

**Course Overview:** This course aims to provide 12th + grade students with a comprehensive understanding of web development and programming concepts. Students will learn the fundamentals of HTML, CSS, and JavaScript, progressing to advanced topics including Bootstrap, jQuery, PHP, and MySQL database connectivity. Through hands-on projects and exercises, students will gain practical experience in building dynamic and interactive web applications.

**Course Duration:** 10 weeks (1 semester)

**Prerequisites:** Basic computer literacy and familiarity with using the internet.

**Course Curriculum:**

### **Weeks 1-2: Introduction to Web Development**

- **Understanding the Internet and the World Wide Web**
- **Introduction to HTML: Structure and Tags**
- **Creating and Styling Web Pages with CSS**

### **Weeks 3-4: Building Interactive Web Pages**

- Introduction to JavaScript: Variables, Data Types, and Operators
- Working with Functions and Events
- Implementing JavaScript in HTML documents

### **Weeks 5-6: Advanced Styling with CSS and Bootstrap**

- Advanced CSS: Selectors, Layouts, and Responsive Design
- Introduction to Bootstrap Framework: Grid System, Components, and Utilities

### **Weeks 7-8: Enhancing User Experience with jQuery**

- Introduction to jQuery Library: Selectors, Events, and Effects
- Implementing jQuery in Web Pages
- Building Interactive Features with jQuery

### **Weeks 9-10: Introduction to PHP**

- Understanding Server-Side vs. Client-Side Scripting
- Basics of PHP Syntax, Variables, and Data Types
- Building Dynamic Web Pages with PHP

### **Weeks 11-12: PHP Control Structures and Functions**

- Conditional Statements and Loops in PHP
- Creating Custom Functions and Modularizing Code

### **Weeks 13-14: Working with Forms and Form Handling in PHP**

- Building HTML Forms
- Processing Form Data with PHP
- Form Validation and Security Measures

### **Weeks 15-16: Introduction to MySQL Database**

- Understanding Relational Databases
- Basics of SQL: Creating Tables, Inserting Data, and Querying Data

### **Weeks 17-18: Connecting PHP to MySQL**

- Establishing Database Connections in PHP
- Performing CRUD Operations (Create, Read, Update, Delete) with PHP and MySQL

### **Weeks 19-20: Final Project and Review**

- Capstone Project: Design and Develop a Dynamic Web Application
- Review of Course Concepts and Skills
- Presenting and Showcasing Final Projects

### **Assessment:**

- Weekly quizzes to assess understanding of concepts
- Project-based assignments to apply learned skills
- Final project evaluation based on creativity, functionality, and adherence to best practices.

---

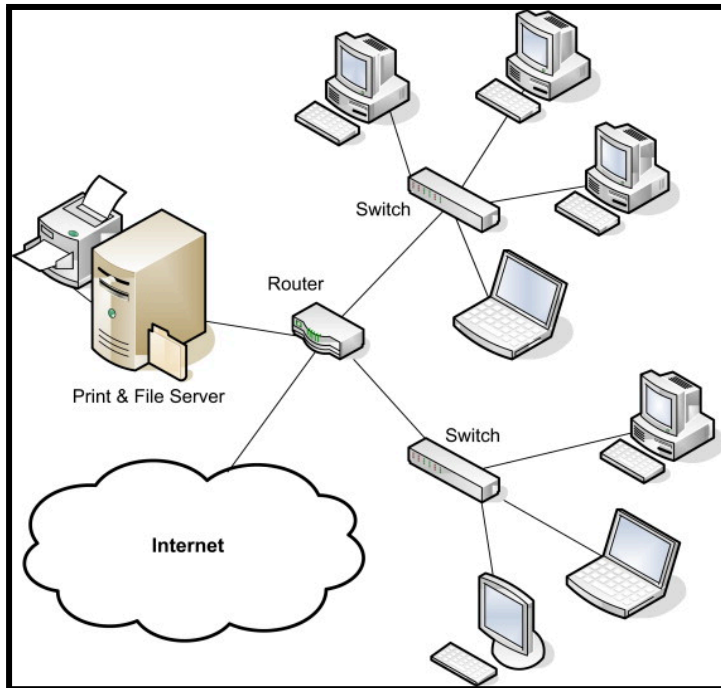
## **1-2 Introduction to Web Development > Understanding the Internet and the World Wide Web**

The Internet and the World Wide Web (WWW) are integral components of modern society, facilitating communication, information sharing, and collaboration on a global scale. In this chapter, we will explore the underlying principles of the Internet and the WWW, providing code examples, samples, and illustrations to help new students grasp these concepts.

### **1. Exploring the Internet:**

The Internet is a vast network of interconnected devices, allowing the exchange of data and information worldwide. Let's delve into the foundational concepts of the Internet:

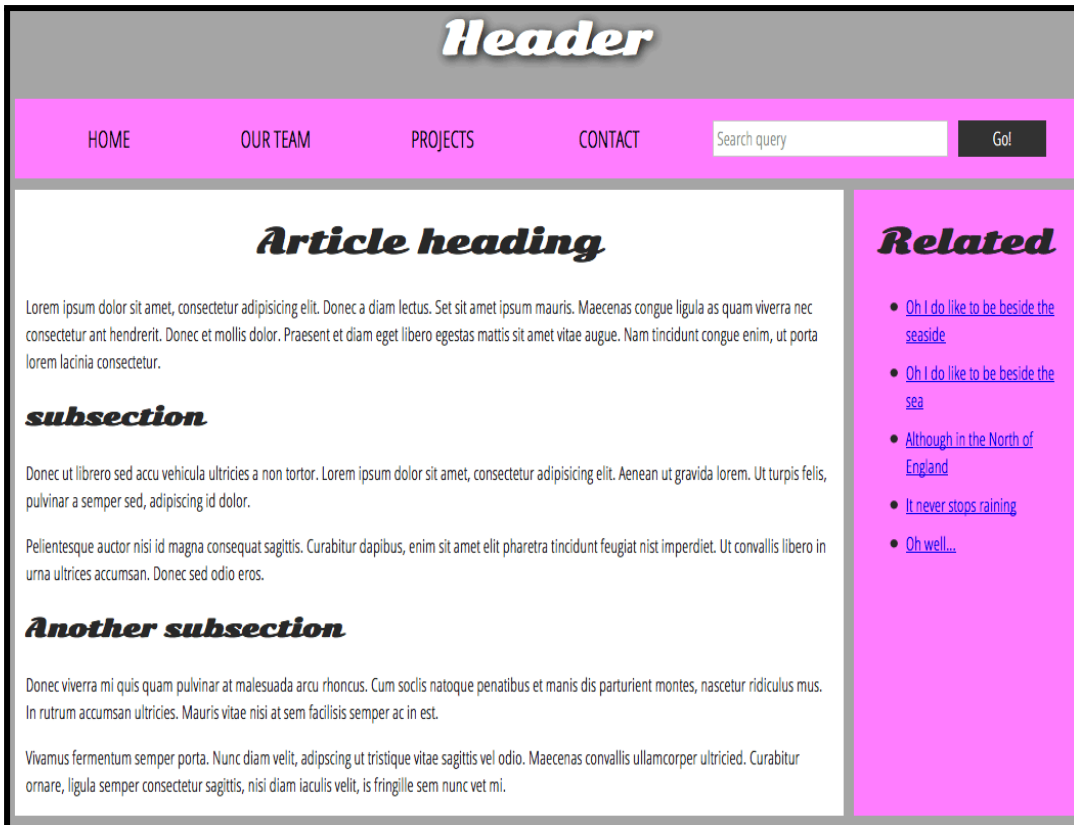
**Illustration - Network of Devices:**



**2. Introduction to the World Wide Web:**

The World Wide Web (WWW) is an information system comprising interconnected web pages and resources accessible via the Internet. Let's explore the key components of the WWW:

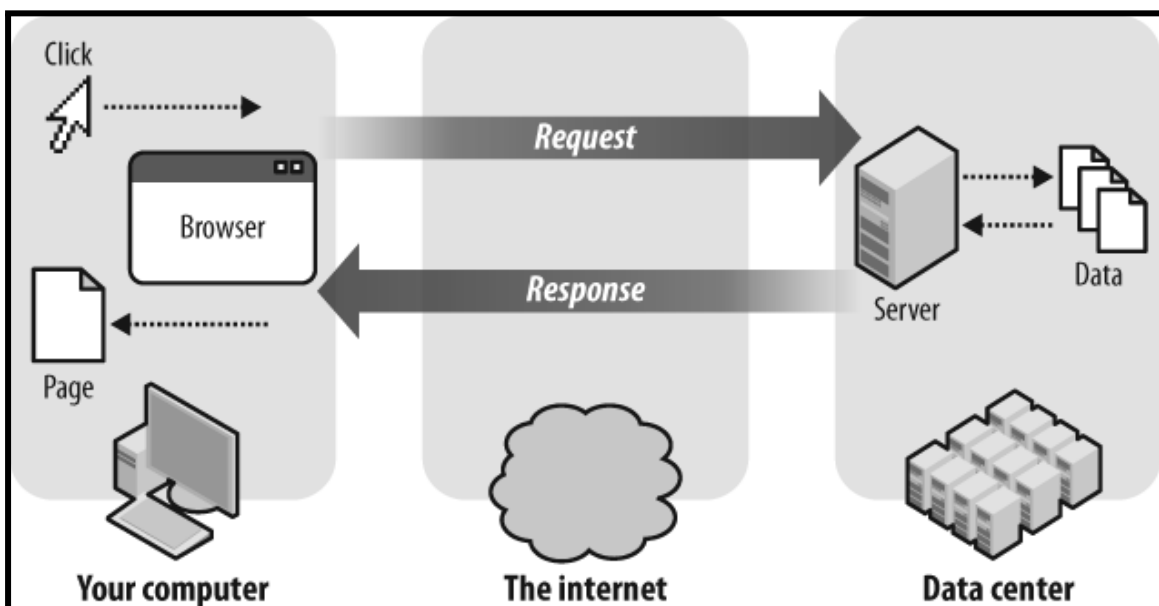
**Illustration - Web Page Structure:**



### 3. Interacting with Web Resources:

Users interact with web resources through web browsers, which render HTML documents and provide access to various online services. Let's examine how users interact with web resources:

#### Illustration - Web Browsing Process:



## 1-2 Introduction to Web Development > Introduction to HTML: Structure and Tags

HTML (Hypertext Markup Language) serves as the backbone of web development, defining the structure and content of web pages. In this chapter, we'll explore the basics of HTML structure and common tags, providing code examples and illustrations to aid understanding.

