❖ Exception Handling

- Exceptions are used to change the normal flow of a script if a specified error occurs.
- Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.
- This is what normally happens when an exception is triggered:
  - The current code state is saved
  - The code execution will switch to a predefined (custom) exception handler function
  - Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code

- **1. Throwing an exception**

  - When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.
  - If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.
  - Example :

```php
<?php

// here we will throw an error but because there is no catch bloc
k it will also generate fatal error
function divide($num1 , $num2) {
    if($num2 == 0) {
        // throw the exception
        throw new Exception ('number can not be divided by zero')
;
    }
    else {
        return $num1 / $num2;
    }
}

echo divide(10,0);

?>
```

  - This will throw fatal error because there is no catch block to handle the exception.
  - To handle the exception we must have to create a catch block.

- **2. try , catch and throw**

    - To avoid the error, we have to create proper blocks or proper exception handling code to handle the error.
    - We can do this by using try , catch and throw.
    - **try block** : if we think that this code might generate an error then we should place that code into try block. If that code will generate error then it will be thrown. And if does not generate error then execution will completed normally.
    - **throw** : using throw we can trigger an exception. Each throw must have at least one catch block.
    - **catch** : A "catch" block retrieves an exception and creates an object containing the exception information.
    - Example :

```php
<?php

$age = 16;
// try block. here we will put that code which might generate exc
eption
try {
    if($age > 18) {
        echo "old enough";
    }
    else {
        // thrown the exception
        throw new Exception ('not old enough');
    }
}
// catch block to catch the exception
catch(Exception $e) {
    // getMessage() is a method of Exception class. it displays t
he message
    echo 'Error: '.$e->getMessage();
}

?>
```

    - In above example age > 18 will be false that’s why an exception will be thrown and then it will be caught by catch block.
    - That’s why the result will be “you are not old enough” instead of an error.
    - This way we can handle error using exception handling in php

- **3. Create custom exception class & custom error messages**
    - To create a custom exception class, we have to create a class that inherits or extends the Exception class.
    - The class which extends Exception class can use all the properties and methods of it.
    - So we can use all the methods of Exception class in our custom exception class.
    - Example :

```php
<?php

$mysqli_host = 'localhost';
$mysqli_user = 'root';
$mysqli_password = '';
$mysqli_database = 'training';

// created new ServerExceotion class which extends Exception class
class ServerException extends Exception {
    public function getDetails() {

        echo 'Error : ' .
$this->getMessage() . '<br><br> thrown at line ' .
$this->getLine() . ' in file ' . $this->getFile();
    }
}
class DatabaseException extends Exception {
    public function getDetails() {

        echo 'Error : ' .
$this->getMessage() . '<br><br> thrown at line ' .
$this->getLine() . ' in file ' . $this->getFile();
    }
}

try {
    if(!@$link = mysqli_connect($mysqli_host, $mysqli_user, $mysqli_
password)) {

        throw new ServerException('Can not connect with server');
    }
    else if(!mysqli_select_db($link, $mysqli_database)){

        throw new DatabaseException('can not connect with database')
;
    }
    else {
        echo 'OK.';
    }
}
catch (ServerException $se) {
    echo $se->getDetails();
}
catch (DatabaseException $de) {
    echo $de->getDetails();
}

?>
```

❖ OOP in PHP

- OOP stands for object oriented programming
- Object-Oriented programming is faster and easier to execute.
- Object-oriented programming has several advantages over procedural programming:
  - OOP is faster and easier to execute
  - OOP provides a clear structure for the programs
  - OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
  - OOP makes it possible to create full reusable applications with less code and shorter development time.

❖ Classes and object

- Classes and objects are the two main aspects of object-oriented programming.
- A class is a template for objects, and an object is an instance of a class.
- When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.
- **Class :** a class is defined by using the **class** keyword. In a class, variables are called **properties** and functions are called **methods**
- **$this** keyword is used to access or point current class' variables and methods
- **Object :** We can create multiple objects from a class.
- Each object has all the properties and methods defined in the class, but they will have different property values.
- Objects of a class is created using the **new** keyword.
- Example :

```
<?php

// created a class
class BankAccount {
    // private property
    private $balance = 7000;

    // public method
    public function withdrawMoney ($amount) {
        if($this->balance < $amount) {
            echo 'sorry, you don\'t have that much money';
        }
        else {
            $this->balance -= $amount;
            echo $msg = 'withdraw of rupees ' . $amount . ' on' . date(
'l d/m/y') . 'is done successfully. thank you <br> now the remaining ba
lance is : ' . $this->balance . '<br><br>';
        }
    }

    // another public method function
    public function depositeMoney ($amount , $name) {
        $this->balance += $amount;
        echo $msg = $amount . ' rupees deposited in your bank account o
n ' . date('l d/m/y') . 'by' . $name . '<br> updated balance is : ' . $
this->balance . '<br><br>';
    }
}

// created object of class
$rohit = new BankAccount;

// used public method of class using object
$rohit->withdrawMoney(3500);

// used public method of class using object
$rohit->depositeMoney(2530 , 'prakashbhai lalwani');

?>
```

