**Module 1 : Introduction à PHP**

1. **Qu'est-ce que PHP ?**
   * Définition et historique
   * Fonctionnalités et usages
2. **Installation et configuration**
   * Installation de PHP (XAMPP, WAMP, LAMP, LARAGON)
   * Configuration d'un serveur local
   * Configuration du fichier php.ini
3. **Premier script PHP**
   * Syntaxe de base
   * Écrire et exécuter un script PHP
   * Utiliser la balise PHP <?php ?>

**Module 2 : Les bases du langage PHP**

1. **Variables et types de données**
   * Types de données : chaîne, entier, flottant, booléen, etc.
   * Déclaration de variables
   * Concaténation et interpolation de chaînes
2. **Opérateurs**
   * Opérateurs arithmétiques, logiques, et de comparaison
3. **Structures de contrôle**
   * Conditions : if, else, elseif, switch
   * Boucles : for, while, do-while, foreach
4. **Fonctions**
   * Définir une fonction
   * Paramètres et valeurs de retour
   * Portée des variables (globales, locales, statiques)

**Module 3 : Travaux avec les chaînes et les tableaux**

1. **Chaînes de caractères**
   * Fonctions de manipulation des chaînes : strlen(), strpos(), substr(), etc.
2. **Tableaux**
   * Tableaux indexés et associatifs
   * Fonctions pour les tableaux : array\_push(), array\_merge(), array\_map()
   * Boucle sur les tableaux

**Module 4 : Programmation orientée objet (POO)**

1. **Concepts de base de la POO**
   * Classes et objets
   * Propriétés et méthodes
2. **Visibilité**
   * Public, privé, protégé
3. **Héritage**
   * Classes parentes et enfants
   * Redéfinition des méthodes
4. **Polymorphisme**
   * Méthodes abstraites et interfaces
5. **Gestion des exceptions**
   * try, catch, throw

**Module 5 : Manipulation des formulaires HTML avec PHP**

1. **Formulaires HTML**
   * Méthodes GET et POST
   * Validation des entrées utilisateur
2. **Sécurité des formulaires**
   * Protéger contre les attaques XSS (Cross-Site Scripting)
   * Protéger contre les attaques CSRF (Cross-Site Request Forgery)

**Module 6 : Interaction avec les bases de données**

1. **Connexion à une base de données MySQL avec PDO**
   * Créer et configurer une base de données
   * Exécuter des requêtes SQL avec PDO
2. **Opérations CRUD**
   * Insérer des données (Create)
   * Lire des données (Read)
   * Mettre à jour des données (Update)
   * Supprimer des données (Delete)
3. **Sécuriser les requêtes SQL**
   * Utiliser des requêtes préparées
   * Éviter les injections SQL

**Module 7 : Gestion des sessions et cookies**

1. **Sessions**
   * Créer et gérer une session utilisateur
   * Détruire une session
2. **Cookies**
   * Créer et lire des cookies
   * Sécuriser les cookies

### ****Module 1 : Introduction à PHP (Résumé)****

#### ****1. Qu'est-ce que PHP ?****

* **Définition et historique** :
  + PHP (Hypertext Preprocessor) est un langage de script côté serveur créé en 1994 par Rasmus Lerdorf, utilisé pour créer des pages web dynamiques.
* **Fonctionnalités et usages** :
  + Générer des pages HTML dynamiques.
  + Gérer les sessions utilisateurs.
  + Interagir avec des bases de données (MySQL, PostgreSQL).
  + Fonctionne sur de nombreux systèmes d'exploitation et serveurs web.

#### ****2. Installation et configuration****

* **Installation de PHP avec Laragon** :
  + Téléchargez Laragon depuis <https://laragon.org>, installez-le et démarrez-le.
  + Tous vos fichiers PHP doivent être placés dans le dossier **C:/laragon/www/**.
* **Configuration d'un serveur local** :
  + Démarrez Laragon et accédez à votre serveur local en tapant **http://localhost** dans le navigateur.
* **Configuration du fichier php.ini** :
  + Dans Laragon, allez dans **Menu > PHP > php.ini** pour modifier des paramètres comme l'affichage des erreurs (display\_errors = On) ou le fuseau horaire (date.timezone = "Europe/Paris").