### 1. Understanding HTML Structure

HTML documents are structured using elements enclosed within tags. These elements define the content and layout of a web page. Let's examine the basic structure of an HTML document:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My First Web Page</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

#### Explanation:

- `<!DOCTYPE html>`: Declaration specifying the version of HTML being used.
- `<html>`: Root element that wraps all content on the page.
- `<head>`: Contains metadata such as the page title.
- `<title>`: Sets the title of the web page.
- `<body>`: Contains the main content of the web page.

### 2. Common HTML Tags

HTML provides a variety of tags for structuring content. Here are some commonly used ones:

- `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`: Headings of varying sizes.
- `<p>`: Represents a paragraph of text.
- `<a>`: Creates hyperlinks to other web pages or resources.
- `<img>`: Embeds images in the document.
- `<ul>`: Defines an unordered list.
- `<ol>`: Defines an ordered list.
- `<li>`: Represents a list item within `<ul>` or `<ol>`.

### 3. Code Example - HTML Structure and Tags:

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML Example</title>
</head>
<body>
  <h1>Welcome to My Website</h1>
  <p>This is a paragraph of text.</p>
  <a href="https://www.example.com">Visit Example.com</a>
  
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>
</body>
</html>
```

Illustration - HTML Structure:

## HTML Page Structure

`<!DOCTYPE html>` ← Tells version of HTML  
`<html>` ← HTML Root Element  
  
`<head>` ← Used to contain page HTML metadata  
`<title>Page Title</title>` ← Title of HTML page  
`</head>`  
  
`<body>` ← Hold content of HTML  
`<h2>Heading Content</h2>` ← HTML heading tag  
`<p>Paragraph Content</p>` ← HTML paragraph tag  
`</body>`  
  
`</html>`

### Practice Codes:

#### 1. Basic HTML Structure:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>My First Web Page</title>

</head>

<body>

  <h1>Hello, World!</h1>

  <p>This is a paragraph of text.</p>
```

```
<a href="https://www.example.com">Visit Example.com</a>
</body>
</html>
```

## 2. List Structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My List</title>
</head>
<body>
  <h2>My Favorite Fruits</h2>
  <ul>
    <li>Apple</li>
    <li>Orange</li>
    <li>Banana</li>
  </ul>
</body>
</html>
```

## 3. Form Structure:



```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Contact Form</title>

</head>

<body>

  <h2>Contact Us</h2>

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

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

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

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

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

    <label for="message">Message:</label><br>

    <textarea id="message" name="message" rows="4"
cols="50"></textarea><br>

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

  </form>

</body>

</html>
```

## 1-2 Introduction to Web Development > **Creating and Styling Web Pages with CSS**

Cascading Style Sheets (CSS) play a crucial role in web development by allowing developers to style and format HTML elements. In this chapter, we'll explore the basics of CSS and demonstrate how it can be used to create visually appealing web pages.

### 1. Understanding CSS

CSS is a style sheet language that defines the presentation of HTML documents. It allows developers to control various aspects of an HTML element's appearance, such as colors, fonts, layout, and spacing. CSS works by selecting HTML elements and applying styles to them using selectors and declarations.

### 2. Basic Syntax of CSS

CSS consists of selectors and declarations. Selectors target HTML elements, while declarations define the styles to be applied. Here's a basic syntax example:

```
selector {  
    property: value;  
}
```

### 3. Code Example - Styling HTML with CSS:

Let's demonstrate how CSS can be used to style HTML elements:

```
<!DOCTYPE html>
<html>
<head>
  <title>Styling Web Pages with CSS</title>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <h1>Welcome to My Website</h1>
  <p>This is a paragraph of text.</p>
  <a href="https://www.example.com">Visit Example.com</a>
</body>
</html>
```

CSS (styles.css):

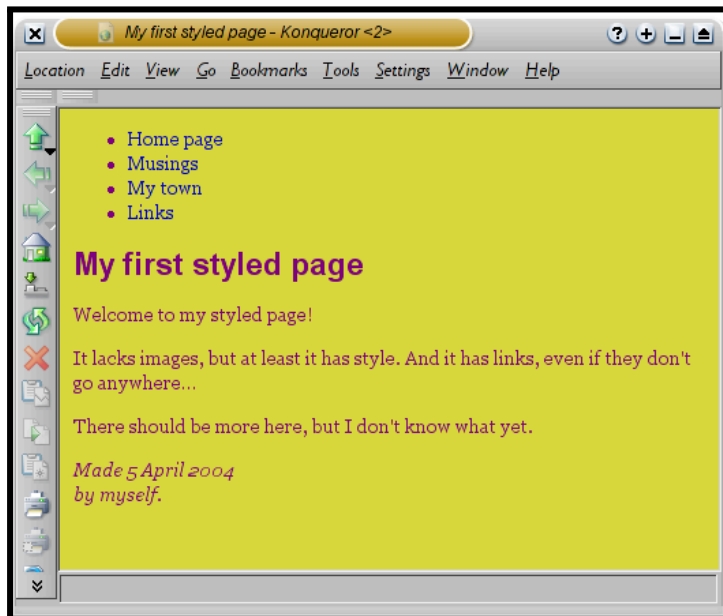
```
/* Apply styles to all <h1> elements */
h1 {
  color: blue;
  font-size: 24px;
}

/* Apply styles to all <p> elements */
p {
  color: green;
  font-size: 18px;
}

/* Apply styles to all <a> elements */
a {
  color: red;
  text-decoration: none;
}
```

Illustration - CSS Styling:

[Insert Image: Illustration showing styled HTML elements with colours and font sizes]



## Practice Examples:

### 1. Basic CSS Styling:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Basic CSS Styling</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      background-color: #f0f0f0;

    }

  </style>

</head>

<body>

  <ul>

    <li>Home page</li>

    <li>Musings</li>

    <li>My town</li>

    <li>Links</li>

  </ul>

  <h2>My first styled page</h2>

  <p>Welcome to my styled page!</p>

  <p>It lacks images, but at least it has style. And it has links, even if they don't go anywhere...</p>

  <p>There should be more here, but I don't know what yet.</p>

  <p>Made 5 April 2004<br>by myself.</p>

</body>

</html>
```

```
    h1 {
        color: blue;
    }

    p {
        font-size: 18px;
    }
</style>

</head>

<body>

    <h1>Welcome to My Website</h1>

    <p>This is a paragraph of text. We can style it using CSS.</p>

</body>

</html>
```

## 2. CSS Box Model:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>CSS Box Model</title>

<style>

    .box {
```

```
        width: 200px;

        height: 100px;

        background-color: lightblue;

        border: 2px solid blue;

        padding: 20px;

        margin: 20px;

    }

</style>

</head>

<body>

    <div class="box">

        <p>This is a box with content.</p>

    </div>

</body>

</html>
```

### 3. Responsive Layout with Media Queries:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Responsive Layout</title>
```

```
<style>

  .container {

    width: 80%;

    margin: 0 auto;

  }

  .box {

    width: 100px;

    height: 100px;

    background-color: lightblue;

    margin: 10px;

    float: left;

  }

  @media (max-width: 600px) {

    .box {

      width: 50%;

    }

  }

</style>

</head>

<body>

  <div class="container">

    <div class="box"></div>
```

```
<div class="box"></div>

<div class="box"></div>

<div class="box"></div>

</div>

</body>

</html>
```

---

### **3-4: Building Interactive Web Pages > Introduction to JavaScript: Variables, Data Types, and Operators**

JavaScript is a versatile programming language that enables the creation of interactive and dynamic web pages. In this chapter, we'll introduce the basics of JavaScript, including variables, data types, and operators, with code examples and illustrations to aid understanding.

#### **1. Introduction to JavaScript**

JavaScript is a client-side scripting language that runs in the web browser. It allows developers to add interactivity, manipulate HTML content, and respond to user actions on web pages.

#### **2. Variables in JavaScript**

Variables are containers used to store data values. In JavaScript, variables are declared using the `var`, `let`, or `const` keywords. Here's how to declare and assign values to variables:



```
// Using var (older method)
var x = 10;

// Using let (introduced in ES6)
let y = 'Hello';

// Using const (constants, value cannot be changed)
const PI = 3.14;
```

### 3. Data Types in JavaScript

JavaScript supports several data types, including:

- Primitive Data Types: such as numbers, strings, booleans, null, and undefined.
- Complex Data Types: such as arrays and objects.

```
// Number
let num = 10;

// String
let name = 'John';


// Boolean
let isTrue = true;

// Null
let empty = null;

// Undefined
let undefinedVar;

// Array
let fruits = ['apple', 'banana', 'orange'];

// Object
let person = {
  name: 'John',
  age: 30
};
```



## 4. Operators in JavaScript

JavaScript supports various operators for performing operations on variables and values. Some common operators include:

- Arithmetic Operators: +, -, \*, /, %
- Assignment Operators: =, +=, -=, \*=, /=
- Comparison Operators: ==, ===, !=, !==, >, <, >=, <=
- Logical Operators: &&, ||, !

