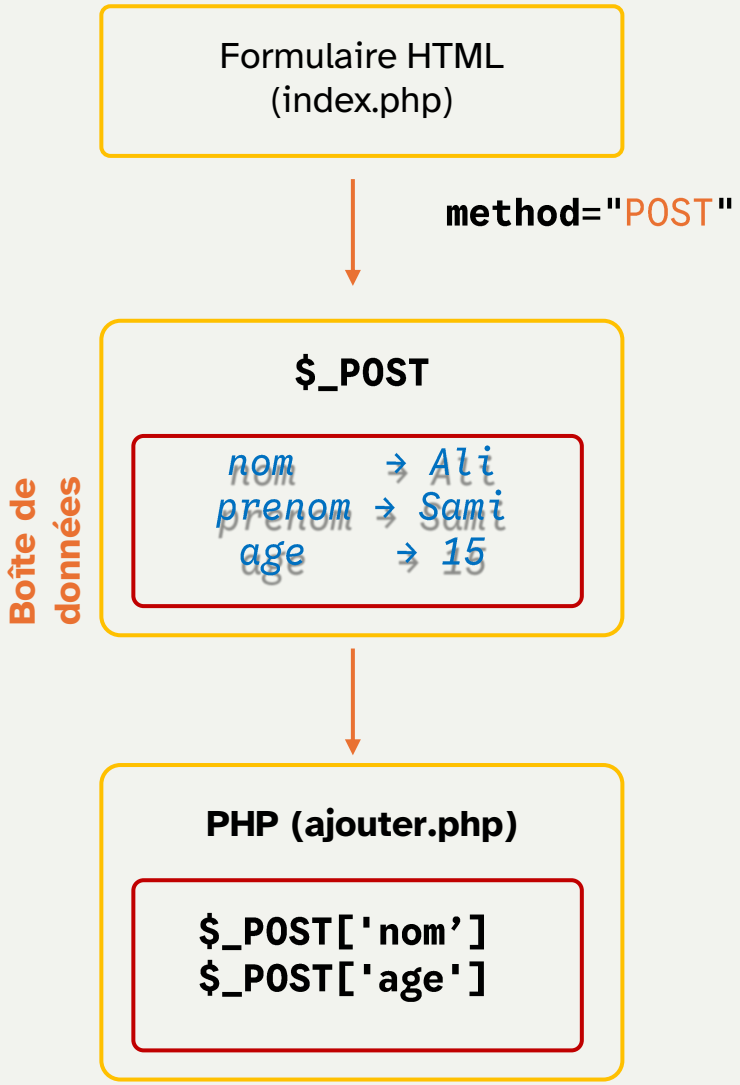
$_POST

Puis a travers la variables superglobales ***$_POST***, on peut récupérer la valeur de la variable nom

Tableau associatif des valeurs passées au script courant via le protocole ***HTTP*** et la méthode ***post***

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

Méthode POST



🧠 Explication avec des mots simples

1. L'utilisateur remplit le formulaire
2. Il clique sur **Envoyer**
3. Les données passent dans **$_POST**
4. PHP lit les données et les utilise

🎒 Exemple

👉 Si l'utilisateur écrit :

- Nom : **Ali**
- Prénom : **Sami**
- Âge : **15**

👉 Alors en PHP :

```
• $_POST['nom']     // Ali
• $_POST['prenom'] // Sami
• $_POST['age']     // 15
```

🎯 Phrase clé à retenir

$_POST est comme un sac qui transporte les données du formulaire vers PHP.

✅ Astuce

Si le ***name*** n'existe pas dans le formulaire, alors **$_POST** ne peut rien récupérer.

Principe de transmission de données entre des pages web

Connexion a la BD

Établir une connexion sécurisée avec le serveur de base de données. C'est la poignée de main initiale.

Requête SQL

Formuler et envoyer une requête SQL (SELECT, INSERT, UPDATE...) pour demander ou modifier des données.

Exploitation du résultat

Traiter la réponse.
Pour une sélection, cela signifie parcourir les résultats ligne par ligne.

