PHP natif : Gestion des erreurs

1 - Introduction aux erreurs PHP

1.1 - Comprendre les erreurs

1.1.1 - Qu'est-ce qu'une erreur en PHP ?

> Une **erreur** en PHP est un problème qui **empêche l'exécution normale** du script. Elle peut être **fatale** (arrêt du script) ou **non fatale** (le script continue mais avec des avertissements).

1.1.2 - Impact des erreurs sur l'application

> Les erreurs peuvent entraîner des **comportements inattendus**, des **fuites de données**, ou même des **failles de sécurité**.

1.1.3 - Qu'est-ce qu'une exception en PHP ?

> Une **exception** en PHP est un événement qui se produit lors de l'exécution d'un script et qui interrompt son flux normal. Les exceptions peuvent être **capturées** et **gérées** à l'aide de blocs `try-catch`.

1.1.4 - Différence entre erreur et exception

> - Erreur : Problème au niveau du code, souvent lié à la syntaxe ou à des ressources manquantes. Gérées par le moteur PHP.
> - Exception : Problème détecté lors de l'exécution, qui peut être géré par le développeur.

1.2 - Types d'erreurs PHP

- E_ERROR : Erreurs fatales
- E_WARNING : Avertissements
- E_NOTICE : Notices
- E_PARSE : Erreurs de syntaxe
- E_DEPRECATED : Fonctionnalités obsolètes

Type d'erreur	Description	Exemple de code générant l'erreur	Effet sur le script
E_ERROR	Erreur fatale (ex : fonction inexistante)	`fonctionInexistante();`	Fatal error, arrêt
E_WARNING	Problème plus grave (ex : fichier inexistant)	`include('fichier_inexistant.php');`	Warning, continue
E_NOTICE	Problème mineur (ex : variable non définie)	`echo $varNonDefinie;`	Notice, continue
E_PARSE	Erreur de syntaxe (ex : parenthèse manquante)	`if ($a == 1 { ...`	Parse error, arrêt
E_DEPRECATED	Utilisation d'une fonction ou syntaxe obsolète	`@mysql_connect('localhost', 'user', 'pass');`	Deprecated, continue

1.3 - Configuration de base

```php
// php.ini ou en début de script
error_reporting(E_ALL);
display_errors = 1;
log_errors = 1;
error_log = "/path/to/error.log"
```

- `error_reporting(E_ALL);` : Affiche toutes les erreurs et avertissements.
- `ini_set('display_errors', 1);` : Affiche les erreurs à l'écran.
- `ini_set('log_errors', 1);` : Enregistre les erreurs dans un fichier de log.
- `ini_set('error_log', '/path/to/error.log');` : Spécifie le fichier de log des erreurs.

Tableau des valeurs possibles pour ces options configurations :

Option	Valeurs possibles
`error_reporting`	`-1` (afficher toutes les erreurs), `E_ALL` (afficher toutes les erreurs), `E_ERROR` (erreurs fatales), `E_WARNING` (avertissements), etc.
`display_errors`	`0` (désactivé), `1` (activé)
`log_errors`	`0` (désactivé), `1` (activé)
`error_log`	Chemin vers le fichier de log

Exemple : Configuration d'un script PHP

```php
<?php
// Configuration des erreurs
error_reporting(E_ALL);
ini_set('display_errors', 1);
ini_set('log_errors', 1);
ini_set('error_log', '/tmp/php_errors.log');

// Exemple de code générant une erreur
echo $undefinedVariable; // E_NOTICE

// Exemple de code générant une erreur fatale
include('non_existent_file.php'); // E_WARNING
?>
```

Exemple : Configuration de `php.ini`

```
[PHP]
error_reporting = E_ALL
display_errors = On
log_errors = On
error_log = /tmp/php_errors.log
```

1.4 - Comprendre les messages d'erreurs

> Les messages d'erreurs en PHP fournissent des **informations sur le type d'erreur**, le fichier et la ligne où elle s'est produite. Ils peuvent inclure des détails supplémentaires comme le contexte d'exécution, ce qui aide à diagnostiquer le problème.

Exemple de message d'erreur :

```
PHP Fatal error:  Uncaught Error: Call to undefined function nonExistentFunction() in /path/to/script.php:10
```

Anatomie d'un message d'erreur : *Type d'erreur : `Fatal error` *Exception : `Uncaught Error` *Message : `Call to undefined function nonExistentFunction()` *Fichier : `/path/to/script.php` *Ligne : `10`

Fonctions natives de gestion d'erreurs

2.1 - Contrôle de l'affichage des erreurs

2.1.1 - error_reporting()

La fonction `error_reporting()` permet de **définir le niveau de rapport** des erreurs. Elle peut prendre en paramètre un nombre, une **constante** ou une **combinaison de constantes** qui déterminent les **types d'erreurs** à signaler.

- `error_reporting(E_ALL)` ou `error_reporting(-1)` : Signale toutes les erreurs, y compris les avertissements et les notices.
- `error_reporting(E_ERROR | E_WARNING | E_PARSE)` : Signale les erreurs fatales, les avertissements et les erreurs de syntaxe.
- `error_reporting(E_NONE)` ou `error_reporting(0)` : Ne signale aucune erreur et désactive l'affichage des erreurs.

```php
error_reporting(0); // Ne signale aucune erreur
echo $variable_non_definie; // E_NOTICE aurait été généré, mais il est masqué
```

2.1.2 - ini_set()

La fonction `ini_set()` permet de modifier la configuration de PHP à l'exécution. Elle est souvent utilisée pour ajuster les paramètres liés aux erreurs.

```php
ini_set('display_errors', 1);
ini_set('log_errors', 1);
ini_set('error_log', '/tmp/php_errors.log');
```