❖ **Access modifiers (public , private , protected)**

- Properties and methods can have access modifiers which control where they can be accessed.
- There are three access modifiers:

    - public - the property or method can be accessed from everywhere. This is default
    - protected - the property or method can be accessed within the class and by classes derived from that class
    - private - the property or method can ONLY be accessed within the class

- Example :

```php
<?php
class Car {
  public $brand;
  public $color;
  public $price;

  // a public function (default)
  function set_brand($brand) {
    $this->brand = $brand;
  }

  // a protected function
  protected function set_color($color) {
    $this->color = $color;
  }

  // a private function
  private function set_price($price) {
    $this->price = $price;
  }
}

$mango = new Car();
$mango->set_brand('Audi'); // OK
$mango->set_color('Yellow'); // ERROR
$mango->set_price('3000000'); // ERROR
?>
```

❖ OOP class constants

- Constants cannot be changed once it is declared.
- Class constants can be useful if you need to define some constant data within a class.
- A class constant is declared inside a class with the const keyword.
- Class constants are case-sensitive. However, it is recommended to name the constants in all uppercase letters.
- We can access a constant from outside the class by using the class name followed by the scope resolution operator (::) followed by the constant name, like :
    - Class_name::Constant_name;

- Or, we can access a constant from inside the class by using the self keyword followed by the scope resolution operator (::) followed by the constant name, like
    - Self::constant_name;

- There are private constants as well as public constants too.

- We can not access private constants from outside the class.
- Example :

```
<?php

// class
class BankAccount {

    public const ACCOUNT_NUMBER = 312256761248;
    private const SAVING_ACCOUNT_NUMBER = 768594231589;

    //function to display constant values
    public function showConstant () {
        echo 'Account Number is : ' . self::ACCOUNT_NUMBER;
        echo '<br>Saving Account Number is : ' . self::SAVING_ACCOUNT_N
UMBER;
    }
}

// object of class BankAccount
$rohit = new BankAccount;
$rohit->showConstant();

// we can use public constant outside the class with class name
echo '<br>' . BankAccount::ACCOUNT_NUMBER;

// error because constant is private
echo BankAccount::SAVING_ACCOUNT_NUMBER;

?>
```

❖ **OOP class constructor**

- A constructor allows you to initialize an object's properties upon creation of the object.
- If you create a __construct() function, PHP will automatically call this function when you create an object from a class.
- Notice that the construct function starts with two underscores (__)
- If we inherit class and now we have to call parent class constructor from child class then we can call it like , **parent::__construct( );**
- Example :

```
<?php

// class
class BaseClass {
```

```php
    // parent class constructor
    public function __construct() {
        echo 'this is base class constructor<br>';
    }
}

class SubClass extends BaseClass {
    // child class constructor
    function __construct(){
        // called parent class constructor from child class
        parent::__construct();
        echo 'this is sub class constructor';
    }
}

$b = new SubClass;

?>
```

❖ **OOP class constructor**

- When a class is derived from another class then the **"process of deriving a new class from existing class is called inheritance"**
- The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods.
- An inherited class is defined by using the **extends** keyword.

- Example :

```php
<?php

class BankAccount {
    public $balance = 50000;

    public function displayBalance () {
        echo 'Your balance is : ' . $this->balance;
    }
}

class SavingsAccount extends BankAccount {
    function incrementBalance () {
        $this->balance += 5000;
    }
}

$obj = new SavingsAccount();
$obj->incrementBalance();
$obj->displayBalance();

?>
```

- **Database connection in OOP way**

- Example :

```php
<?php

class DatabaseConnection {
    function __construct($host , $user , $password)
    {
        if($this->connect($host , $user , $password)) {
            echo 'connected successfully';
        }
        else if (!@$this->connect($host , $user , $password)) {
            echo 'sorry, connection can not be established';
        }
    }

    private function connect($host , $user , $password) {
        if(!@mysqli_connect($host , $user , $password)) {
            return false;
        }
        else {
            return true;
        }
    }
}

$connection = new DatabaseConnection('localhost','root','');

?>
```