**MODULE – 4 (Advance PHP) OOPS**

**• What Is Object Oriented Programming?**

**=>**Object-oriented programming in PHP helps developers build reusable and complex web applications. Object-oriented programming is a programming style that refers to the association of various components and revolves around the inheritance, polymorphism, encapsulation, and abstraction concepts.

**• What Are Properties Of Object Oriented Systems?**

Object-oriented systems are built on the principles of object-oriented programming (OOP), which is a paradigm that organizes software design around the concept of objects. Here are some key properties of object-oriented systems:

1. **Encapsulation:**

* Encapsulation is the bundling of data and the methods that operate on that data into a single unit, known as an object.
* It hides the internal state of an object and requires all interactions to occur through an object's well-defined interface.

1. **Abstraction:**

* Abstraction is the process of simplifying complex systems by modeling classes based on real-world entities and their interactions.
* It focuses on the essential properties and behaviors of an object while ignoring unnecessary details.

1. **Inheritance:**
   * Inheritance is a mechanism that allows a class (subclass/derived class) to inherit properties and behaviors from another class (superclass/base class).
   * It promotes code reusability and establishes a relationship between classes, forming a hierarchy.
2. **Polymorphism:**

* Polymorphism allows objects of different types to be treated as objects of a common type.
* It enables a single interface to represent entities of different types, and it can take various forms, such as method overloading and method overriding.

1. **Modularity:**

* Modularity involves breaking down a system into smaller, self-contained, and interchangeable modules or classes.
* It promotes ease of maintenance, code reusability, and collaborative development.

1. **Message Passing:**

* Objects in an object-oriented system communicate by sending messages to each other. This involves invoking methods on objects to perform specific actions.
* Message passing is a fundamental concept in OOP and is central to the way objects interact.

1. **Encapsulation of State and Behavior:**

* Objects encapsulate both state (attributes or properties) and behavior (methods or functions).
* The state is represented by the object's attributes, and behavior is defined by the methods associated with the object.

1. **Dynamic Binding:**

* Dynamic binding (also known as late binding or runtime polymorphism) allows the selection of the specific method or function to be executed at runtime, based on the actual type of the object.
* This provides flexibility and extensibility in the code.

1. **Class and Instance:**

* A class is a blueprint or template that defines the structure and behavior of objects.
* An instance is an individual object created from a class and represents a specific occurrence of that class.

**• What Is Difference Between Class And Interface?**

### Classes:

1. **Definition:**

* A class is a blueprint for creating objects. It defines a data structure along with methods to manipulate that data.
* Classes can have attributes (fields or properties) to store data and methods to perform actions.

2 .**Implementation:**

* You can provide a complete implementation (code) for the methods in a class.
* Classes can contain constructors, which are special methods used for object initialization.

1. **Access Modifiers:**

* Classes can have access modifiers (e.g., public, private, protected) to control the visibility of their members.

1. **Inheritance:**

* Classes support the concept of inheritance, allowing one class to inherit the properties and behaviors of another class.

1. **Instantiation:**

* Objects are instances of classes. You can create multiple objects from a single class.

1. **Multiple Inheritance:**

* In most programming languages, a class cannot inherit from multiple classes (diamond problem).

### Interfaces:

**1.Definition:**

* An interface is a contract for a class. It defines a set of methods that a class implementing the interface must provide.
  + Interfaces do not contain any implementation for methods; they only declare the method signatures.

**2.Implementation:**

* + Classes that implement an interface must provide concrete implementations for all the methods declared in the interface.

**3.Access Modifiers:**

* + All methods in an interface are implicitly public and abstract. Fields are implicitly public, static, and final.

**4.Inheritance:**

* + A class can implement multiple interfaces, allowing it to inherit method signatures from multiple sources.

**5.Instantiation:**

* + Interfaces cannot be instantiated on their own. They are implemented by classes, and objects are created from these classes.

**6.Abstract Classes:**

* + Unlike interfaces, abstract classes can have a combination of abstract methods and concrete methods.

**• What Is Overloading?**

overloading refers to the ability to define multiple methods in a class with the same name but different parameter lists. PHP supports method overloading through a feature called "variable-length argument lists" or "func\_num\_args," "func\_get\_arg," and "func\_get\_args" functions. However, it's important to note that PHP doesn't support method overloading in the same way as some other programming languages, like Java or C++.

Here's a brief overview of how method overloading works in PHP:

1. **Variable-length argument lists**. **:** PHP allows you to define functions or methods with a variable number of parameters. You can use the functions **func\_num\_args()**, **func\_get\_arg()**, and **func\_get\_args()** to work with the arguments dynamically

**class Example**

**{**

public function overloadExample()

{

$numArgs = func\_num\_args();

if ($numArgs == 1)

{

// Handle case with one argument

$arg1 = func\_get\_arg(0);

}

elseif ($numArgs == 2)

{

// Handle case with two arguments

$arg1 = func\_get\_arg(0);

$arg2 = func\_get\_arg(1);

}

// Additional logic based on the number of arguments

}

}

**2.Default values for parameters:** PHP also supports default values for function or method parameters. This allows you to achieve a form of overloading by providing default values for some parameters.

class Example

{

public function overloadExample($arg1, $arg2 = null)

{

// Your logic here

}

}

**• What Is T\_PAAMAYIM\_NEKUDOTAYIM (Scope Resolution Operator (::) with Example**

**T\_PAAMAYIM\_NEKUDOTAYIM** is a term associated with the PHP programming language and refers to the Scope Resolution Operator **::**. The phrase "T\_PAAMAYIM\_NEKUDOTAYIM" is actually Hebrew and translates to "double colon" in English. In the context of PHP, it is used as a token name for the **::** operator.