## 5. Code Example - JavaScript Variables, Data Types, and Operators:

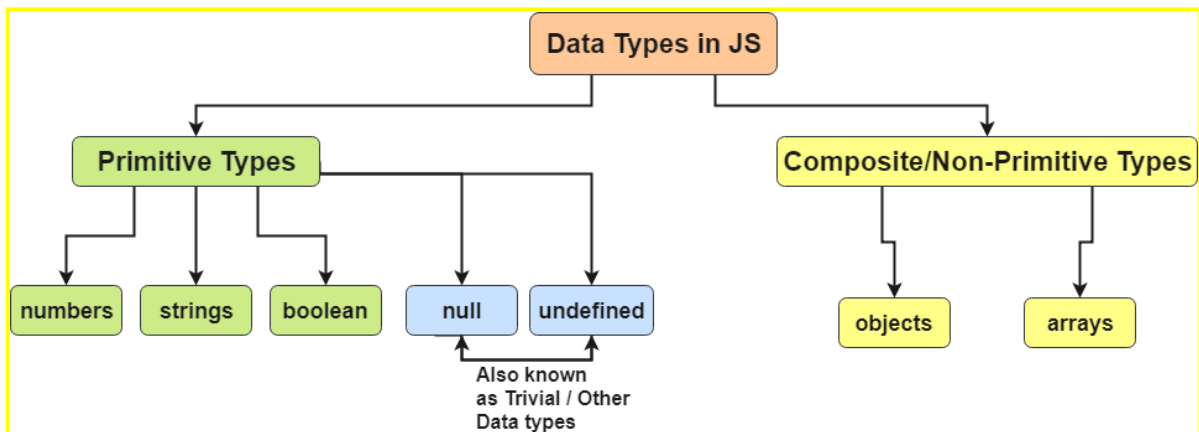
```
// Variables
let x = 10;
let name = 'John';
const PI = 3.14;

// Data Types
let isTrue = true;
let empty = null;
let undefinedVar;
let fruits = ['apple', 'banana', 'orange'];
let person = {
  name: 'John',
  age: 30
};

// Operators
let a = 5;
let b = 3;
let sum = a + b; // Addition
let product = a * b; // Multiplication
let isEqual = (a === b); // Comparison
let logicalResult = (a > 0 && b < 5); // Logical
```

Illustration - JavaScript Variables, Data Types, and Operators:

[Insert Image: Illustration showing the use of JavaScript variables, data types, and operators]



## Practice Examples:

### 1. Variables and Data Types:

```
// Declare variables of different data types

let name = "John"; // String

let age = 30; // Number

let isMale = true; // Boolean

let fruits = ["apple", "banana", "orange"]; // Array

let person = {name: "Jane", age: 25}; // Object

// Print variables to console

console.log(name);

console.log(age);

console.log(isMale);

console.log(fruits);

console.log(person);
```

### 2. Arithmetic Operators:

```
// Perform arithmetic operations
```

```
let num1 = 10;

let num2 = 5;

let sum = num1 + num2;

let difference = num1 - num2;

let product = num1 * num2;

let quotient = num1 / num2;

let remainder = num1 % num2;

console.log("Sum:", sum);

console.log("Difference:", difference);

console.log("Product:", product);

console.log("Quotient:", quotient);

console.log("Remainder:", remainder);
```

### 3. Comparison Operators:

```
// Perform comparison operations

let x = 10;

let y = 5;

console.log("x > y:", x > y);

console.log("x < y:", x < y);

console.log("x >= y:", x >= y);

console.log("x <= y:", x <= y);

console.log("x === y:", x === y);

console.log("x !== y:", x !== y);
```

#### 4. Conditional Statements:

```
// Use conditional statements

let temperature = 25;

if (temperature < 0) {

    console.log("It's freezing!");

} else if (temperature >= 0 && temperature < 10) {

    console.log("It's cold.");

} else if (temperature >= 10 && temperature < 20) {

    console.log("It's cool.");

} else if (temperature >= 20 && temperature < 30) {

    console.log("It's warm.");

} else {

    console.log("It's hot!");

}
```

### 3-4: Building Interactive Web Pages > **Working with Functions and Events**

In this chapter, we will explore how to create interactive and dynamic web pages using JavaScript functions and events. Functions allow us to encapsulate reusable blocks of code, while events enable us to respond to user actions on the web page.

#### 1. Understanding Functions

Functions are blocks of reusable code that perform a specific task. They allow us to organize and modularize our code, making it easier to manage and maintain. In JavaScript, functions can be defined using the `function` keyword.

```
// Function declaration
function greet() {
    console.log("Hello, world!");
}

// Function invocation
greet();
```

## 2. Working with Parameters and Return Values

Functions in JavaScript can accept parameters and return values, allowing for greater flexibility and reusability.

```
// Function with parameters
function greet(name) {
    console.log("Hello, " + name + "!");
}

// Function with return value
function add(x, y) {
    return x + y;
}
```

## 3. Introduction to Events

Events are actions or occurrences that happen in the browser, such as a button click or a page load. JavaScript allows us to attach event listeners to HTML elements and execute code in response to these events.

```
// Event listener for button click
document.getElementById("myButton").addEventListener("click", function() {
    console.log("Button clicked!");
});
```

## 4. Code Example - Working with Functions and Events:

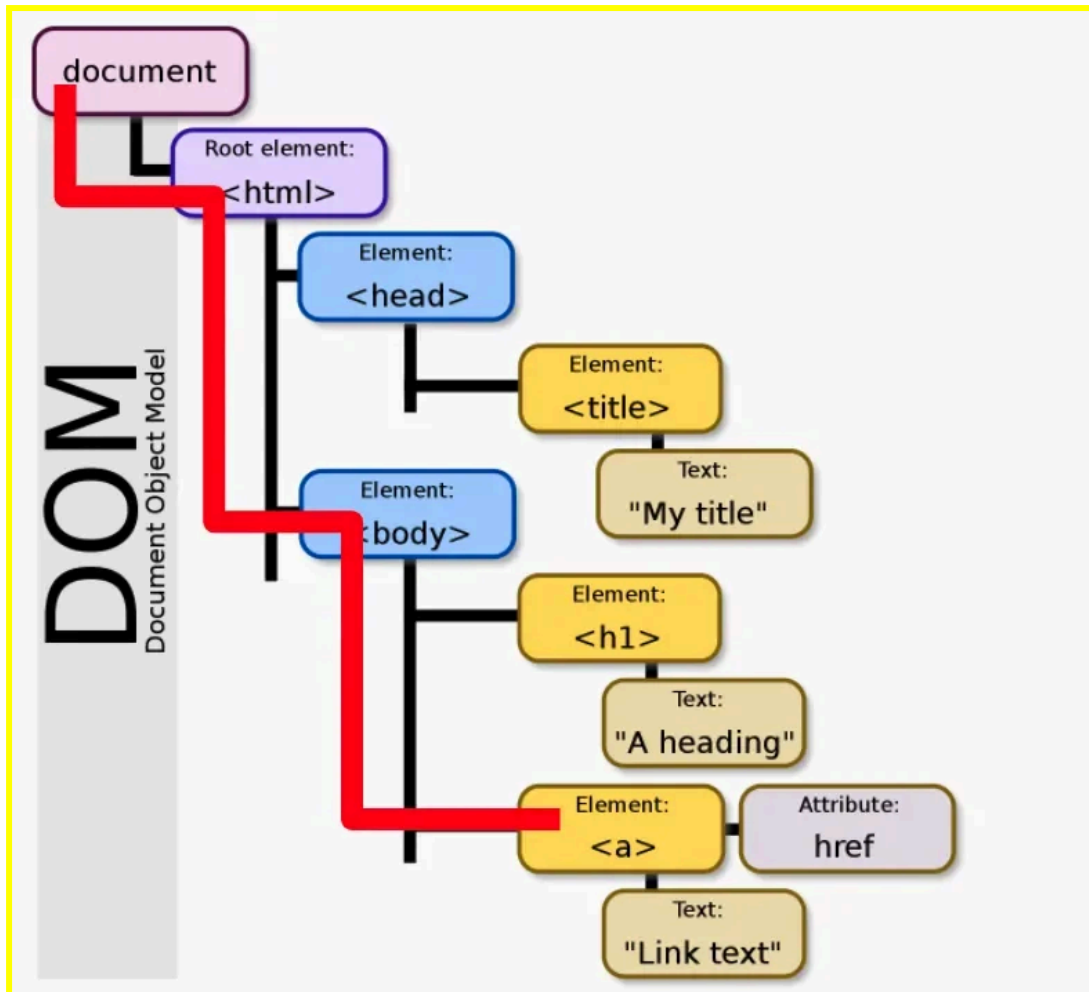
```
<!DOCTYPE html>
<html>
<head>
  <title>Working with Functions and Events</title>
  <script>
    // Function to greet the user
    function greet() {
      alert("Hello, world!");
    }

    // Function to calculate the sum of two numbers
    function add(x, y) {
      return x + y;
    }

    // Event listener for button click
    document.getElementById("myButton").addEventListener("click", function() {
      greet();
      let result = add(5, 3);
      console.log("The sum is: " + result);
    });
  </script>
</head>
<body>
  <h1>Welcome to Interactive Web Pages</h1>
  <button id="myButton">Click Me</button>
</body>
</html>
```

Illustration - Working with Functions and Events:

[Insert Image: Illustration demonstrating the use of functions and events in JavaScript]



## Practice Example:

### 1. Function Declaration and Invocation:

```
// Function declaration
function greet(name) {
  console.log("Hello, " + name + "!");
}
// Function invocation
greet("John"); // Output: Hello, John!
```

### 2. Anonymous Function and Event Handling:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```



```
<title>Event Handling</title>
<script>
  // Anonymous function and event handling
  document.getElementById("myButton").addEventListener("click",
function() {
  alert("Button clicked!");
  });
</script>
</head>
<body>
  <button id="myButton">Click Me</button>
</body>
</html>
```

### 3. Function with Return Value:

```
// Function with return value
function add(a, b) {
  return a + b;
}

// Call the function and store the result
let result = add(5, 3);
console.log("Result:", result);
// Output: Result: 8
```

### 4. Using Arrow Functions:

```
// Arrow function
let square = (num) => {
  return num * num;
}

// Call the function and store the result
let squaredValue = square(4);
console.log("Squared Value:", squaredValue);
// Output: Squared Value: 16
```

## 3-4: Building Interactive Web Pages > **Implementing JavaScript in HTML documents**

JavaScript is an essential part of web development, allowing developers to add interactivity and dynamic behaviour to HTML documents. In this chapter, we will explore how to implement JavaScript directly within HTML documents, providing code examples and illustrations.