#### ****3. Premier script PHP****

* **Syntaxe de base** :
  + PHP s'écrit entre les balises <?php et ?>.

**Exemple** :

<?php

  echo "Bonjour depuis PHP!";

?>

 **Écrire et exécuter un script PHP** :

* Placez votre fichier PHP (ex. **index.php**) dans **C:/laragon/www/**.
* Accédez à **http://localhost/index.php** pour voir le résultat.

 **Utilisation de la balise PHP <?php ?>** :

* PHP s'intègre dans les balises HTML pour générer du contenu dynamique.

**Exemple** :

<p>Date actuelle : <?php echo date('Y-m-d H:i:s'); ?></p>

### ****Module 2 : Les bases du langage PHP (Résumé)****

#### ****1. Variables et types de données****

* **Types de données** :
  + **Chaîne** (string) : texte, ex. "Bonjour".
  + **Entier** (int) : nombre entier, ex. 42.
  + **Flottant** (float) : nombre à virgule, ex. 3.14.
  + **Booléen** (bool) : vrai ou faux, ex. true ou false.
* **Déclaration de variables** :
  + Les variables commencent par $ suivi du nom. Elles n'ont pas besoin d'être déclarées avec un type.
* $nom = "Dara";
* $age = 25;

**Concaténation et interpolation de chaînes** :

* **Concaténation** : Utilisation du symbole . pour joindre des chaînes.
* $message = "Bonjour " . $nom;

**Interpolation** : Insertion directe de variables dans une chaîne avec des guillemets doubles.

echo "Bonjour $nom";

#### ****2. Opérateurs****

* **Opérateurs arithmétiques** : Addition (+), soustraction (-), multiplication (\*), division (/), modulo (%).

$total = 5 + 3;  // Résultat : 8

* **Opérateurs logiques** : && (ET), || (OU), ! (NON).

$result = ($age > 18 && $status == "actif");

* **Opérateurs de comparaison** : == (égalité), != (différent), > (supérieur), < (inférieur).

if ($age >= 18) { echo "Majeur"; }

#### ****3. Structures de contrôle****

* **Conditions** : if, else, elseif, switch.

if ($age >= 18) {

  echo "Majeur";

} else if ($age < 18) {

  echo "Mineur";

} else {

  echo "Erreur";

}

* **Switch** (alternative aux multiples if):

switch ($jour) {

  case 'Lundi':

      echo "Début de la semaine";

      break;

  case 'Vendredi':

      echo "Fin de semaine";

      break;

  default:

      echo "Jour classique";

}

* **Boucles** : for, while, do-while, foreach.

// Boucle for

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

  echo "Nombre : $i";

}

// Boucle foreach pour parcourir un tableau

$jours = ["Lundi", "Mardi", "Mercredi"];

foreach ($jours as $jour) {

  echo $jour;

}

#### ****4. Fonctions****

* **Définir une fonction** :

function direBonjour() {

  echo "Bonjour!";

}

**Paramètres et valeurs de retour** :

* Les fonctions peuvent accepter des paramètres et retourner des valeurs.

function additionner($a, $b) {

  return $a + $b;

}

echo additionner(5, 3);  // Résultat : 8

**Portée des variables** :

* **Locale** : Les variables définies dans une fonction sont locales et ne sont pas accessibles en dehors.

function demo() {

  $local = "Locale";

}

**Globale** : Les variables définies en dehors d'une fonction doivent être marquées comme **globales** pour être accessibles à l'intérieur de la fonction.

$global = "Global";

function demo() {

    global $global;

    echo $global;

}

**Statique** : Les variables déclarées comme **statiques** dans une fonction conservent leur valeur entre les appels de la fonction.

function compteur() {

  static $count = 0;

  $count++;

  echo $count;

}

Ce module présente les bases essentielles du langage PHP, allant des types de données aux structures de contrôle, ainsi que la création et l'utilisation de fonctions.