The Scope Resolution Operator is used in PHP to access static members (properties and methods) of a class or to override static properties and methods. Here's an example to illustrate its usage:

class MyClass

{

const MY\_CONSTANT = 10;

public static $myStaticVariable = 20;

public static function myStaticMethod()

{

echo "Hello from myStaticMethod!";

}

}

**• What are the differences between abstract classes and interfaces?**

**Abstract Class:**

* Abstract class comes under partial abstraction.
* Abstract classes can maintain abstract methods and non abstract methods.
* In abstract classes, we can create the variables.
* In abstract classes, we can use any access specifier.
* By using 'extends' keyword we can access the abstract class features from derived class.
* Multiple inheritance is not possible.

**Interface:**

* Interface comes under fully abstraction.
* Interfaces can maintain only abstract methods.
* In interfaces, we can't create the variables.
* In interface, we can use only public access specifier.
* By using 'implement' keyword we can get interface from derived class.
* By using interfaces multiple inheritance is possible.

**• Define Constructor and Destructor?**

=> **Constructor :** Constructors are the blueprints for object creation providing values for member functions and member variables.

**Syntax:**

* **\_\_construct():**

function \_\_construct()

{

// initialize the object and its properties by assigning

//values

}

**Destructor:** Once the object is initialized, the constructor is automatically called. Destructors are for destroying objects and automatically called at the end of execution.

**Syntax:**

* **\_\_destruct():**

function \_\_destruct()

{

// destroying the object or clean up resources here

}

**• How to Load Classes in PHP?**

load classes using the **include** or **require** statements. These statements are used to include external files, such as PHP files containing class definitions.

Here's a basic example of how you can load classes in PHP:

Assume you have a file named **MyClass.php** with the following class definition:

class MyClass // MyClass.php

{

public function sayHello()

{

echo "Hello from MyClass!";

}

}

It's a common practice to use an autoloader in larger projects to automatically include class files based on the class name. Autoloaders can be registered using the **spl\_autoload\_register** function.

**• How to Call Parent Constructor?**

**parent::\_\_construct()** syntax. This is typically done within the constructor of a child class.

Here's an example:

class ParentClass

{

public function \_\_construct($param)

{

echo "Constructor of ParentClass with parameter: $param\n";

}

}

class ChildClass extends ParentClass

{

public function \_\_construct($childParam)

{

parent::\_\_construct($childParam); // Call the constructor of the parent class with the required parameter

echo "Constructor of ChildClass\n"; // Additional child class constructor code

}

}

$childObject = new ChildClass("Hello"); // Instantiate the child class

**• Are Parent Constructor Called Implicitly When Create An Object Of Class?**

**=>**create an object of a class, the parent constructor is not called implicitly. If a class has a parent class, and the child class defines its own constructor, the child class constructor is responsible for explicitly calling the parent constructor if it needs to invoke it.

For example-

class ParentClass

{

public function \_\_construct()

{

echo "Parent constructor called\n";

}

}

class ChildClass extends ParentClass

{

public function \_\_construct()

{

parent::\_\_construct(); // Explicitly calling the parent constructor

echo "Child constructor called\n";

}

}

$object = new ChildClass();

**• What Happen, If Constructor Is Defined As Private Or Protected?**

**=>** A constructor is a special method in a class that is automatically called when an object is created from that class. Typically, constructors are declared as public, allowing them to be accessed from outside the class, so objects can be instantiated.

If a constructor is defined as private or protected, it has specific implications:

**1.Private Constructor:**

* A private constructor can only be called from within the same class. Instances of the class cannot be created directly from outside the class.
* This is often used in scenarios where you want to control the instantiation of objects and force the use of static methods or a factory pattern.

**Example:-**

class MyClass

{

private function \_\_construct()

{

// private constructor logic

}

public static function createInstance()

{

return new self();

}

}

// Attempting to create an instance directly will result in an error

// $obj = new MyClass(); // Error: Call to private MyClass::\_\_construct() from context...

// Instead, use the static method to create an instance

$obj = MyClass::createInstance();

**2.Protected Constructor:-**

A protected constructor allows the constructor to be called from within the class and its subclasses. Instances cannot be created directly from outside the class or its subclasses.

This is often used in inheritance scenarios where you want to ensure that only subclasses can create instances of the base class.

Example:-

class MyBaseClass

{

protected function \_\_construct()

{

// protected constructor logic

}

}

class MySubClass extends MyBaseClass

{

public static function createInstance()

{

return new self();

}

}

// Attempting to create an instance directly will result in an error

// $obj = new MyBaseClass(); // Error: Call to protected MyBaseClass::\_\_construct()...

// Instead, use the static method from a subclass to create an instance

$obj = MySubClass::createInstance();

**• What are PHP Magic Methods/Functions? List them Write program for Static Keyword in PHP?**

**=> Magic Methods/Functions:-**

magic methods, also known as magic functions, are special methods that begin with a double underscore (**\_\_**). These methods are automatically called by the PHP interpreter under certain circumstances. Here are some commonly used magic methods:

**1.‘\_\_construct’:** This method is called when an object is created. It is commonly used for initializing object properties or performing setup tasks.

2. **‘\_\_destruct’:** This method is called when an object is no longer referenced or when the script ends. It is used for cleanup tasks or releasing resources.

3. **‘\_\_get’:** This method is called when an inaccessible property is accessed using the ‘**->’** syntax.

4. **‘\_\_set’:** This method is called when an inaccessible property is set using the ‘**->’** syntax.

5. **‘\_\_isset’:** This method is called when ‘**isset()’** or ‘**empty()’** is called on an inaccessible property.

6. **‘\_\_unset’:** This method is called when ‘**unset()’** is called on an inaccessible property.