## 1. Inline JavaScript

JavaScript code can be included directly within HTML documents using the `<script>` element. This allows developers to define scripts inline, making it convenient for small scripts or quick prototypes.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Inline JavaScript Example</title>
</head>
<body>
  <h1>Inline JavaScript Example</h1>

  <!-- Inline JavaScript -->
  <script>
    // JavaScript code goes here
    alert("Hello, world!");
  </script>
</body>
</html>
```

## 2. External JavaScript Files

For larger scripts or better organisation, JavaScript code can be placed in external files with a `.js` extension and linked to HTML documents using the `<script>` element's `src` attribute.

**HTML File:**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>External JavaScript Example</title>
  <!-- Linking to external JavaScript file -->
  <script src="script.js"></script>
</head>
<body>
  <h1>External JavaScript Example</h1>
</body>
</html>
```

### JavaScript File (script.js):

```
// JavaScript code in external file
alert("Hello, world!");
```

### 3. Code Example - Implementing JavaScript in HTML Documents:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Implementing JavaScript in HTML</title>
</head>
<body>
  <h1>Implementing JavaScript in HTML</h1>

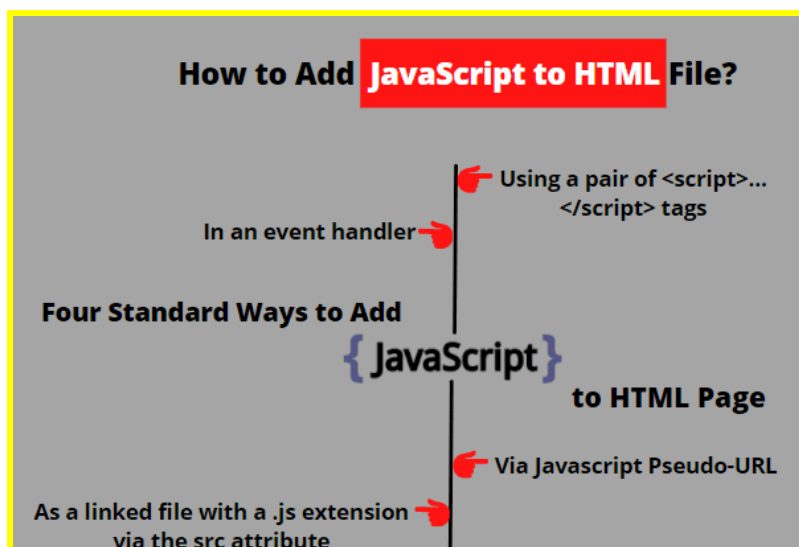
  <!-- Inline JavaScript -->
  <script>
    // JavaScript code goes here
    document.write("This content is generated dynamically using inline JavaScript")
  </script>

  <!-- External JavaScript -->
  <script src="script.js"></script>
</body>
</html>

```

Illustration - Implementing JavaScript in HTML Documents:

[Insert Image: Illustration showing the implementation of JavaScript in HTML documents]



Practice Example:

1. Inline JavaScript:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Inline JavaScript</title>

</head>

<body>

  <h1>Inline JavaScript Example</h1>

  <button onclick="alert('Button clicked!')">Click Me</button>

</body>

</html>
```

## 2. Internal JavaScript:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Internal JavaScript</title>

<script>

  // Internal JavaScript

  function greet() {
```

```
        alert('Hello, World!');
    }
</script>
</head>
<body>
    <h1>Internal JavaScript Example</h1>
    <button onclick="greet()">Say Hello</button>
</body>
</html>
```

### 3. External JavaScript:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>External JavaScript</title>
    <script src="script.js"></script>
</head>
<body>
    <h1>External JavaScript Example</h1>
    <button onclick="greet()">Say Hello</button>
</body>
```

```
</html>
```

#### 4. Event Listeners:

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

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

```
  <title>Event Listeners</title>
```

```
  <script>
```

```
    // Event listener
```

```
    document.addEventListener('DOMContentLoaded', function() {
```

```
      document.getElementById('myButton').addEventListener('click',  
function() {
```

```
        alert('Button clicked!');
```

```
      });
```

```
    });
```

```
  </script>
```

```
</head>
```

```
<body>
```

```
  <h1>Event Listeners Example</h1>
```

```
  <button id="myButton">Click Me</button>
```

```
</body>
```

```
</html>
```

---

## 5-6: Advanced Styling with CSS and Bootstrap > **Advanced CSS: Selectors, Layouts, and Responsive Design**

In this chapter, we will explore advanced CSS techniques, including selectors, layouts, and responsive design. Additionally, we will introduce Bootstrap, a popular CSS framework for building responsive and mobile-first web projects.

### 1. Advanced CSS: Selectors

CSS selectors allow developers to target specific HTML elements and apply styles to them. Understanding advanced CSS selectors is crucial for building efficient and maintainable stylesheets.

Example:

```
/* Universal selector */
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

/* Descendant selector */
div p {
  color: blue;
}

/* Class selector */
.title {
  font-size: 24px;
}

/* ID selector */
#main-content {
  background-color: #f0f0f0;
}
```



## 2. Advanced CSS: Layouts

Creating complex layouts with CSS involves techniques such as flexbox and grid layout. These layout methods provide powerful tools for organising content on the web page.

Example:

```
/* Flexbox layout */
.container {
  display: flex;
  justify-content: center;
  align-items: center;
}

/* Grid layout */
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-gap: 20px;
}
```

## 3. Advanced CSS: Responsive Design

Responsive design ensures that web pages adapt and display appropriately on various devices and screen sizes. Media queries and viewport settings are commonly used techniques for achieving responsive layouts.

Example:

```
/* Media query for small screens */
@media only screen and (max-width: 600px) {
  .container {
    flex-direction: column;
  }
}
```

## 4. Introduction to Bootstrap

Bootstrap is a popular CSS framework that provides pre-designed components and styles for building responsive web pages quickly. It includes a grid system, typography, forms, buttons, and more.

Example:

```

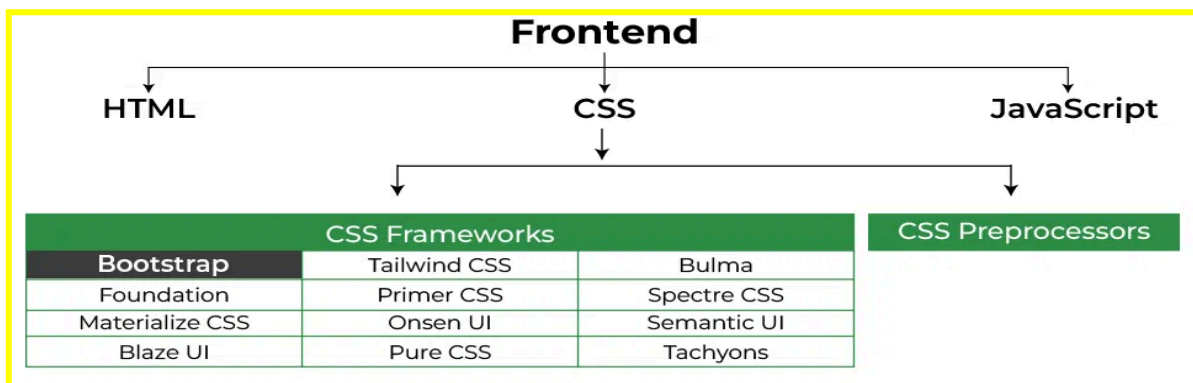
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bootstrap Example</title>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
</head>
<body>
  <div class="container">
    <h1 class="text-center">Welcome to Bootstrap</h1>
    <p class="text-muted">This is a paragraph of text.</p>
    <button class="btn btn-primary">Click Me</button>
  </div>

  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.n
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min
</body>
</html>

```

Illustration - Advanced Styling with CSS and Bootstrap:

[Insert Image: Illustration showing advanced CSS techniques and Bootstrap components]



Practice Example:

## 1. Advanced CSS Selectors:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Advanced CSS Selectors</title>

  <style>

    /* Descendant Selector */

    .container p {

      color: blue;

    }

    /* Attribute Selector */

    a[href^="https://"] {

      color: green;

    }

    /* Pseudo-class Selector */

    input:focus {

      background-color: lightyellow;

    }

  </style>

</head>
```

```
<body>

  <div class="container">

    <p>This paragraph has advanced CSS selectors applied.</p>

    <a href="https://www.example.com">Visit Example.com</a>

    <input type="text" placeholder="Type something...">

  </div>

</body>

</html>
```

## 2. CSS Flexbox Layout:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>CSS Flexbox Layout</title>

  <style>

    .container {

      display: flex;

      justify-content: space-between;

    }

    .item {

      width: 100px;
```

```
        height: 100px;

        background-color: lightblue;

        margin: 10px;

    }

</style>

</head>

<body>

    <div class="container">

        <div class="item">Item 1</div>

        <div class="item">Item 2</div>

        <div class="item">Item 3</div>

    </div>

</body>

</html>
```

### 3. Responsive Design with Media Queries:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Responsive Design with Media Queries</title>

    <style>
```

```
/* Default styles */

.container {

    width: 50%;

    margin: 0 auto;

    background-color: lightgrey;

    padding: 20px;

}

/* Media query for smaller screens */

@media (max-width: 600px) {

    .container {

        width: 80%;

    }

}

</style>

</head>

<body>

    <div class="container">

        <h2>Responsive Design Example</h2>

        <p>This container adjusts its width based on the screen size.</p>

    </div>

</body>

</html>
```

## **5-6: Advanced Styling with CSS and Bootstrap > Introduction to Bootstrap Framework: Grid System, Components, and Utilities**

In this chapter, we will introduce the Bootstrap framework, focusing on its grid system, components, and utilities. Bootstrap is a powerful CSS framework that simplifies the process of building responsive and mobile-first web projects.

### **1. Introduction to Bootstrap**

Bootstrap is a popular front-end framework developed by Twitter. It provides a set of pre-designed CSS styles and JavaScript components for building responsive web pages quickly and easily.

### **2. Bootstrap Grid System**

The Bootstrap grid system allows developers to create responsive layouts using a 12-column grid. This grid system makes it easy to create complex and flexible layouts that adapt to different screen sizes.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bootstrap Grid Example</title>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-sm-6">
        <p>Column 1</p>
      </div>
      <div class="col-sm-6">
        <p>Column 2</p>
      </div>
    </div>
  </div>
  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

### 3. Bootstrap Components

Bootstrap provides a wide range of pre-styled components such as buttons, navigation bars, forms, cards, and more. These components can be easily customised and integrated into web projects.

Example:



```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bootstrap Components Example</title>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
  <div class="container">
    <h1>Welcome to Bootstrap</h1>
    <button class="btn btn-primary">Click Me</button>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
      <a class="navbar-brand" href="#">Navbar</a>
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link" href="#">Link</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Link</a>
        </li>
      </ul>
    </nav>
  </div>
  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>

```

#### 4. Bootstrap Utilities

Bootstrap provides a set of utility classes for quickly styling elements. These utility classes can be used to apply common styles such as colors, spacing, and text alignment.

Example:

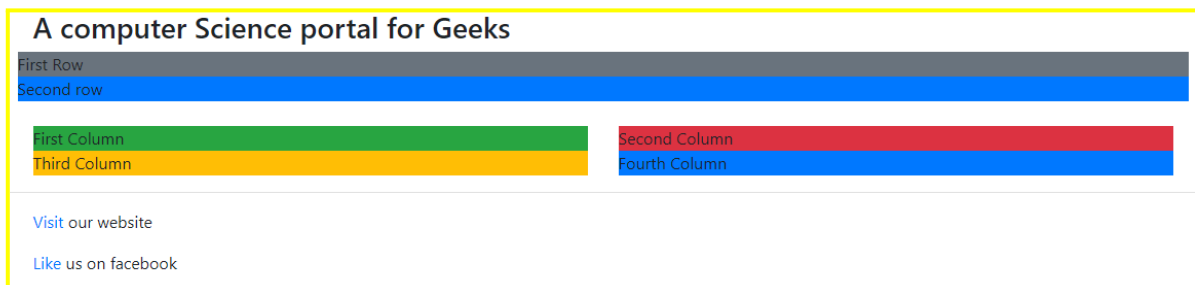
```