Fermeture de la connexion

Fermer proprement la connexion pour libérer les ressources du serveur,

Principe de transmission de données entre des pages web

Connexion a la BD

Requête SQL

Exploitation du résultat

Fermeture de la connexion

```
$cnx = mysqli_connect('localhost', 'root', '', 'nomBaseDeDonnee')
```

Or `die(«connexion échouée»)`

Principe de transmission de données entre des pages web

Connexion
a la BD

**Requête
SQL**

Exploitation
du résultat

Fermeture
de la connexion

```
$req = « SELECT * FROM USER » ;
$res = mysqli_query($cnx, $req)
Or die(mysqli_error($cnx)) ;
```

Principe de transmission de données entre des pages web

Connexion
a la BD

Requête
SQL

Exploitation
du résultat

Fermeture
de la connexion

$res

champ1	Champ2	Champ3	champ4

$row

$nb = *nombres des lignes de l'objet $res*

```
$row = mysqli_fetch_array($res); echo $row['nom'];
$row = mysqli_fetch_rows($res);echo $row[1];
$nb =  mysqli_num_rows($res);
$x =  mysqli_affected_rows($cnx);
```

Principe de transmission de données entre des pages web

Connexion
a la BD

Requête
SQL

Exploitation
du résultat

**Fermeture
de la connexion**

```
mysqli_close($cnx);
```

Principe de transmission de données entre des pages web *mysqli*

Fonction	Description	Exemples
`mysqli_connect(host, user, password, db)`	*Etablie une connexion a une BD MySQL*	`$cnx = mysqli_connect('localhost', 'root', '', 'db03')` `Or die(«connexion échouée»)`

`mysqli_query(cnx, req)`	*Envoie une requête SQL a la BD*	`$req = «SELECT * FROM USER; » ;`
		`$res = mysqli_query($cnx, $req)`
		`Or die(mysqli_error($cnx)) ;`

`mysqli_error(cnx)`	*Retourne un message d'erreur générer par dernière requête SQL exécutéeFerme*	`echo mysqli_error($cnx);`
`mysqli_fetch_array(res)`	*Retourne une ligne de résultat sous forme de tableau associatif et numérique*	`$row = mysqli_fetch_array($res);` `echo $row['nom'];`

Principe de transmission de données entre des pages web

Fonction	Description	Exemples
`mysqli_fetch_rows(res)`	*Retourne une ligne de résultat sous forme de tableau indexe numériquement*	`$row = mysqli_fetch_rows($res);` `echo $row[1];`
`mysqli_num_rows(res)`	*Retourne le nombre de lignes dans un résultat de requête*	`echo mysqli_num_rows($res);`
`mysqli_affected_rows(cnx)`	*Retourne le nombre des lignes affecte par la dernière requête SQL exécutée.*	`echo mysqli_affected_rows($cnx);`
`mysqli_close(cnx)`	*Ferme la connexion a la BD*	`mysqli_close($cnx);`

Principe de transmission de données entre des pages web

Fonction	Description	Exemples
`mysqli_connect(host, user, password, db)`	*Etablie une connexion a une BD MySQL*	`$cnx = mysqli_connect('localhost', 'root', '', 'db03')` `Or die(«connexion échouée»)`

`mysqli_query(cnx, $sql)`	*Envoie une requête SQL a la BD*	`$req = «SELECT * FROM USER; » ;`
		`$res = mysqli_query($cnx, $req)`
		`Or die(mysqli_error($cnx)) ;`

`mysqli_error(cnx)`	*Retourne un message d'erreur générer par dernière requête SQL exécutée*	`echo mysqli_error($cnx);`
`mysqli_fetch_array(res)`	*Retourne une ligne de résultat sous forme de tableau associatif et numérique*	`$row = mysqli_fetch_array($res);` `echo $row['nom'];`
`mysqli_fetch_rows(res)`	*Retourne une ligne de résultat sous forme de tableau indexe numériquement*	`$row = mysqli_fetch_rows($res);` `echo` $row[1];
`mysqli_num_rows(cnx)`	*Retourne le nombre de lignes dans un résultat de requête*	`echo mysqli_num_rows($res);`
`mysqli_affected_rows(cnx)`	*Retourne le nombre des lignes affecte par la dernière requête SQL exécutée.*	`echo mysqli_affected_rows($cnx);`
`mysqli_close(cnx)`	*Ferme la connexion a la BD*	`mysqli_close($cnx);`