**Les différentes formes de fonctions utilitaires en PHP**

PHP propose de nombreuses façons de définir et d’utiliser des fonctions avec différents types de paramètres, de valeurs de retour et de signatures typées. Voici un tour d’horizon des concepts essentiels liés aux fonctions en PHP.

#### ****1. Paramètres par défaut****

Les paramètres par défaut permettent de définir une valeur par défaut si un argument n'est pas fourni lors de l'appel de la fonction.

function saluer($nom = "Invité") {

  echo "Bonjour, $nom!";

}

saluer(); // Affiche : Bonjour, Invité!

saluer("Dara"); // Affiche : Bonjour, Dara!

#### ****2. Paramètres typés****

PHP permet de définir le type de données attendu pour les arguments. Cela garantit que les arguments fournis à la fonction respectent un type spécifique (string, int, array, etc.).

function additionner(int $a, int $b): int {

  return $a + $b;

}

echo additionner(5, 3); // Affiche : 8

// echo additionner(5, "3"); // Provoquerait une erreur si strict\_types est activé.

Le retour de la fonction peut également être typé avec : type après les parenthèses.

#### ****3. Paramètres de type**** null

On peut définir un paramètre comme étant facultatif avec un type nullable en ajoutant ? avant le type.

function afficherNom(?string $nom) {

  if ($nom) {

      echo "Bonjour, $nom!";

  } else {

      echo "Aucun nom fourni";

  }

}

afficherNom(null); // Affiche : Aucun nom fourni

afficherNom("Dara"); // Affiche : Bonjour, Dara!

#### ****4. Paramètres variables (Rest Parameters)****

Les fonctions PHP peuvent accepter un nombre variable d'arguments grâce à l'opérateur .... Ceci est utile pour des cas comme la somme de plusieurs nombres ou la concaténation de plusieurs chaînes.

function somme(...$nombres) {

  return array\_sum($nombres);

}

echo somme(1, 2, 3, 4); // Affiche : 10

* Tous les arguments sont capturés dans un tableau.

#### ****5. Paramètres nommés (à partir de PHP 8.0)****

Les **paramètres nommés** permettent de passer des arguments à une fonction en spécifiant leur nom, ce qui permet de les passer dans n'importe quel ordre et de laisser de côté les paramètres avec valeurs par défaut.

function creerUtilisateur($nom, $age, $role = "utilisateur") {

  echo "Nom : $nom, Âge : $age, Rôle : $role";

}

// En utilisant des paramètres nommés

creerUtilisateur(age: 25, nom: "Dara");

// Affiche : Nom : Dara, Âge : 25, Rôle : utilisateur

#### ****6. Fonction avec référence (&)****

Les paramètres peuvent être passés par référence à l’aide de &. Cela signifie que toute modification apportée à l’intérieur de la fonction affectera également la variable d’origine.

function ajouterValeur(&$nombre) {

  $nombre += 10;

}

$valeur = 5;

ajouterValeur($valeur);

echo $valeur; // Affiche : 15

#### ****7. Fonctions anonymes (Closures)****

Les **fonctions anonymes** ou **closures** sont des fonctions sans nom qui peuvent être assignées à des variables ou passées en tant qu'arguments. Elles peuvent aussi capturer des variables externes via use.

$saluer = function($nom) {

  return "Bonjour, $nom!";

};

echo $saluer("Dara"); // Affiche : Bonjour, Dara!

// Capturer des variables externes

$prefixe = "Bonjour";

$salutation = function($nom) use ($prefixe) {

  return "$prefixe, $nom!";

};

echo $salutation("Dara"); // Affiche : Bonjour, Dara!

#### ****8. Fonctions fléchées (Arrow Functions)****

Introduites en PHP 7.4, les **fonctions fléchées** sont une syntaxe concise pour les fonctions anonymes. Elles capturent automatiquement les variables externes.

$multiplier = fn($a, $b) => $a \* $b;

echo $multiplier(3, 4); // Affiche : 12

À la différence des closures, les **arrow functions** capturent les variables externes **par valeur** et non par référence.