<div class="container">
  <p class="text-center">Centered Text</p>
  <div class="alert alert-success" role="alert">
    Success Alert
  </div>
  <button class="btn btn-danger">Danger Button</button>
</div>

```

Illustration - Introduction to Bootstrap Framework:

[Insert Image: Illustration showing the use of Bootstrap grid system, components, and utilities]



## Practice Example:

### 1. Grid System:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Bootstrap Grid System</title>
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.c
ss">
</head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-sm">
        Column 1

```

```

    </div>
    <div class="col-sm">
        Column 2
    </div>
    <div class="col-sm">
        Column 3
    </div>
</div>
</div>
</body>
</html>

```

## 2. Bootstrap Components:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Bootstrap Components</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.c
ss">
</head>
<body>
    <div class="container">
        <h1>Bootstrap Components</h1>
        <button type="button" class="btn btn-primary">Primary Button</button>
        <button type="button" class="btn btn-secondary">Secondary
Button</button>
        <button type="button" class="btn btn-success">Success Button</button>
    </div>
</body>
</html>

```

## 3. Bootstrap Utilities:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

```

```
<title>Bootstrap Utilities</title>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
  <div class="container">
    <h1 class="text-primary">Primary Heading</h1>
    <p class="text-danger">Danger Text</p>
    <div class="alert alert-success" role="alert">
      This is a success alert!
    </div>
    <div class="alert alert-warning" role="alert">
      This is a warning alert!
    </div>
  </div>
</body>
</html>
```

---

## 7-8: Enhancing User Experience with jQuery > **Introduction to jQuery Library: Selectors, Events, and Effects**

jQuery is a powerful JavaScript library that simplifies DOM manipulation, event handling, and animation effects, enabling developers to enhance user experience on web pages. In this chapter, we'll delve into the basics of jQuery, covering selectors, events, and effects, with code examples and illustrations.

### 1. Introduction to jQuery

jQuery is a fast, small, and feature-rich JavaScript library that simplifies tasks like DOM manipulation, event handling, and animation. It abstracts away many complexities of JavaScript, making it easier to write concise and efficient code for web development.

### 2. Selectors in jQuery

jQuery selectors allow developers to target and manipulate HTML elements with ease. Similar to CSS selectors, jQuery selectors enable precise targeting of elements based on their attributes, classes, IDs, and more.

Example:

```
// Selecting elements by tag name
$("p")

// Selecting elements by class name
$(".myClass")

// Selecting elements by ID
$("#myId")

// Selecting elements by attribute
$("[href]")
```

### 3. Events Handling with jQuery

jQuery simplifies event handling by providing methods to attach event listeners to DOM elements and execute code in response to user actions. Events such as clicks, mouse movements, keyboard interactions, and more can be easily handled using jQuery.

Example:

```
// Click event
$("button").click(function(){
    alert("Button clicked!");
});

// Mouseover event
$("div").mouseover(function(){
    $(this).css("background-color", "yellow");
});
```

### 4. Effects in jQuery

jQuery allows developers to add animation effects to HTML elements, enhancing the user experience. With jQuery's built-in animation methods, developers can create effects like fading, sliding, and toggling elements with minimal effort.

Example:

```
// Hide element with fade effect
$("button").click(function(){
    $("p").fadeOut();
});

// Toggle element visibility
$("button").click(function(){
    $("p").toggle();
});
```

## 5. Code Example - Introduction to jQuery Library:

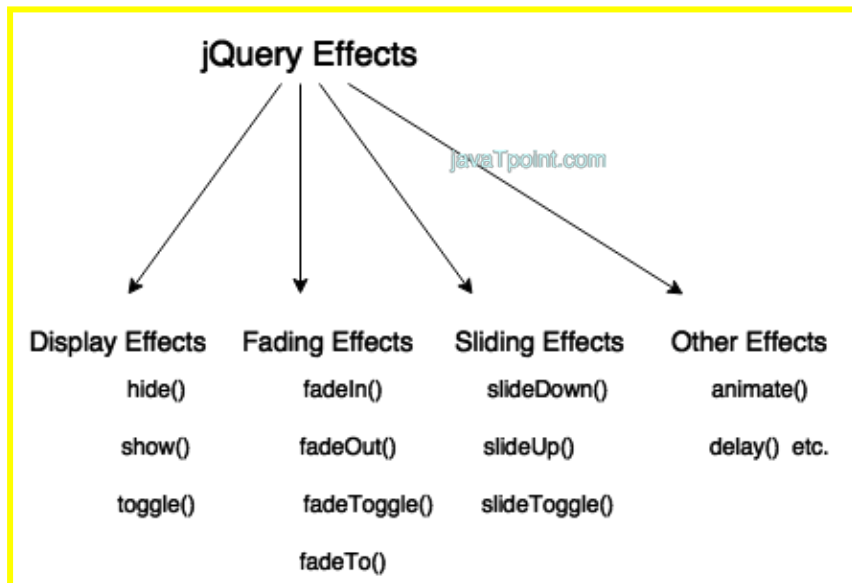
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Introduction to jQuery</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <style>
    .highlight {
      background-color: yellow;
    }
  </style>
</head>
<body>
  <h1>Welcome to jQuery</h1>
  <p class="highlight">Click this paragraph to see jQuery in action!</p>
  <button>Hide Paragraph</button>

  <script>
    // jQuery code
    $(document).ready(function(){
      $("p").click(function(){
        $(this).toggleClass("highlight");
      });

      $("button").click(function(){
        $("p").fadeOut();
      });
    });
  </script>
</body>
</html>
```

## 6. Illustration - Introduction to jQuery Library:

[Insert Image: Illustration demonstrating jQuery selectors, events, and effects in action]



## Practice Example:

### 1. jQuery Selectors:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>jQuery Selectors</title>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

  <style>

    .highlight {

      background-color: yellow;

    }

  </style>

</head>

</html>
```

```
</style>

</head>

<body>

  <h2>jQuery Selectors Example</h2>

  <p id="paragraph">This is a paragraph.</p>

  <button id="changeColor">Change Color</button>

  <script>

    // jQuery code

    $(document).ready(function(){

      $('#changeColor').click(function(){

        $('#paragraph').toggleClass('highlight');

      });

    });

  </script>

</body>

</html>
```

## 2. jQuery Events:

```
<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8">
```



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

<title>jQuery Events</title>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

</head>

<body>

  <h2>jQuery Events Example</h2>

  <button id="myButton">Click Me</button>

  <script>

    // jQuery code

    $(document).ready(function(){

      $('#myButton').click(function(){

        alert('Button clicked!');

      });

    });

  </script>

</body>

</html>
```

### 3. jQuery Effects:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">
```

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

<title>jQuery Effects</title>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

</head>

<body>

  <h2>jQuery Effects Example</h2>

  <p id="message">This is a message.</p>

  <button id="hide">Hide Message</button>

  <button id="show">Show Message</button>

  <script>

    // jQuery code

    $(document).ready(function(){

      $('#hide').click(function(){

        $('#message').hide();

      });

      $('#show').click(function(){

        $('#message').show();

      });

    });

  </script>

</body>

</html>
```

## 7-8: Enhancing User Experience with jQuery > **Implementing jQuery in Web Pages**

In this chapter, we will explore how to implement jQuery in web pages to enhance user experience. We'll cover how to include jQuery in HTML documents, basic usage, and provide code examples and illustrations to illustrate its implementation.

### 1. Introduction to Implementing jQuery

jQuery is a JavaScript library designed to simplify HTML DOM manipulation, event handling, and animation, making it an excellent tool for enhancing user experience on web pages. Integrating jQuery into web pages is straightforward and can significantly improve the development process.

### 2. Including jQuery in HTML Documents

jQuery can be included in HTML documents by linking to it directly from a content delivery network (CDN) or by downloading and hosting it locally. The preferred method is often to link to a CDN, as it provides faster loading times and reduces the need for local storage.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Implementing jQuery</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <!-- Your HTML content here -->
</body>
</html>
```

### 3. Basic Usage of jQuery


Once jQuery is included in the HTML document, it can be used to select elements, manipulate the DOM, handle events, and perform animations. jQuery syntax is

concise and intuitive, making it easy to use for both beginners and experienced developers.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Implementing jQuery</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <h1>Welcome to jQuery</h1>
  <button id="myButton">Click Me</button>

  <script>
    // jQuery code
    $(document).ready(function(){
      $("#myButton").click(function(){
        alert("Button clicked!");
      });
    });
  </script>
</body>
</html>
```



#### 4. Illustration - Implementing jQuery in Web Pages:

[Insert Image: Illustration demonstrating the implementation of jQuery in a web page]



## Practice Example:

### 1. External jQuery Script:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Implementing jQuery</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <h2>Implementing jQuery Example</h2>
  <p>Click the button to hide this paragraph.</p>
  <button id="hideButton">Hide Paragraph</button>

  <script src="script.js"></script>
</body>
</html>
```

### **script.js:**

```
// jQuery code
$(document).ready(function(){
  $('#hideButton').click(function(){
    $('p').hide();
  });
});
```

### **2. jQuery CDN Usage:**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Implementing jQuery</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <h2>Implementing jQuery Example</h2>
  <p>Hover over the text to change its color.</p>

  <script>
    // jQuery code
    $(document).ready(function(){
      $('p').hover(function(){
        $(this).css('color', 'red');
      }, function(){
        $(this).css('color', 'black');
      });
    });
  </script>
</body>
</html>
```

### **3. jQuery Animation:**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Implementing jQuery</title>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<style>
  .box {
    width: 100px;
    height: 100px;
    background-color: lightblue;
    position: relative;
  }
</style>
</head>
<body>
  <h2>Implementing jQuery Example</h2>
  <div class="box"></div>

  <script>
    // jQuery code
    $(document).ready(function(){
      $('.box').click(function(){
        $(this).animate({
          left: '250px',
          opacity: '0.5',
          height: '150px',
          width: '150px'
        }, 'slow');
      });
    });
  </script>
</body>
</html>

```

## 7-8: Enhancing User Experience with jQuery > **Building Interactive Features with jQuery**

In this chapter, we'll focus on building interactive features using jQuery to further enhance user experience on web pages. We'll explore how to create dynamic content, handle user input, and implement interactive elements with code examples and illustrations.

## 1. Dynamic Content Manipulation

jQuery simplifies the process of dynamically updating content on web pages. Using jQuery, developers can easily add, remove, or modify HTML elements based on user interactions or data changes.