7. **‘\_\_call’:** This method is called when an inaccessible method is invoked.

8. **‘\_\_callStatic’:** This method is called when an inaccessible static method is invoked.

9. **‘\_\_toString’:** This method is called when an object is treated as a string, for example, when using ‘**echo’** on an object.

10. **‘\_\_invoke’:** This method is called when an object is used as a function.

11. **‘\_\_clone’:** This method is called when an object is cloned using the ‘**clone’** keyword.

Example:-

<?php

class MyClass

{

private $name;

public function \_\_construct($name)

{

$this->name = $name;

echo "Object created. Name: $name\n";

}

public function getName()

{

return $this->name;

}

}

$obj = new MyClass("John");

echo "Name: " . $obj->getName() . "\n";

?>

**• Create multiple Traits and use it in to a single class?**

**=>** Traits to group functionality in a fine-grained and consistent way. Traits are similar to classes, but they are intended to group functionality in a fine-grained and consistent way. Here's an example of how you can create multiple traits and use them in a single class:

// Define Trait 1

trait Loggable

{

public function log($message)

{

echo "Log: $message\n";

}

}

// Define Trait 2

trait Authenticatable

{

public function authenticate()

{

echo "User authenticated.\n";

}

}

// Define Trait 3

trait Notifiable

{

public function notify($message)

{

echo "Notification: $message\n";

}

}

// Create a class that uses the traits

class User

{

use Loggable; // Use Trait 1

use Authenticatable; // Use Trait 2

use Notifiable; // Use Trait 3

private $username; // Additional class-specific properties and methods

public function \_\_construct($username)

{

$this->username = $username;

}

public function getUsername()

{

return $this->username;

}

}

$user = new User("john\_doe"); // Create an instance of the User class

$user->log("User created"); // Use methods from Trait 1

$user->authenticate(); // Use methods from Trait 2

$user->notify("Welcome to our platform!"); // Use methods from Trait 3

**• Write PHP Script of Object Iteration?**

**=>** object iteration refers to the process of iterating over the properties of an object using a loop. This can be achieved using the “foreach” loop, which allows you to loop through the properties of an object and perform operations on each one.

To iterate over an object, you need to define a variable to represent each property of the object, and then use the foreach loop to loop through the object properties.

**Syntax:**

foreach ($object as $property => $value)

{

// Perform operations on $value

}

**• Use of The $this keyword?**

=>the **$this** keyword is a special variable that is used within a class to refer to the current instance of the class. It is used to access properties and methods of the class from within the class itself.

Here are two main uses of the **$this** keyword

* + **Accessing Class Properties:**
  + **Calling Class Methods:**

1. **Accessing Class Properties : $this** to access class properties (variables) from within methods of the class. This is particularly useful when you have a local variable with the same name as a class property.

**Example:**

<?php

class MyClass

{

public $myProperty = "Hello, World!";

public function printProperty()

{

echo $this->myProperty;

}

}

$obj = new MyClass();

$obj->printProperty(); // Outputs: Hello, World!

1. **Calling Class Methods: $this** to call other methods within the same class. This is useful when you need to invoke other methods from within a class method.

**Example:**

<?php

class Calculator

{

private $result = 0;

public function add($number)

{

$this->result += $number;

}

public function getResult()

{

return $this->result;

}

}

$calc = new Calculator();

$calc->add(5);

$calc->add(3);

echo $calc->getResult(); // Outputs: 8

**• Consider the exercise11and add a edit link near delete link e.g. Clicking up on edit button a particular row should be open in**

adding an "edit" link next to a "delete" link in a table or list.

**1.HTML Structure:** Ensure each row has a unique identifier. You might use the **id** attribute for this purpose.

<table>

<tr id="row1">

<td>Data 1</td>

<td>Data 2</td>

<td><a href="#" class="edit-link" onclick="editRow('row1')">Edit</a></td>

<td><a href="#" class="delete-link" onclick="deleteRow('row1')">Delete</a></td>

</tr>

<!-- More rows... -->

</table>

**2.JavaScript Functions:** Add JavaScript functions to handle the edit and delete actions. These functions will be responsible for performing the necessary actions on the data or UI.

<script>

function editRow(rowId)

{

// Add logic to open the edit form or perform edit action

console.log("Editing row with ID: " + rowId);

}

function deleteRow(rowId)

{

// Add logic to delete the row

console.log("Deleting row with ID: " + rowId);

}

</script>

**3.Styling:** You might want to style the edit and delete links to make them visually distinct.

.edit-link, .delete-link

{

margin-right: 10px;

color: blue; /\* Edit link color \*/

}

.delete-link

{

//code

}

**• editing mode**

**1.Visual Studio Code (VSCode):**

* + A lightweight and powerful code editor with good PHP support through extensions.
  + Link: [Visual Studio Code](https://code.visualstudio.com/)

**2.Sublime Text:**

* + A versatile text editor with a simple interface and numerous packages for PHP development.
  + Link: [Sublime Text](https://www.sublimetext.com/)

**3.Atom:**

* + A customizable and open-source text editor with a variety of packages for PHP development.
  + Link: [Atom](https://atom.io/)

**4.PHPStorm:**

* + A dedicated IDE for PHP development with advanced features like code completion, debugging, and testing.
  + Link: [PHPStorm](https://www.jetbrains.com/phpstorm/)

**5.NetBeans:**

* + An open-source IDE with PHP support, offering features like code debugging, profiling, and more.
  + Link: [NetBeans](https://netbeans.apache.org/)

**• e.g. on the Particular row there should be filled text box with data and on the option column there should be a confirm button clicking upon it arrow should be updated.**

**1. Particular Row:**

* + This likely refers to a specific entry or record in your data.

**2.Text Box with Data:**

* + This is where information related to the particular row is displayed or entered.