#### ****9. Fonction avec retour typé****

Vous pouvez spécifier le type de la valeur de retour d'une fonction avec : type.

function obtenirNom(): string {

  return "Dara";

}

echo obtenirNom(); // Affiche : Dara

Si la fonction doit retourner plusieurs types, vous pouvez utiliser l'union de types (à partir de PHP 8.0) :

function obtenirValeur(bool $estEntier): int|string {

  return $estEntier ? 42 : "Quarante-deux";

}

echo obtenirValeur(true); // Affiche : 42

#### ****10. Valeurs de retour multiples avec tableau****

PHP ne prend pas en charge directement les valeurs de retour multiples, mais vous pouvez retourner un tableau ou un objet.

function obtenirInfos(): array {

  return ['nom' => "Dara", 'age' => 25];

}

$infos = obtenirInfos();

echo $infos['nom']; // Affiche : Dara

#### ****Résumé****

En PHP, les fonctions offrent une grande flexibilité grâce à divers concepts :

1. Paramètres par défaut pour des valeurs optionnelles.
2. Typage des paramètres et du retour pour garantir des types spécifiques.
3. Paramètres variables avec ... pour un nombre d’arguments flexible.
4. Passage par référence pour modifier les variables externes.
5. Fonctions anonymes et fléchées pour des callbacks ou logiques simples.
6. Paramètres nommés pour plus de clarté et de flexibilité à l’appel.

Ces concepts permettent de structurer et d’optimiser les fonctions pour différents cas d’usage, tout en rendant le code plus lisible et maintenable.

**Module 3 : Travaux avec les chaînes et les tableaux**

### 1. Chaînes de caractères

#### a. Fonctions de manipulation des chaînes

PHP offre de nombreuses fonctions pour manipuler les chaînes de caractères. Voici quelques-unes des plus courantes :

* **strlen()** : Renvoie la longueur d'une chaîne.

<?php

 $texte = "Bonjour, Dara Sow!";

 echo "La longueur de la chaîne est : " . strlen($texte);

 // Résultat : La longueur de la chaîne est : 17

?>

* **strpos()** : Trouve la position d'une sous-chaîne dans une chaîne. Retourne false si la sous-chaîne n'est pas trouvée.

<?php

$texte = "Bonjour, Dara Sow!";

$position = strpos($texte, "Dara");

if ($position !== false) {

    echo "Le mot 'Dara' commence à la position : " . $position;

} else {

    echo "Mot non trouvé.";

}

// Résultat : Le mot 'Dara' commence à la position : 9

?>

* **substr()** : Extrait une portion d'une chaîne.

<?php

$texte = "Bonjour, Dara Sow!";

$sous\_chaine = substr($texte, 9, 4);

// Commence à l'indice 9, extrait 4 caractères

echo $sous\_chaine;

// Résultat : Dara

?>

* **str\_replace()** : Remplace une sous-chaîne par une autre dans une chaîne.

<?php

$texte = "Bonjour, Dara Sow!";

$nouveau\_texte = str\_replace("Bonjour", "Salut", $texte);

echo $nouveau\_texte;

// Résultat : Salut, Dara Sow!

?>

### 2. Tableaux

#### a. Tableaux indexés et associatifs

* **Tableaux indexés** : Les valeurs sont stockées avec des indices numériques.

<?php

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

echo $fruits[1]; // Affiche "Banane"

?>

* **Tableaux associatifs** : Les clés sont des chaînes et non des indices numériques.

<?php

$ages = ["Dara" => 30, "Sow" => 25];

echo "L'âge de Dara est : " . $ages["Dara"];

// Affiche "L'âge de Dara est : 30"

?>

#### b. Fonctions pour les tableaux

* **array\_push()** : Ajoute des éléments à la fin d'un tableau.

<?php

$fruits = ["Pomme", "Banane"];

array\_push($fruits, "Orange");

print\_r($fruits);

// Résultat : Array ( [0] => Pomme [1] => Banane [2] => Orange )

?>

* **array\_merge()** : Fusionne plusieurs tableaux.