Example: Adding Elements Dynamically

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dynamic Content Manipulation</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <ul id="list"></ul>
  <button id="addButton">Add Item</button>

  <script>
    $(document).ready(function(){
      $("#addButton").click(function(){
        $("#list").append("<li>New Item</li>");
      });
    });
  </script>
</body>
</html>
```

## 2. Handling User Input

jQuery simplifies handling user input, such as form submissions, button clicks, and keyboard events. With jQuery event handlers, developers can respond to user actions and perform corresponding actions or validations.

Example: Form Validation



```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Form Validation</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <form id="myForm">
    <input type="text" id="username" placeholder="Username">
    <input type="password" id="password" placeholder="Password">
    <button type="submit">Submit</button>
  </form>

  <script>
    $(document).ready(function(){
      $("#myForm").submit(function(event){
        var username = $("#username").val();
        var password = $("#password").val();

        if(username === "" || password === "") {
          alert("Please fill in all fields");
          event.preventDefault();
        }
      });
    });
  </script>
</body>
</html>

```

### 3. Implementing Interactive Elements

jQuery enables developers to create interactive elements such as accordions, tabs, sliders, and modal dialogs easily. These interactive components improve usability and engagement on web pages.

Example: Modal Dialog

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Modal Dialog</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <style>
    .modal {
      display: none;
      position: fixed;
      top: 50%;
      left: 50%;
      transform: translate(-50%, -50%);
      background-color: rgba(0, 0, 0, 0.5);
      padding: 20px;
    }
    .modal-content {
      background-color: white;
      padding: 20px;
      border-radius: 5px;
    }
    .close {
      float: right;
      cursor: pointer;
    }
  </style>
</head>
<body>
  <button id="openModal">Open Modal</button>
  <div id="myModal" class="modal">
    <div class="modal-content">
      <span class="close">&times;</span>
      <p>This is a modal dialog.</p>
    </div>
  </div>

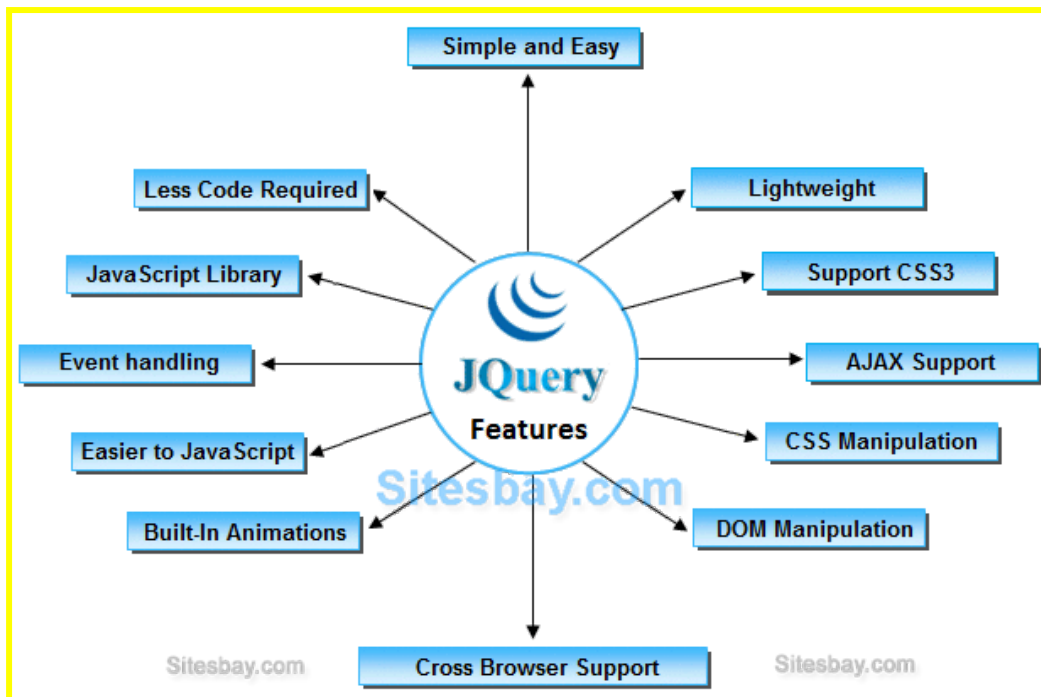
  <script>
    $(document).ready(function(){
      $("#openModal").click(function(){
        $("#myModal").fadeIn();
      });

      $(".close").click(function(){
        $("#myModal").fadeOut();
      });
    });
  </script>
</body>
</html>

```

#### 4. Illustration - Building Interactive Features with jQuery:

[Insert Image: Illustration demonstrating interactive features built with jQuery]



## Practice Example:

### 1. Toggle Class on Click:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Building Interactive Features with jQuery</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <style>
    .highlight {
      background-color: yellow;
    }
  </style>
</head>
<body>
  <h2>Toggle Class Example</h2>
  <p>This paragraph can be highlighted by clicking the button.</p>
  <button id="toggleButton">Toggle Highlight</button>

  <script>
    // jQuery code
    $(document).ready(function(){

```

```

        $('#toggleButton').click(function(){
            $('p').toggleClass('highlight');
        });
    });
</script>
</body>
</html>

```

## 2. Slide Toggle Effect:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Building Interactive Features with jQuery</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <style>
    .content {
      display: none;
      padding: 10px;
      background-color: lightblue;
    }
  </style>
</head>
<body>
  <h2>Slide Toggle Example</h2>
  <button id="toggleButton">Toggle Content</button>
  <div class="content">
    <p>This is some hidden content that can be toggled.</p>
  </div>

  <script>
    // jQuery code
    $(document).ready(function(){
      $('#toggleButton').click(function(){
        $('.content').slideToggle();
      });
    });
  </script>
</body>
</html>

```

### 3. Fade In/Out Effect:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Building Interactive Features with jQuery</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <style>
    .content {
      display: none;
      padding: 10px;
      background-color: lightblue;
    }
  </style>
</head>
<body>
  <h2>Fade In/Out Example</h2>
  <button id="fadeInButton">Fade In</button>
  <button id="fadeOutButton">Fade Out</button>
  <div class="content">
    <p>This is some content that can be faded in and out.</p>
  </div>

  <script>
    // jQuery code
    $(document).ready(function(){
      $('#fadeInButton').click(function(){
        $('.content').fadeIn();
      });
      $('#fadeOutButton').click(function(){
        $('.content').fadeOut();
      });
    });
  </script>
</body>
</html>
```

---

## **9-10: Introduction to PHP >Understanding Server-Side vs. Client-Side Scripting**

In this chapter, we will introduce PHP and explore the differences between server-side and client-side scripting. We'll provide code examples and illustrations to help you understand the concepts effectively.

### **1. Introduction to PHP**

PHP (Hypertext Preprocessor) is a widely-used open-source server-side scripting language that is especially suited for web development. It is embedded within HTML and is executed on the server, generating dynamic content that is then sent to the client's web browser.

### **2. Server-Side vs. Client-Side Scripting**

Server-side scripting and client-side scripting are two different approaches to generating dynamic content on web pages:

- **Server-Side Scripting:** In server-side scripting, scripts are executed on the web server before the content is sent to the client's browser. PHP is a popular server-side scripting language used to generate dynamic web pages based on user input, database queries, and other server-side actions.
- **Client-Side Scripting:** In client-side scripting, scripts are executed on the client's web browser after the web page has been loaded. JavaScript is the most commonly used client-side scripting language and is used to enhance the interactivity and functionality of web pages without requiring server interaction.

### **3. Code Example - Server-Side Scripting with PHP:**

```

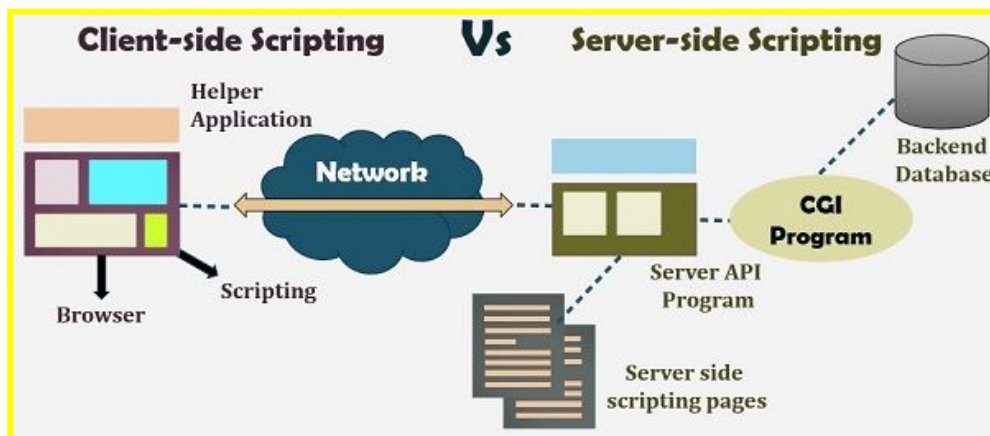
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Server-Side Scripting with PHP</title>
</head>
<body>
  <h1>Welcome, <?php echo "User"; ?>!</h1>
  <p>The current server time is <?php echo date("H:i:s"); ?>.</p>
</body>
</html>

```

In this example, PHP code is embedded within HTML tags. The server executes the PHP code and generates dynamic content (e.g., user's name and current server time) before sending the HTML response to the client's browser.

#### 4. Illustration - Server-Side vs. Client-Side Scripting:

[Insert Image: Illustration showing the difference between server-side and client-side scripting processes]



#### Practice Example:

##### 1. Client-Side Scripting with HTML and JavaScript:

```

<!DOCTYPE html>

<html lang="en">

```

```
<head>

  <meta charset="UTF-8">

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

  <title>Client-Side Scripting Example</title>

</head>

<body>

  <h2>Client-Side Scripting Example</h2>

  <p id="output">Click the button to change this text.</p>

  <button onclick="changeText()">Change Text</button>

  <script>

    function changeText() {

      document.getElementById("output").innerHTML = "Text changed using
client-side scripting!";

    }

  </script>

</body>

</html>
```

## 2. Server-Side Scripting with PHP:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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



```
<title>Server-Side Scripting Example</title>

</head>

<body>

  <h2>Server-Side Scripting Example</h2>

  <?php

    $message = "Text changed using server-side scripting!";

    echo "<p>$message</p>";

  ?>

</body>

</html>
```

### 3. Combined Client-Side and Server-Side Scripting:

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Combined Scripting Example</title>

</head>

<body>

  <h2>Combined Scripting Example</h2>

  <?php

    $message = "Hello from PHP!";
```

```
        echo "<p>$message</p>";

?>

<button onclick="changeText()">Change Text</button>

<script>

    function changeText() {

        document.getElementById("output").innerHTML = "Text changed using
client-side scripting!";

    }

</script>

</body>

</html>
```

## 9-10: Introduction to PHP > **Basics of PHP Syntax, Variables, and Data Types**

In this chapter, we'll explore the fundamentals of PHP, covering syntax, variables, and data types. We'll provide code examples and illustrations to help you grasp these concepts effectively.