**3.Option Column:**

* + This could be a column providing different actions or options for each row.

**4.Confirm Button:**

* + Clicking on this button likely triggers a specific action related to the row.

**5.Arrow Update:**

* + The arrow update suggests a visual change, possibly indicating the completion or confirmation of an action.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Interactive Table</title>

<style>

table

{

border-collapse: collapse;

width: 100%;

}

th, td

{

border: 1px solid #ddd;

padding: 8px;

text-align: left;

}

button

{

padding: 5px 10px;

cursor: pointer;

}

</style>

</head>

<body>

<table>

<thead>

<tr>

<th>Particular</th>

<th>Text Box with Data</th>

<th>Options</th>

</tr>

</thead>

<tbody>

<tr>

<td>Row 1</td>

<td><input type="text" id="dataInput1"></td>

<td>

<button onclick="confirmAction(1)">Confirm</button>

</td>

</tr>

<tr>

<td>Row 2</td>

<td><input type="text" id="dataInput2"></td>

<td>

<button onclick="confirmAction(2)">Confirm</button>

</td>

</tr>

<!-- Add more rows as needed -->

</tbody>

</table>

<script>

function confirmAction(rowNumber)

{

// Perform actions related to the confirmation, e.g., update arrow

alert('Confirmed for Row ' + rowNumber);

// You can update the arrow or perform other actions here

}

</script>

</body>

</html>

**• Create Hotel Room Booking System User can book room by 3 ways**

**1.Website Booking:**

* + **User Interface:**
    - Homepage: Displays available rooms, promotions, and general information.
    - Search Page: Allows users to filter rooms based on criteria (e.g., date, number of guests).
    - Room Details: Provides information about each room, including amenities and prices.
    - Booking Form: Users can input details such as check-in/out dates, the number of guests, and payment information.
  + **Backend Logic:**
    - Validate user inputs and check room availability.
    - Calculate total cost based on room rates and selected dates.
    - Secure payment processing.
    - Send confirmation email with booking details.
  + **Database:**
    - Store room information (availability, rates, amenities).
    - Maintain user data (bookings, personal information).

**2.Mobile App Booking:**

* + **User Interface:**
    - Similar to the website but optimized for mobile devices.
    - Responsive design for various screen sizes.
  + **Backend Logic:**
    - Utilize APIs for seamless communication between the app and server.
    - Implement push notifications for booking updates.
    - Offline functionality for viewing bookings without an internet connection.
  + **Database:**
    - Synchronize data between the app and the central database.

**3.Phone Call Booking:**

* + **Interactive Voice Response (IVR) System:**
    - Welcome and guide the user through the booking process.
    - Voice recognition for user input.
    - Provide options for room selection, dates, and payment.
  + **Backend Logic:**
    - Convert voice inputs to text and process them.
    - Check room availability and calculate the total cost.
    - Generate a booking confirmation.
  + **Database:**
    - Update room availability and store booking details.

**• Full day**

**1. User Registration:**

* Users need to register for an account to book a room.
* Collect basic information like name, email, and password for registration.
* Implement authentication to secure user accounts.

**2. Room Selection:**

* Display a list of available rooms with details (e.g., room type, price, amenities).
* Allow users to select a room for booking.

**3. Booking Methods:**

* + **a. Online Booking:**
* Users can log in, choose the desired room, and make an online payment.
* Integrate a payment gateway for secure transactions.
* Upon successful payment, confirm the booking and provide a booking ID.
  + **b. Walk-in Booking:**
* Users can physically visit the hotel and book a room at the reception.
* Receptionist enters user details and assigns a room.
* Generate a booking ID and provide it to the user.
  + **c. Phone Booking:**
* Users can call the hotel to inquire about room availability.
* Hotel staff can check availability, take user details over the phone, and book the room.
* Provide a booking confirmation with a unique ID.

**4.Reservation Management:**

* Implement a system to manage reservations.
* Avoid double bookings by updating room availability in real-time.
* Provide an admin interface to view and manage bookings.

**5. User Dashboard:**

* Users can log in to their accounts to view their booking history.
* Display upcoming and past bookings with details.

**6. Notifications:**

* Send email/SMS notifications for successful bookings and reminders.
* Notify users of any changes to their reservations.

**7. Cancellation:**

* Allow users to cancel bookings with a refund policy.
* Implement cancellation confirmation and refund processing.

**8. Reporting:**

* Generate reports for the hotel staff to analyze booking trends, occupancy rates, etc.

**9. Security Measures:**

* Implement secure coding practices.
* Use HTTPS for secure data transmission.
* Regularly update and patch the system to prevent security vulnerabilities.

**• Half day**

**1. User Registration:**

* Users should be able to create an account or log in if they already have one.
* Collect necessary information such as name, email, contact details, and password.

**2. Room Selection:**

* Display a list of available rooms with details like room type, price, and availability.
* Allow users to choose a room based on their preferences.
* Implement a calendar to show room availability for half-day bookings.

**3. Booking Process:**

* Users can initiate a booking by selecting the check-in and check-out dates and times.
* For half-day bookings, provide options for morning or afternoon check-in/check-out.
* Calculate the total cost based on the selected room and duration.

**4. Payment Integration:**

* Integrate a secure payment gateway to handle transactions.
* Accept various payment methods (credit/debit cards, online wallets, etc.).
* Confirm the booking only after successful payment.

**5. Booking Confirmation:**

* Send a confirmation email or message to the user with booking details.
* Display a confirmation page with a summary of the booking.

**6. User Dashboard:**

* Provide a user dashboard where users can view their booking history.
* Allow users to cancel or modify bookings within a specified timeframe.