<?php

$fruits1 = ["Pomme", "Banane"];

$fruits2 = ["Orange", "Mangue"];

$tous\_fruits = array\_merge($fruits1, $fruits2);

print\_r($tous\_fruits);

// Résultat : Array ( [0] => Pomme [1] => Banane [2] => Orange [3] => Mangue )

?>

* **array\_map()** : Applique une fonction à chaque élément d'un tableau.

<?php

function majuscule($mot) {

    return strtoupper($mot);

}

$fruits = ["pomme", "banane", "orange"];

$fruits\_maj = array\_map("majuscule", $fruits);

print\_r($fruits\_maj);

// Résultat : Array ( [0] => POMME [1] => BANANE [2] => ORANGE )

?>

#### c. Boucle sur les tableaux

* **Boucle for** pour les tableaux indexés :

<?php

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

for ($i = 0; $i < count($fruits); $i++) {

    echo $fruits[$i] . " ";

}

// Résultat : Pomme Banane Orange

?>

* **Boucle foreach** pour les tableaux associatifs :

<?php

$ages = ["Dara" => 30, "Sow" => 25];

foreach ($ages as $nom => $age) {

    echo "$nom a $age ans. ";

}

// Résultat : Dara a 30 ans. Sow a 25 ans.

?>

Cela conclut la section sur les chaînes et les tableaux dans le cadre de votre cours PHP. Vous pouvez enrichir cette partie avec plus de détails ou d'exemples si nécessaire.

**Module 4** : **Programmation orientée objet (POO)**

### 1. Concepts de base de la POO

#### a. Classes et objets

Un **constructeur** est une méthode spéciale d'une classe qui est automatiquement appelée lorsque vous créez un nouvel objet. Il permet d'initialiser les propriétés de l'objet.

* **Exemple** :

<?php

class Voiture {

    public $marque;

    public $couleur;

    // Constructeur pour initialiser les propriétés

    public function \_\_construct($marque, $couleur) {

        $this->marque = $marque;

        $this->couleur = $couleur;}

    public function demarrer() {

        return "La voiture $this->marque de couleur $this->couleur démarre.";

    }

}// Création d'un objet Voiture avec des valeurs passées au constructeur

$maVoiture = new Voiture("Toyota", "Rouge");

echo $maVoiture->demarrer();// Résultat : La voiture Toyota de couleur Rouge démarre.

?>

**2. Visibilité**

Les niveaux de visibilité (public, privé, protégé) s'appliquent aussi bien aux propriétés qu'aux méthodes, y compris le constructeur. Voici comment cela fonctionne avec un constructeur.

<?php

class Personne {

    public $nom;          // Accessible partout

    private $age;         // Accessible uniquement dans la classe

    // Constructeur public pour initialiser les propriétés

    public function \_\_construct($nom, $age) { $this->nom = $nom;

        $this->age = $age;

    }

    // Méthode publique pour afficher le nom

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

    // Méthode privée pour afficher l'âge

    private function afficherAge() {return $this->age;}

}

$personne = new Personne("Dara", 30);

echo $personne->afficherNom(); // Résultat : Dara

// echo $personne->afficherAge(); // Erreur, méthode privée

?>

Dans cet exemple, le constructeur public **\_\_construct($nom, $age)** initialise les propriétés nom et age, mais l'âge reste privé et inaccessible directement en dehors de la classe.

**3. Héritage**

L'héritage permet de réutiliser les propriétés et méthodes d'une classe parente dans une classe enfant. Lorsqu'une classe enfant hérite d'une classe parente avec un constructeur, elle peut soit utiliser ce constructeur, soit redéfinir le sien tout en appelant celui de la classe parente.