### 1. Introduction to PHP Syntax

PHP code is embedded within HTML and is executed on the server before the resulting HTML is sent to the client's browser. Here are some key points about PHP syntax:

- PHP code is enclosed within `<?php` and `?>` tags.
- Statements end with a semicolon (`;`).
- Comments can be single-line (`//`) or multi-line (`/* */`).

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>PHP Syntax Example</title>
</head>
<body>
  <h1><?php echo "Hello, World!"; ?></h1>
  <?php
    // This is a PHP comment
    echo "This is a PHP statement.";
  ?>
</body>
</html>
```

## 2. Variables in PHP

Variables in PHP are used to store data that can be manipulated and referenced throughout the script. PHP variables start with the dollar sign (\$) followed by the variable name. Variable names are case-sensitive and must begin with a letter or underscore.

Example:

```
<?php
  $name = "John";
  $age = 30;
  $isStudent = true;
?>
```

## 3. Data Types in PHP

PHP supports various data types, including:

- **String:** A sequence of characters enclosed in quotes (" " or ' ').

- **Integer:** A whole number without a decimal point.
- **Float:** A number with a decimal point.
- **Boolean:** Represents true or false.
- **Array:** A collection of key-value pairs.
- **Object:** Instances of user-defined classes.
- **Null:** Represents a variable with no value.

Example:

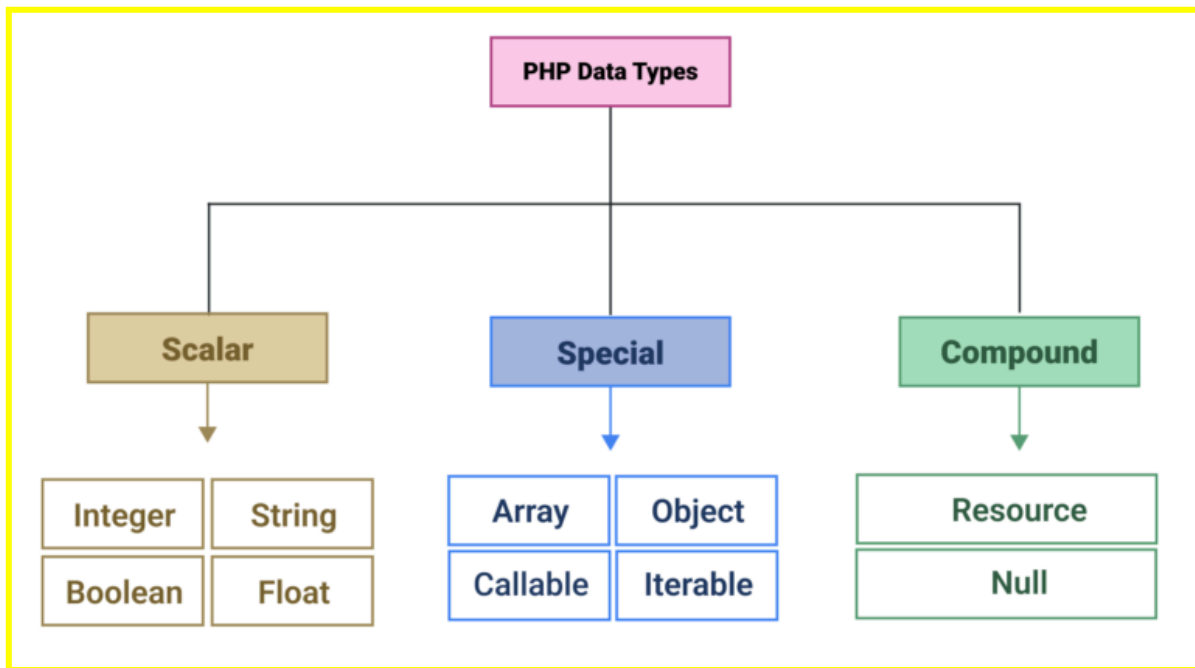
```
<?php
    $name = "John"; // String
    $age = 30; // Integer
    $height = 5.9; // Float
    $isStudent = true; // Boolean

    $colors = array("red", "green", "blue"); // Array
    $person = new stdClass(); // Object
    $person->name = "Alice";
    $person->age = 25;

    $nullVariable = null; // Null
?>
```

#### 4. Illustration - Basics of PHP Syntax, Variables, and Data Types:

[Insert Image: Illustration demonstrating PHP data types]



## Practice Example:

### 1. Variables and Data Types:

```

<?php
// Declare variables of different data types
$name = "John"; // String
$age = 30; // Integer
$isMale = true; // Boolean
$height = 6.2; // Float
$fruits = array("apple", "banana", "orange"); // Array
$person = array("name" => "Jane", "age" => 25); // Associative Array

// Print variables
echo "Name: " . $name . "<br>";
echo "Age: " . $age . "<br>";
echo "Is Male: " . ($isMale ? "Yes" : "No") . "<br>";
echo "Height: " . $height . "<br>";
echo "Fruits: " . implode(", ", $fruits) . "<br>";
echo "Person: " . $person['name'] . ", Age: " . $person['age'];
?>
  
```

### 2. Arithmetic Operations:

```

<?php
  
```

```

// Perform arithmetic operations
$num1 = 10;
$num2 = 5;

$sum = $num1 + $num2;
$difference = $num1 - $num2;
$product = $num1 * $num2;
$quotient = $num1 / $num2;
$remainder = $num1 % $num2;

// Print results
echo "Sum: " . $sum . "<br>";
echo "Difference: " . $difference . "<br>";
echo "Product: " . $product . "<br>";
echo "Quotient: " . $quotient . "<br>";
echo "Remainder: " . $remainder;
?>

```

### 3. Conditional Statements:

```

<?php
// Use conditional statements
$temperature = 25;

if ($temperature < 0) {
    echo "It's freezing!";
} elseif ($temperature >= 0 && $temperature < 10) {
    echo "It's cold.";
} elseif ($temperature >= 10 && $temperature < 20) {
    echo "It's cool.";
} elseif ($temperature >= 20 && $temperature < 30) {
    echo "It's warm.";
} else {
    echo "It's hot!";
}
?>

```

## 9-10: Introduction to PHP > **Building Dynamic Web Pages with PHP**

In this chapter, we'll explore how PHP is used to build dynamic web pages. We'll cover topics such as embedding PHP in HTML, processing form data, interacting with databases, and generating dynamic content.

## 1. Embedding PHP in HTML

PHP code is embedded within HTML using special tags (`<?php` and `?>`). This allows developers to mix PHP and HTML to create dynamic web pages.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dynamic Web Page with PHP</title>
</head>
<body>
  <h1>Welcome, <?php echo "User"; ?>!</h1>
  <p>The current server time is <?php echo date("H:i:s"); ?>.</p>
</body>
</html>
```

## 2. Processing Form Data

PHP is commonly used to process form submissions from users. When a form is submitted, PHP can retrieve the submitted data and perform actions based on it, such as storing it in a database or sending an email.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Form Processing with PHP</title>
</head>
<body>
  <form action="process.php" method="post">
    <input type="text" name="username" placeholder="Username">
    <input type="password" name="password" placeholder="Password">
    <button type="submit">Submit</button>
  </form>
</body>
</html>
```

process.php:

```
<?php
  $username = $_POST['username'];
  $password = $_POST['password'];

  // Process the form data (e.g., validate, store in database)
?>
```

### 3. Interacting with Databases

PHP can interact with databases to retrieve, insert, update, and delete data. This allows developers to build dynamic web applications that store and retrieve information from databases.

Example:



```

<?php
    // Connect to the database
    $connection = mysqli_connect("localhost", "username", "password", "database");

    // Check connection
    if (!$connection) {
        die("Connection failed: " . mysqli_connect_error());
    }

    // Perform SQL query
    $sql = "SELECT * FROM users";
    $result = mysqli_query($connection, $sql);

    // Fetch data and display it
    if (mysqli_num_rows($result) > 0) {
        while ($row = mysqli_fetch_assoc($result)) {
            echo "Username: " . $row['username'] . "<br>";
        }
    } else {
        echo "No users found";
    }

    // Close connection
    mysqli_close($connection);
?>

```

#### 4. Generating Dynamic Content

PHP can generate dynamic content based on user input, database queries, or other conditions. This allows developers to create personalised and interactive web pages.

Example:

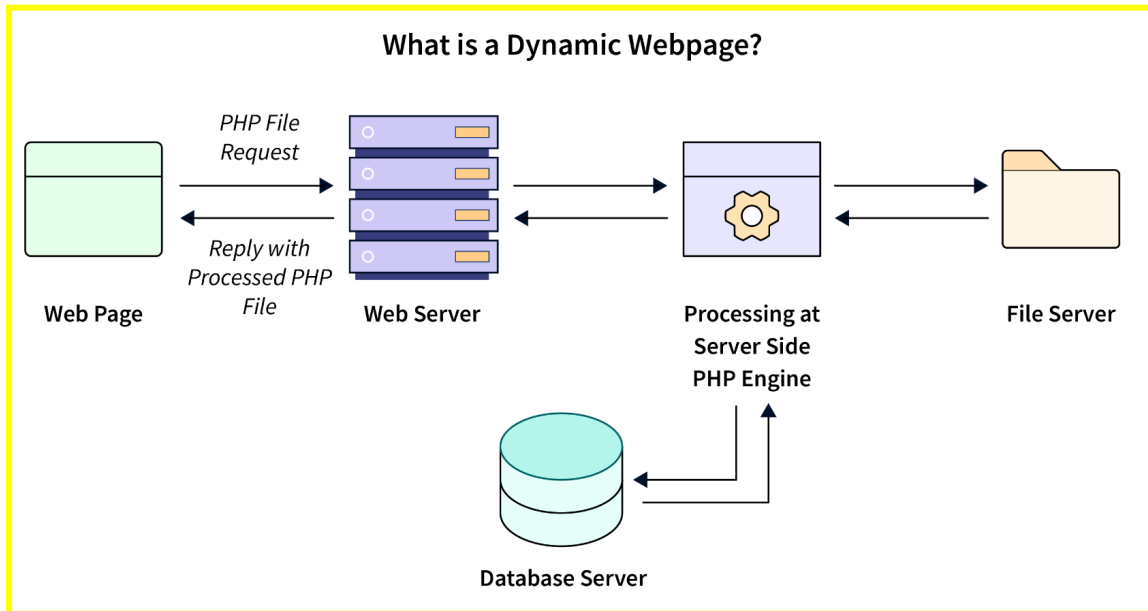
```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dynamic Content with PHP</title>
</head>
<body>
    <h1>Welcome, <?php echo $_GET['name']; ?>!</h1>
</body>
</html>

```

## 5. Illustration - Building Dynamic Web Pages with PHP:

[Insert Image: Illustration demonstrating the process of building dynamic web pages with PHP]



### Practice Example:

#### 1. Dynamic Content Generation:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dynamic Web Page Example</title>
</head>
<body>
  <h1>Welcome <?php echo "User"; ?></h1>
  <p>Today's date is <?php echo date("Y-m-d"); ?></p>
</body>
</html>
```