**7. Admin Panel:**

* Implement an admin panel for hotel staff to manage room availability, bookings, and user accounts.
* Allow admins to add new rooms, update prices, and view booking reports.

**8. Notifications:**

* Send reminders to users before their check-in/check-out time.
* Notify users of successful bookings, cancellations, or any changes to their reservation.

**9. Reviews and Feedback:**

* Allow users to leave reviews and ratings for their stay.
* Display reviews to help future users make informed decisions.

**10. Security:**

* Implement secure authentication and authorization mechanisms.
* Ensure that sensitive information such as payment details is encrypted.

**11. Mobile Responsiveness:**

* Design the system to be mobile-friendly for users who prefer booking on smartphones or tablets.

**12. Analytics:**

* Integrate analytics to track user behavior, popular rooms, and booking patterns.

**• Custom**

### Hotel Room Booking System Components:

**1.Database:**

* + Create a database to store information about rooms, reservations, and users. Tables could include **Rooms**, **Reservations**, and **Users**.

**2.Backend:**

* + Use a backend framework (e.g., Flask, Django, Express.js) to handle server-side logic and interact with the database.

**3.User Authentication:**

* + Implement a user authentication system to secure user accounts and booking history.

**4.Web Interface:**

* + Create a web interface for users to book rooms. Use HTML, CSS, and JavaScript for the frontend. Ensure a responsive design for different devices.

**5.Mobile App:**

* + Develop a mobile app (iOS/Android) using a framework like React Native or Flutter, providing similar functionalities to the web interface.

**6.Phone Booking:**

* + Allow users to book rooms over the phone by calling a dedicated reservation hotline. Staff members can use a secure admin panel to manually enter reservations made via phone.

**7.Room Availability:**

* + Implement a mechanism to check room availability for specific dates. Update the availability status in real-time.

**8.Payment Integration:**

* + Integrate a payment gateway for online bookings. Ensure secure handling of payment information.

**• If user select for the full day than user only have selection for the checking checkout date**

**1.Radio Buttons for Stay Option:**

* + Full Day
  + Custom Stay

**2.Date Picker for Check-In:**

* + Check-In Date: [Date Picker]

**3.Date Picker for Check-Out (Conditional):**

* + Check-Out Date: [Date Picker]

**• If user select Half day than user have option of date and slot option(like user want to book room for first half – Morning (8AM to 6PM) if user select for second halfit‟s for evening (7PM to Morning 7AM)). Do proper validation like if user can book only available slot. (have touse jQuery -> Ajax, validation, Json passing).**

**=>**To implement the described functionality with jQuery, Ajax, and JSON passing for booking a room with date and slot options, you can follow these steps:

**1. HTML Structure :**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Room Booking</title>

<!-- Include jQuery -->

<script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>

</head>

<body>

<h2>Room Booking</h2>

<label for="bookingType">Select Booking Type:</label>

<select id="bookingType">

<option value="fullDay">Full Day</option>

<option value="halfDay">Half Day</option>

</select>

<div id="dateSlotOptions" style="display:none;">

<label for="bookingDate">Select Date:</label>

<input type="date" id="bookingDate">

<label for="bookingSlot">Select Slot:</label>

<select id="bookingSlot">

<option value="morning">Morning (8AM to 6PM)</option>

<option value="evening">Evening (7PM to 7AM)</option>

</select>

</div>

<button id="submitBooking">Submit Booking</button>

<script src="script.js"></script>

</body>

</html>

1. **JavaScript (script.js):**

$(document).ready(function()

{

// Event listener for booking type change

$("#bookingType").change(function()

{

if ($(this).val() === "halfDay")

{

$("#dateSlotOptions").show();

}

else

{

$("#dateSlotOptions").hide();

}

});

// Event listener for submit button

$("#submitBooking").click(function()

{

// Validate and gather user inputs

var bookingType = $("#bookingType").val();

var bookingDate = $("#bookingDate").val();

var bookingSlot = $("#bookingSlot").val();

// Validate user inputs

if (bookingType === "halfDay" && (!bookingDate || !bookingSlot))

{

alert("Please select date and slot for half-day booking.");

return;

}

// Construct data to send via Ajax

var bookingData =

{

type: bookingType,

date: bookingDate,

slot: bookingSlot

};

// Perform Ajax request

$.ajax({

url: "your\_booking\_endpoint.php", // Replace with your server-side endpoint

type: "POST",

data: JSON.stringify(bookingData),

contentType: "application/json; charset=utf-8",

dataType: "json",

success: function(response)

{

// Handle success response from server

alert("Booking successful!");

},

error: function(error)

{

// Handle error response from server

alert("Booking failed. Please try again.");

}

});

});

});

1. **Server-side (your\_booking\_endpoint.php):**

<?php

header('Content-Type: application/json');

// Retrieve JSON data from the request

$data = json\_decode(file\_get\_contents('php://input'), true);

// Validate and process booking

if ($data['type'] === 'halfDay')

{

// Perform validation for half-day booking and process accordingly

// Replace this with your actual validation and booking logic

$isValidBooking = true; // Replace with your validation logic

if ($isValidBooking)

{

// Booking successful

$response = array('status' => 'success', 'message' => 'Booking successful!');

}

else

{

// Booking failed

$response = array('status' => 'error', 'message' => 'Invalid date or slot for half-day booking.');

}

}

else

{

// Process full-day booking

// Replace this with your logic for full-day booking

$response = array('status' => 'success', 'message' => 'Booking successful!');

}

// Send JSON response back to the client

echo json\_encode($response);

?>

**Jquery**

**a)What is jQuery?**

**=>** jQuery is a fast, small, and feature-rich JavaScript library. It simplifies the process of traversing HTML documents, handling events, creating animations, and managing Ajax interactions. jQuery was created by John Resig and was first released in 2006. It quickly gained popularity due to its ease of use and the ability to perform common tasks with less code compared to raw JavaScript.