* **Exemple d'héritage**:

Classe de base :

<?php

class Animal {

    public $nom;

    // Constructeur de la classe parente

    public function \_\_construct($nom) { $this->nom = $nom; }

    public function manger() { return "$this->nom mange.";}

}

?>

**Classe enfant:**

class Chat extends Animal {

    public $couleur;

    // Constructeur de la classe enfant

    public function \_\_construct($nom, $couleur) {

        // Appel du constructeur de la classe parente

        // pour initiliser le nom

        parent::\_\_construct($nom);

        $this->couleur = $couleur;

    }

  public function miauler() {

    return "$this->nom, le chat de couleur $this->couleur, miaule.";

   }

}

$monChat = new Chat("Felix", "Blanc");

echo $monChat->manger() . '<br>';  // Résultat : Felix mange.

echo $monChat->miauler(); // Résultat : Felix, le chat de couleur Blanc, miaule.

?>

Dans cet exemple, le constructeur de la classe Chat utilise parent::\_\_construct($nom) pour appeler le constructeur de la classe parente Animal.

**4. Polymorphisme**

Le polymorphisme permet à des classes dérivées d'adopter des comportements différents tout en partageant la même interface ou classe abstraite. Un constructeur peut également être utilisé dans ce cadre pour initialiser des propriétés spécifiques à chaque classe dérivée.

**Exemple**

<?php

// Classe de base

class Animal {

    protected $nom;

    public function \_\_construct($nom) {

        $this->nom = $nom;

    }

    // Méthode redéfinie dans les classes enfants

    public function crier() {

        echo "{$this->nom} fait un bruit.\n";

    }

}

**Les classe enfants :**

// Classe dérivée : Chien

class Chien extends Animal {

    public function crier() {

        echo "{$this->nom} aboie.\n";

    }

}

// Classe dérivée : Chat

class Chat extends Animal {

    public function crier() {

        echo "{$this->nom} miaule.\n";

    }

}

**Utilisation** :

// Fonction qui accepte une référence de la classe Animal (polymorphisme)

function faireCrier(Animal $animal) {

    $animal->crier();

}

// Instanciation des objets de classes dérivées

$animal1 = new Chien("Rex");

// c'est constructeur de la class de base qui est appeler

$animal2 = new Chat("Misty");

// c'est constructeur de la class de base qui est appeler

// Ici, on utilise la classe de base comme référence

faireCrier($animal1); // Affiche : "Rex aboie."

faireCrier($animal2); // Affiche : "Misty miaule."

**NB** : si la méthode **crier()** n’existe pas dans la classe fille c’est celle de la classe parent qui est appeler , si la méthode n’existe dans aucune classe c’est une erreur qui est déclencher

La methode **faireCrier()** reçoit une instante d’Animal mais lors de son appel on envoi des intances de type **Chien** ou **Chat,** PHP accepte car un **Chien** et un **Chat** sont tous des animaux qui dérivent de la class Animal, du coup d’après la notion de polymorphisme il est tout à fait licite qu’un objet de type Animal reference un objet de type enfant, mais attention l’inverse n’es pas possible les gars il ne faut même pas essayer.

### ****5. Classes abstraites****

Une **classe abstraite** est une classe qui ne peut pas être instanciée directement. Elle sert de modèle pour les autres classes. Les classes qui héritent d'une classe abstraite doivent implémenter les méthodes abstraites définies dans celle-ci.

// Classe abstraite

abstract class Vehicule {

    protected $marque;

    public function \_\_construct($marque) {

        $this->marque = $marque;

    }

    // Méthode abstraite, doit être implémentée obligatoirement par les classes dérivées sinon c’est erreur

    abstract public function deplacer();

}

**Utilisation avec des classes enfants :**

// Classe dérivée : Voiture

class Voiture extends Vehicule {

    public function deplacer() {

        echo "La voiture de marque {$this->marque} roule sur la route.";

    }

}

// Classe dérivée : Bateau

class Bateau extends Vehicule {

    public function deplacer() {

        echo "Le bateau de marque {$this->marque} navigue sur l'eau.";

    }

}

// Test avec les classes dérivées

// le contructeur de la classe parent est automatiquement utiliser

$vehicule1 = new Voiture("Toyota");

$vehicule2 = new Bateau("Yamaha");

$vehicule3 = new Vehicule("Yamaha");//erreur Vehicule est une class abstraite

// Affiche : "La voiture de marque Toyota roule sur la route."