2.1.3 - ini_get()

La fonction `ini_get()` permet de récupérer la valeur d'une directive de configuration PHP.

```php
$displayErrors = ini_get('display_errors'); // Récupère la valeur de display_errors
$logErrors = ini_get('log_errors'); // Récupère la valeur de log_errors
$errorLog = ini_get('error_log'); // Récupère la valeur de error_log
```

2.1.4 - display_errors vs log_errors

- `display_errors` : Affiche les erreurs directement dans le navigateur. Utile en développement, mais à éviter en production.
- `log_errors` : Enregistre les erreurs dans un fichier de log. Recommandé pour la production afin de ne pas exposer les erreurs aux utilisateurs.

```php
ini_set('display_errors', 1); // Affiche les erreurs
ini_set('log_errors', 1); // Enregistre les erreurs dans un fichier de log
ini_set('error_log', '/var/log/php_errors.log'); // Spécifie le fichier de log
```

2.2 - Gestionnaires d'erreurs personnalisés

2.2.1 - set_error_handler()

La fonction `set_error_handler()` permet de **définir une fonction personnalisée** pour gérer les erreurs. Cette fonction sera appelée chaque fois qu'une erreur PHP se produit, permettant ainsi de personnaliser le comportement de gestion des erreurs.

Exemple :

```php
 error_reporting(E_ALL); // Activer tous les rapports d'erreurs
ini_set('display_errors', 1); // Afficher les erreurs à l'écran
ini_set('error_log', __DIR__.DIRECTORY_SEPARATOR.'php_errors.log'); // Spécifie le fichier de log

// Définition d'un gestionnaire d'erreurs personnalisé
function monGestionnaireErreur($niveau, $message, $fichier, $ligne) {
    // Logique de gestion des erreurs
    error_log("Erreur [$niveau] : $message dans $fichier à la ligne $ligne");
    // Affiche le message d'erreur dans le navigateur
    echo "<div style='color:red;background:#FFEEEE;padding:10px;'>";
    echo "Erreur personnalisée [$niveau] : $message<br>";
    echo "Fichier : $fichier<br>";
    echo "Ligne : $ligne<br>";
    echo "</div>";
}

// Enregistrement du gestionnaire d'erreurs
set_error_handler('monGestionnaireErreur');


// Exemple de code générant une erreur
echo $variable_non_definie; // E_NOTICE
```

2.2.2 - restore_error_handler()

La fonction `restore_error_handler()` permet de **restaurer le gestionnaire d'erreurs précédent**. Elle est utile pour revenir à la gestion des erreurs par défaut de PHP après avoir défini un gestionnaire personnalisé.

```php
restore_error_handler(); // Restaure le gestionnaire d'erreurs précédent
```

Exceptions en PHP

3.1 - Introduction aux exceptions

3.1.1 - Qu'est-ce qu'une exception ?

Une **exception** est un événement qui se produit lors de l'exécution d'un script et qui interrompt son flux normal. Les exceptions peuvent être **capturées** et **gérées** à l'aide de blocs `try-catch`, permettant ainsi de traiter les erreurs de manière plus contrôlée.

3.2 - Fonctions de base pour les exceptions

3.2.1 - throw

La déclaration `throw` est utilisée pour **lancer une exception**. Elle crée une instance de la classe `Exception` ou d'une classe dérivée et lève cette exception.

```php
throw new Exception("Ceci est une exception");
```

3.2.2 - try et catch

```php
try {
   $resultat = 15;
   if ($resultat < 20) {
      throw new Exception("Le résultat est inférieur à 20");
   }
   // Code qui peut générer une exception
   throw new Exception("Ceci est une exception");
} catch (Exception $e) {
   // Gestion de l'exception
   echo "Erreur capturée : " . $e->getMessage();
}
finally {
   // Bloc finally (optionnel)
   echo "Bloc finally exécuté.";
}
```

3.3 - Gestion des exceptions

3.3.1 - getMessage()

La méthode `getMessage()` de la classe `Exception` retourne le message d'erreur associé à l'exception.

```php
try {
   throw new Exception("Ceci est une exception");
} catch (Exception $e) {
   echo "Message d'erreur : " . $e->getMessage();
}
```

3.3.2 - getCode()

La méthode `getCode()` retourne le code d'erreur associé à l'exception.

```php
try {
   throw new Exception("Ceci est une exception", 404);
} catch (Exception $e) {
   echo "Code d'erreur : " . $e->getCode();
}
```

3.3.3 - getFile()

La méthode `getFile()` retourne le nom du fichier dans lequel l'exception a été levée.

```php
try {
  throw new Exception("Ceci est une exception");
} catch (Exception $e) {
  echo "Fichier de l'exception : " . $e->getFile();
}
```

3.3.4 - getLine()

La méthode `getLine()` retourne le numéro de ligne dans le fichier où l'exception a été levée.

```php
try {
  throw new Exception("Ceci est une exception");
} catch (Exception $e) {
  echo "Ligne de l'exception : " . $e->getLine();
}
```

3.3.5 - Exemple complet

```php
try {
  $count = 10;
  if ($count < 20) {
     throw new Exception("Le compte est inférieur à 20");
  }
  // Code qui peut générer une exception
  throw new Exception("Ceci est une exception");
} catch (Exception $e) {
  // Gestion de l'exception
  echo "Erreur capturée : " . $e->getMessage(). "<br>"
      . "Avec le code : " . $e->getCode() . "<br>"
      . "Dans le fichier : " . $e->getFile() . "<br>"
      . "À la ligne : " . $e->getLine() . "<br>";
}
finally {
  // Bloc finally (optionnel)
  echo "Bloc finally exécuté.";
}
```