Some key features of jQuery include:

**1.DOM Manipulation:** jQuery simplifies the process of selecting and manipulating HTML elements on the page. It provides a concise syntax for common tasks like changing content, attributes, and styles.

**2.Event Handling:** jQuery makes it easy to handle various events, such as clicks, keypresses, and mouse movements. Event handling in jQuery is straightforward and cross-browser compatible.

**3.Ajax Support:** jQuery simplifies asynchronous JavaScript and XML (Ajax) requests, allowing developers to fetch data from a server and update parts of a web page without requiring a full page refresh.

**4.Animation Effects:** jQuery includes built-in functions for creating animations and transitions, making it easier to add visual effects to web pages.

**5. Utilities:** jQuery includes a variety of utility functions that streamline common tasks, such as working with arrays and objects, making AJAX requests, and handling browser compatibility issues.

**b) How are JavaScript and jQuery different?**

**=> 1.Core Purpose:**

* **JavaScript:** It is a general-purpose programming language that can be used for a wide range of tasks, not limited to web development. It is supported by all major web browsers and can be used for both client-side and server-side development.
* **jQuery:** It is a lightweight, cross-platform JavaScript library designed to simplify the client-side scripting of HTML. jQuery makes it easier to perform common tasks like DOM manipulation, event handling, animation, and AJAX with less code.

**2.Syntax:**

* **JavaScript:** It is a programming language with its syntax. It provides a way to interact with the DOM (Document Object Model) and handle events, among other things.
  + **jQuery:** It is essentially a set of functions written in JavaScript. jQuery syntax is often considered simpler and more concise than raw JavaScript, especially for tasks like DOM manipulation.

**3.DOM Manipulation:**

* + **JavaScript:** It uses native methods to manipulate the DOM. For example, **document.getElementById()**, **document.createElement()**, etc.
  + **jQuery:** It provides a simplified and often more cross-browser compatible syntax for DOM manipulation. For instance, **$("#elementId")** or **$(".className")** to select elements.

**4.Event Handling:**

* + **JavaScript:** Event handling in JavaScript can be done using the **addEventListener** method or by assigning event attributes directly in the HTML.
  + **jQuery:** It simplifies event handling with methods like **click()**, **change()**, etc. For example, **$("#myButton").click(function() { /\* do something \*/ });**.

**5.Ajax:**

* + **JavaScript:** Native JavaScript provides the **XMLHttpRequest** object for making AJAX requests.
  + **jQuery:** jQuery simplifies AJAX requests with methods like **$.ajax()** or shorthand methods like **$.get()**, **$.post()**.

**6.Cross-browser Compatibility:**

* + **JavaScript:** Developers need to be aware of and handle cross-browser compatibility issues themselves.
  + **jQuery:** One of the main reasons jQuery gained popularity was its ability to abstract away many cross-browser inconsistencies, providing a more consistent interface.

**7.Size and Performance:**

* + **JavaScript:** Native JavaScript code can sometimes be more lightweight because it doesn't include the additional abstraction layer that jQuery provides.
  + **jQuery:** While it simplifies coding, it adds an extra layer, and the library itself adds some file size overhead. In modern web development, where bandwidth and performance are crucial, some developers opt to use JavaScript directly for smaller projects.

**c) Which is the starting point of code execution in jQuery?**

**=>** the starting point of code execution is often inside a document-ready event handler. The document-ready event occurs when the DOM (Document Object Model) has been fully loaded and is ready to be manipulated. This ensures that the jQuery code runs after the HTML document has been parsed.

Syntax:-

$(document).ready(function()

{

// jQuery code goes here

});

**d) Document Load Vs Window. Load() jQuery**

**=> $(document).ready():**

* This event occurs when the DOM (Document Object Model) is ready, which means the HTML structure of the page has been fully loaded and parsed.
* It is triggered as soon as the DOM is ready, even if other external resources like images, stylesheets, and scripts are still loading.
* It is suitable for tasks that don't require waiting for all external resources to be fully loaded.

**Example:**

$(document).ready(function()

{

// Your code here

});

**$(window).load():**

* This event occurs when all assets on the page, including images and other resources, have finished loading.
* It waits for the entire page, including external resources, to be fully loaded before triggering.
* It is suitable for tasks that depend on the dimensions or content of images, as it ensures that all images have been loaded.

**Example:**

$(window).load(function()

{

// Your code here

});

**e) What is the difference between prop and attr?**

**=>1.Attribute (attr):**

* In HTML, attributes provide additional information about HTML elements and are always included in the opening tag of an element.
* Attributes define the properties of an HTML element and are used to configure or modify the behavior of the element.
* Examples of attributes include **class**, **id**, **src**, **href**, **alt**, etc.
* Attributes are part of the HTML document and can be accessed and modified using JavaScript.

**Example:**

<img src="image.jpg" alt="An example image" class="my-image">

**2.Property (prop):**

* In the context of JavaScript and the Document Object Model (DOM), properties are values associated with JavaScript objects.
* Elements in the DOM are represented as objects, and these objects have properties that can be accessed and modified using JavaScript.
* Properties often represent the current state of an element, and they can be dynamic, changing based on user interactions or other events.
* Examples of properties include **innerHTML**, **value**, **style**, etc.

**Example:**

var myElement = document.getElementById("exampleElement");

console.log(myElement.innerHTML); // Accessing the innerHTML property of an element

**f) Explain Difference Between JQuery And JavaScript?**

**=>1.Core Purpose:**

* **JavaScript:** It is a general-purpose programming language that can be used for a wide range of tasks, not limited to web development. It is supported by all major web browsers and can be used for both client-side and server-side development.
* **jQuery:** It is a lightweight, cross-platform JavaScript library designed to simplify the client-side scripting of HTML. jQuery makes it easier to perform common tasks like DOM manipulation, event handling, animation, and AJAX with less code.