$vehicule1->deplacer();

// Affiche : "Le bateau de marque Yamaha navigue sur l'eau."

$vehicule2->deplacer();

### ****6. Interfaces****

Une **interface** est une classe abstraite pure c’est-à-dire elle est abstraite et toutes les methodes sont aussi abstraites, toutes les méthodes d'une interface doivent être publiques et abstraites. Une classe peut implémenter plusieurs interfaces.

<?php

// Définition d'une interface

interface Travailler {

    public function travailler();

}

// Classe qui implémente l'interface

class Ouvrier implements Travailler {

    public function travailler() {

        echo "L'ouvrier travaille dans une usine.";

    }

}

// Classe qui implémente l'interface

class Ingenieur implements Travailler {

    public function travailler() {

        echo "L'ingénieur conçoit des bâtiments.";

    }

}

// Test avec les classes qui implémentent l'interface

$ouvrier = new Ouvrier();

$ingenieur = new Ingenieur();

$ouvrier->travailler(); // Affiche : "L'ouvrier travaille dans une usine."

$ingenieur->travailler(); // Affiche : "L'ingénieur conçoit des bâtiments."

?>

**Gestion des exceptions en PHP :**

En PHP, la gestion des exceptions permet de capturer les erreurs ou des conditions anormales lors de l'exécution du programme sans que celui-ci ne s'arrête brutalement. Cela se fait avec les blocs **try**, **catch**, et **throw**.

Voici les principaux éléments à comprendre :

1. **try** : Le bloc try contient le code qui pourrait potentiellement générer une exception.
2. **catch** : Le bloc catch capture l'exception lancée par throw et permet de gérer l'erreur.
3. **throw** : Cette instruction lance une exception qui sera capturée par le bloc catch.

**Creons une exception en divisant 10 par 0**:

// 10 diviser par 0 c'est une exception,

function diviser($numerateur, $denominateur) {

    return $numerateur / $denominateur;

}

echo diviser(10, 0);

//Erreur declenchée “Fatal error: Uncaught DivisionByZeroError: Division by zero in C:\laragon\www\CoursPHP\index.php:19 Stack trace: #0 C:\laragon\www\CoursPHP\index.php(21): diviser(10, 0) #1 {main} thrown in C:\laragon\www\CoursPHP\index.php on line 19”

On pouvais bricoler en disant:

function diviser($numerateur, $denominateur) {

    if($denominateur == 0) return "Erreur le denominateur doit etre different de 0";

    return $numerateur / $denominateur;

}

echo diviser(10, 0);

Mais c’est pas professionel on devrait prevoir une classe personnaliser :

// Création d'une classe d'exception personnalisée

class DivisionParZeroException extends Exception {

    public function \_\_construct($message) {

// On passe le message d'erreur au constructeur parent (Exception)

        parent::\_\_construct($message);

    }

}

function diviser($numerateur, $denominateur) {

    if ($denominateur == 0) {

        // On lance une exception si le dénominateur est zéro

        throw new DivisionParZeroException("Division par zéro interdite.");

    }

    return $numerateur / $denominateur;

}

try {

    // Tentative de division par zéro

    echo diviser(10, 0);

} catch (DivisionParZeroException $e) {

    // Capture et traitement de l'exception

    echo "Erreur : " . $e->getMessage(); // Affiche : Erreur : Division par zéro interdite.

} finally {

    echo "\nFin du traitement."; // Ce bloc est toujours exécuté

}

### Module 5 : Manipulation des formulaires HTML avec PHP

#### 1. Formulaires HTML

##### a. Méthodes GET et POST

* **GET** : Envoie les données dans l'URL, idéal pour récupérer des informations non sensibles. Exemple :

<form method="get" action="index.php">

    <div>

        <label for="nom">Nom:</label>

        <input type="text" id="nom" name="nom">

    </div>

    <div>

        <label for="nom">Prenom:</label>

        <input type="text" id="prenom" name="prenom">

    </div>

    <button type="submit">Envoyer</button>

</form>

<!-- La logique, on est toujours dans le meme fichier index.php-->

<?php

if (isset($\_GET['nom'])) {

    echo 'Nom : ' . htmlspecialchars($\_GET['nom']) .'<br>';