**2.Syntax:**

* **JavaScript:** It is a programming language with its syntax. It provides a way to interact with the DOM (Document Object Model) and handle events, among other things.
  + **jQuery:** It is essentially a set of functions written in JavaScript. jQuery syntax is often considered simpler and more concise than raw JavaScript, especially for tasks like DOM manipulation.

**3.DOM Manipulation:**

* + **JavaScript:** It uses native methods to manipulate the DOM. For example, **document.getElementById()**, **document.createElement()**, etc.
  + **jQuery:** It provides a simplified and often more cross-browser compatible syntax for DOM manipulation. For instance, **$("#elementId")** or **$(".className")** to select elements.

**4.Event Handling:**

* + **JavaScript:** Event handling in JavaScript can be done using the **addEventListener** method or by assigning event attributes directly in the HTML.
  + **jQuery:** It simplifies event handling with methods like **click()**, **change()**, etc. For example, **$("#myButton").click(function() { /\* do something \*/ });**.

**5.Ajax:**

* + **JavaScript:** Native JavaScript provides the **XMLHttpRequest** object for making AJAX requests.
  + **jQuery:** jQuery simplifies AJAX requests with methods like **$.ajax()** or shorthand methods like **$.get()**, **$.post()**.

**6.Cross-browser Compatibility:**

* + **JavaScript:** Developers need to be aware of and handle cross-browser compatibility issues themselves.
  + **jQuery:** One of the main reasons jQuery gained popularity was its ability to abstract away many cross-browser inconsistencies, providing a more consistent interface.

**7.Size and Performance:**

* + **JavaScript:** Native JavaScript code can sometimes be more lightweight because it doesn't include the additional abstraction layer that jQuery provides.
  + **jQuery:** While it simplifies coding, it adds an extra layer, and the library itself adds some file size overhead. In modern web development, where bandwidth and performance are crucial, some developers opt to use JavaScript directly for smaller projects.

**g) How We Can Select The Specified <li> Element From The ListOf <li> Elements In <ul>?**

**=>** To select a specific **<li>** element from a list of **<li>** elements within a **<ul>** (unordered list) using JavaScript, you can use various methods.

Example:

### 1. Using JavaScript and the DOM:

Assuming you have an unordered list with an id, let's say "myList":

<ul id="myList">

<li>Item 1</li>

<li>Item 2</li>

<li>Item 3</li>

</ul>

JavaScript code to select the second **<li>** element:

// Get the <ul> element by its id

var myList = document.getElementById("myList");

// Get the second <li> element (index 1, as indices start from 0)

var secondLi = myList.getElementsByTagName("li")[1];

// Now 'secondLi' contains the reference to the second <li> element

console.log(secondLi.textContent);

### Using querySelector:

// Use querySelector to directly select the second <li> element

var secondLi = document.querySelector("#myList li:nth-child(2)");

// Now 'secondLi' contains the reference to the second <li> element

console.log(secondLi.textContent);

### 3. Using querySelectorAll:

var allLiElements = document.querySelectorAll("#myList li"); // Use querySelectorAll to select all <li> elements and then choose the second one

var secondLi = allLiElements[1]; // Get the second <li> element (index 1, as indices start from 0)

console.log(secondLi.textContent); // Now 'secondLi' contains the reference to the second <li> element

**h) In <table> Design Change The Color Of Even <tr> Elements To “green” And Change The Color Off Odd <tr> Elements To “blue” Color? Give An Example Code?**

**=>**the even and odd **<tr>** elements differently. Here's an example code:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<style>

table

{

width: 100%;

border-collapse: collapse;

}

tr:nth-child(even)

{

background-color: green;

}

tr:nth-child(odd)

{

background-color: blue;

}

th, td

{

border: 1px solid black;

padding: 8px;

text-align: left;

}

</style>

</head>

<body>

<table>

<thead>

<tr>

<th>Header 1</th>

<th>Header 2</th>

<th>Header 3</th>

</tr>

</thead>

<tbody>

<tr>

<td>Row 1, Cell 1</td>

<td>Row 1, Cell 2</td>

<td>Row 1, Cell 3</td>

</tr>

<tr>

<td>Row 2, Cell 1</td>

<td>Row 2, Cell 2</td>

<td>Row 2, Cell 3</td>

</tr>

<tr>

<td>Row 3, Cell 1</td>

<td>Row 3, Cell 2</td>

<td>Row 3, Cell 3</td>

</tr>

</tbody>

</table>

</body>

</html>

**i)How We Can Implement Animation Effects In Jquery?**

**=>** Implementing animation effects in jQuery involves using the built-in animation functions provided by jQuery. These functions allow you to animate the properties of HTML elements, such as their position, size, opacity, and more.

**Include jQuery:**

Ensure that you include the jQuery library in your HTML file. You can either download it and host it locally or use a CDN (Content Delivery Network). For example:

<script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>

**Select Elements:** Use jQuery to select the HTML elements you want to animate. You can do this using the **$** function.

<div id="myElement">Animate me!</div>

var myElement = $("#myElement");

**Animate Properties:** Use jQuery's animation functions to change the properties of the selected elements. Common properties include **height**, **width**, **opacity**, **left**, **top**, etc.

myElement.animate(

{

opacity: 0.5,

left: '+=50',

height: 'toggle'

}, 1000); // 1000 is the duration in milliseconds

**Chaining Animations:** You can chain multiple animations together using the **queue** option or by simply calling additional animation functions.

myElement

.animate({ left: '+=50' }, 500)

.animate({ opacity: 0.5 }, 500);

**Callback Functions:** You can use callback functions to execute code after an animation completes.

myElement.animate({ opacity: 0.5 }, 1000, function() {

// Code to execute after the animation is complete

});

**Easing:** jQuery provides easing options to control the speed of animations. You can use built-in options like **'swing'** or **'linear'**, or include a custom easing function.

myElement.animate({ left: '+=50' }, { duration: 500, easing: 'linear' });

**Stop Animation:** You can stop an animation in progress using the **stop** function.

myElement.stop();

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>

<title>jQuery Animation Example</title>

</head>

<body>

<button id="animateButton">Animate</button>

<div id="animatedDiv">Hello, I'm animated!</div>

<script>

$(document).ready(function()

{

$("#animateButton").click(function()

{

$("#animatedDiv").animate(

{

left: '+=100',

opacity: 0.5

}, 1000);

});

});

</script>

</body>

</html>

**j) Apply jQuery validation using library.**

**=>** jQuery Validation is a popular library for handling form validation in web applications. To use jQuery Validation, you need to include the jQuery library and the jQuery Validation plugin in your HTML file. Here's a step-by-step guide:

**Include jQuery:** Make sure to include the jQuery library in your HTML file. You can do this by adding the following line in the **<head>** section of your HTML:

<script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>

**Include jQuery Validation:** Include the jQuery Validation plugin by adding the following line below the jQuery script tag:

<script src="https://cdn.jsdelivr.net/jquery.validation/1.19.3/jquery.validate.min.js"></script>

**Create a Form:** Create an HTML form that you want to validate. For example:

<form id="myForm">

<label for="name">Name:</label>

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

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

<input type="submit" value="Submit">

</form>

**Initialize jQuery Validation:** Add a script to initialize the jQuery Validation on your form. Place this script after including the jQuery and jQuery Validation scripts:

<script>

$(document).ready(function ()

{

$("#myForm").validate();

});

</script>

**k) Create custom dynamic function for require field validator.**

**=>** To create a custom dynamic function for a required field validator in JavaScript, you can define a function that takes an object as input and checks if the required fields are present. Here's an example of a simple dynamic required field validator function:

function createRequiredFieldValidator(requiredFields)

{

return function (data)

{

const missingFields = [];

// Check each required field

requiredFields.forEach(field =>

{

if (!data.hasOwnProperty(field) || data[field] === null || data[field] === undefined || data[field] === '')

{

missingFields.push(field);

}

});

// Return the result

if (missingFields.length === 0)

{

return { isValid: true };

}

else

{

return { isValid: false, missingFields };

}

};

}

**l) Get state data by country selection (Ajax).**

**=>** To retrieve state data based on country selection using Ajax, you'll need to create a web page with HTML, JavaScript, and use a server-side script (like PHP, Node.js, or any server-side technology of your choice) to handle the data fetching. Below is a simple example using HTML, JavaScript, and PHP:

**(HTML)Index.php**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Country and State Selection</title>

</head>

<body>

<label for="country">Select Country:</label>

<select id="country" onchange="getStates()">

<!-- Populate this with your list of countries -->

<option value="usa">USA</option>

<option value="canada">Canada</option>

<!-- Add more options as needed -->

</select>

<label for="state">Select State:</label>

<select id="state"></select>

<script>

function getStates()

{

var countrySelect = document.getElementById('country');

var stateSelect = document.getElementById('state');

var selectedCountry = countrySelect.value;

// Make an Ajax request to get states based on the selected country

var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function()

{

if (xhr.readyState == 4 && xhr.status == 200)

{

// Parse the JSON response

var states = JSON.parse(xhr.responseText);

// Clear existing options

stateSelect.innerHTML = '';

// Populate the state dropdown with new options

for (var i = 0; i < states.length; i++)

{

var option = document.createElement('option');

option.value = states[i];

option.text = states[i];

stateSelect.add(option);

}

}

};

// Replace 'get\_states.php' with the path to your server-side script

xhr.open('GET', 'get\_states.php?country=' + selectedCountry, true);

xhr.send();

}

</script>

</body>

</html>

**PHP (get\_states.php):**

<?php

// Replace this with your own logic to fetch states based on the selected country

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

$selectedCountry = $\_GET['country'];

// Example: Provide states based on the selected country

$states = [];

if($selectedCountry == 'usa') {

$states = ['New York', 'California', 'Texas'];

} elseif($selectedCountry == 'canada') {

$states = ['Ontario', 'Quebec', 'British Columbia'];

}

// Send the states as a JSON response

echo json\_encode($states);

}

?>

**m) Image uploading with preview.**

**=>** To implement image uploading with a preview in a web application, you can use HTML, JavaScript, and CSS.

Example:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Image Upload with Preview</title>

<style>

#preview-container

{

display: flex;

justify-content: center;

align-items: center;

height: 200px;

border: 2px dashed #ccc;

margin-bottom: 20px;

}

#preview-img

{

max-width: 100%;

max-height: 100%;

}

</style>

</head>

<body>

<input type="file" id="imageInput" accept="image/\*" onchange="previewImage(event)">

<div id="preview-container">

<img id="preview-img" src="#" alt="Image Preview">

</div>

<script>

function previewImage(event)

{

const input = event.target;

const previewImg = document.getElementById('preview-img');

const previewContainer = document.getElementById('preview-container');

const file = input.files[0];

if (file)

{

const reader = new FileReader();

reader.onload = function (e)

{

previewImg.src = e.target.result;

};

reader.readAsDataURL(file);

previewContainer.style.display = 'block';

}

else

{

previewImg.src = '#';

previewContainer.style.display = 'none';

}

}

</script>

</body>

</html>