}

if (isset($\_GET['prenom'])) {

    echo 'prenom : ' . htmlspecialchars($\_GET['prenom']);

}

?>

// L’url : // http://localhost/CoursPhp/index.php?nom=Sow&prenom=Mamadou

Surtout il faut toujours s’assurer que les donner envoyer ne sont pas sensible car elles sont evisible par tout le monde, d’où l’importance des framework frontend comme Angular

**POST** : Envoie les données de manière sécurisée dans le corps de la requête. Convient pour des informations sensibles. Exemple :

<form method="post" action="index.php">

    <div>

        <label for="nom">Nom:</label>

        <input type="text" id="nom" name="nom">

    </div>

    <div>

        <label for="nom">Prenom:</label>

        <input type="text" id="prenom" name="prenom">

    </div>

    <button type="submit">Envoyer</button>

</form>

<!-- La logique, on est toujours dans le meme fichier index.php-->

<?php

if (isset($\_POST['nom'])) {

    echo 'Nom : ' . htmlspecialchars($\_POST['nom']) .'<br>';

}

if (isset($\_POST['prenom'])) {

    echo 'prenom : ' . htmlspecialchars($\_POST['prenom']);

}

?>

L’avantage est que les données ne sont plus exposées

##### b. Validation des entrées utilisateur

* **Validation côté serveur** pour s'assurer que les données soumises sont correctes avant de les traiter. Exemple :

<!-- La logique, on est toujours dans le meme fichier index.php-->

<?php

if (isset($\_POST['nom']) && isset($\_POST['prenom'])) {

    // supression des espaces avant et apres

    $nom = trim($\_POST['nom']);

    $prenom = trim($\_POST['prenom']);

    if (!empty($nom) && strlen($nom) > 2) {

        echo 'Nom '. $nom . ' valide.' . '</br>';

    } else {

        echo 'Nom '. $nom . ' Invalide.' . '</br>';

    }

    if (!empty($prenom) && strlen($prenom) > 2) {

        echo 'prenom '. $prenom . ' valide.' . '</br>';

    } else {

        echo 'prenom '. $prenom . ' Invalide.' . '</br>';

    }

}

?>

// a ce stade on peut meme creer des balise dans les zones de saisie

// <strong style='color : red'>Sow</strong>

// <strong style='color : red'>Barry</strong> il va tout interpreter



Il a donc interpreter les balises, qui n’es pas bon.

* **Sanitisation** avec **htmlspecialchars()** pour éviter les injections de scripts malveillants, XSS (Cross-Site Scripting):

<!-- La logique, on est toujours dans le meme fichier index.php-->

<?php

if (isset($\_POST['nom']) && isset($\_POST['prenom'])) {

    $nom = htmlspecialchars($\_POST['nom'], ENT\_QUOTES, 'UTF-8');

    $prenom = htmlspecialchars($\_POST['prenom'], ENT\_QUOTES, 'UTF-8');

    // supression des espaces avant et apres

    $nom = trim($nom);

    $prenom = trim($prenom);

    if (!empty($nom) && strlen($nom) > 2) {

        echo 'Nom '. $nom . ' valide.' . '</br>';

    } else {

        echo 'Nom '. $nom . ' Invalide.' . '</br>';

    }

    if (!empty($prenom) && strlen($prenom) > 2) {

        echo 'prenom '. $prenom . ' valide.' . '</br>';

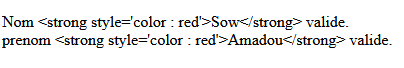
    } else {

        echo 'prenom '. $prenom . ' Invalide.' . '</br>';

    }

}

?>



Il a concerver les balise cette fois ci

##### b. Protéger contre les attaques CSRF (Cross-Site Request Forgery)

* **Utilisation d'un jeton CSRF** pour vérifier que le formulaire soumis est légitime.

Exemple de génération et de vérification d'un jeton CSRF :