#### 2. Using PHP in HTML Forms:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>PHP Form Handling Example</title>
</head>
<body>
<h2>PHP Form Handling Example</h2>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
  <label for="name">Name:</label>
  <input type="text" id="name" name="name">
  <input type="submit" value="Submit">
</form>
<?php
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST["name"];
    echo "<p>Hello, $name!</p>";
  }
?>
</body>
</html>

```

### 3. Dynamic Content from Database:

```

<?php
  // Connect to database
  $servername = "localhost";
  $username = "username";
  $password = "password";
  $dbname = "myDB";

  $conn = new mysqli($servername, $username, $password, $dbname);

  if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
  }

  // Query database
  $sql = "SELECT id, firstname, lastname FROM MyGuests";
  $result = $conn->query($sql);

  if ($result->num_rows > 0) {
    // Output data of each row
    while ($row = $result->fetch_assoc()) {

```

```
        echo "ID: " . $row["id"]. " - Name: " . $row["firstname"]. " " .  
$row["lastname"]. "<br>";  
    }  
} else {  
    echo "0 results";  
}  
  
$conn->close();  
?>
```

---

## 11-12: PHP Control Structures and Functions > **Conditional Statements and Loops in PHP**

In this chapter, we'll dive into conditional statements and loops in PHP, which are essential for controlling the flow of execution and repeating tasks in a program. We'll provide code examples and illustrations to help you understand these concepts effectively.

### 1. Conditional Statements

Conditional statements allow developers to execute different blocks of code based on specified conditions. PHP supports several types of conditional statements, including `if`, `else`, `elseif`, and the `switch` statement.

**Example:**

```
<?php
    $age = 25;

    if ($age < 18) {
        echo "You are a minor.";
    } elseif ($age >= 18 && $age < 65) {
        echo "You are an adult.";
    } else {
        echo "You are a senior citizen.";
    }

?>
```

## 2. Loops

Loops are used to repeat a block of code multiple times until a specified condition is met. PHP supports several types of loops, including `for`, `while`, `do-while`, **and** `foreach`.

### Example:

```
<?php
    // Using a for loop to display numbers from 1 to 5
    for ($i = 1; $i <= 5; $i++) {
        echo $i . "<br>";
    }

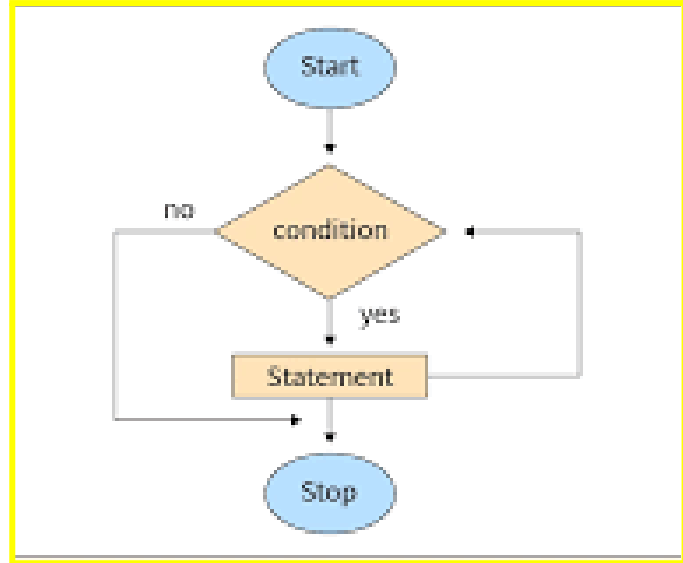
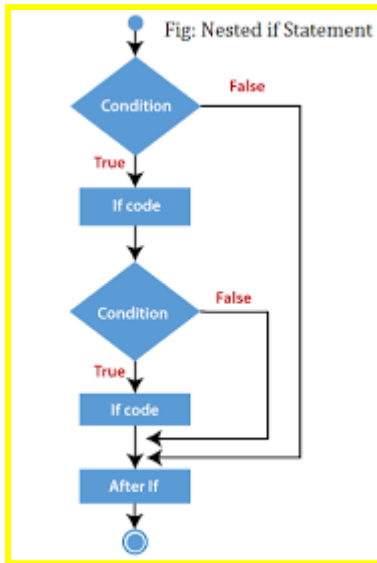
    // Using a while loop to display numbers from 1 to 5
    $i = 1;
    while ($i <= 5) {
        echo $i . "<br>";
        $i++;
    }

    // Using a do-while loop to display numbers from 1 to 5
    $i = 1;
    do {
        echo $i . "<br>";
        $i++;
    } while ($i <= 5);

    // Using a foreach loop to iterate over an array
    $colors = array("red", "green", "blue");
    foreach ($colors as $color) {
        echo $color . "<br>";
    }
?>
```

### 3. Illustration - Conditional Statements and Loops in PHP:

[Insert Image: Illustration demonstrating conditional statements and loops in PHP]



**Practice Example:**

**1. If-Else Statement:**

```

<?php
$num = 10;

if ($num > 0) {
    echo "$num is positive.";
} elseif ($num < 0) {
    echo "$num is negative.";
} else {
    echo "$num is zero.";
}
?>
  
```

**2. While Loop:**

```

<?php
$i = 1;

while ($i <= 5) {
    echo "Number: $i <br>";
    $i++;
}
?>
  
```

**3. For Loop:**

```
<?php
    for ($i = 1; $i <= 5; $i++) {
        echo "Number: $i <br>";
    }
?>
```

#### 4. Foreach Loop with Array:

```
<?php
    $fruits = array("apple", "banana", "orange", "mango");

    foreach ($fruits as $fruit) {
        echo "Fruit: $fruit <br>";
    }
?>
```

## 11-12: PHP Control Structures and Functions > **Creating Custom Functions and Modularizing Code**

In this chapter, we'll explore how to create custom functions in PHP to modularize code and improve code reusability. We'll provide code examples and illustrations to demonstrate the concepts effectively.

### 1. Introduction to Custom Functions

Custom functions in PHP allow developers to encapsulate blocks of code into reusable components. By defining functions, developers can modularize their code, improve readability, and avoid repetition.

Example:



```
<?php
    // Defining a custom function
    function greet($name) {
        echo "Hello, $name!";
    }

    // Calling the custom function
    greet("John");
?>
```

## 2. Function Parameters and Return Values

Functions in PHP can accept parameters, which are variables passed to the function, and can also return values. Parameters allow functions to operate on different data inputs, while return values allow functions to produce outputs.

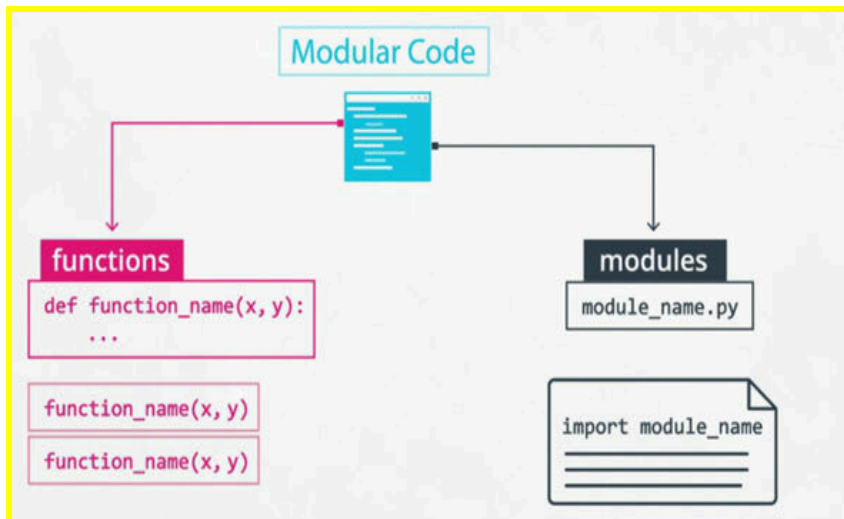
Example:

```
<?php
    // Function with parameters and return value
    function add($a, $b) {
        return $a + $b;
    }

    // Calling the function and storing the return value
    $result = add(3, 5);
    echo "Result: $result"; // Output: Result: 8
?>
```

## 3. Illustration - Creating Custom Functions and Modularizing Code:

[Insert Image: Illustration demonstrating the creation of custom functions and modularizing code]



#### 4. Benefits of Using Functions

- **Code Reusability:** Functions allow developers to reuse code across multiple parts of an application.
- **Modularization:** Functions help break down complex tasks into smaller, manageable components.
- **Readability:** Well-named functions make code more readable and understandable.

#### 5. Best Practices for Creating Functions

- **Descriptive Names:** Choose descriptive names for functions that clearly indicate their purpose.
- **Single Responsibility:** Keep functions focused on a single task to improve maintainability.
- **Avoid Side Effects:** Minimise side effects within functions to improve predictability and debugging.

#### Practice Example:

##### 1. Simple Function:

```
<?php
// Function to greet a user
function greet($name) {
    echo "Hello, $name!";
}

// Call the function
```

```
    greet("John");  
?>
```

## 2. Function with Parameters and Return Value:

```
<?php  
    // Function to calculate the sum of two numbers  
    function add($num1, $num2) {  
        return $num1 + $num2;  
    }  
  
    // Call the function and store the result in a variable  
    $result = add(5, 3);  
    echo "Sum: $result";  
?>
```

## 3. Modularizing Code:

```
<?php  
    // Include external PHP file  
    include 'functions.php';  
  
    // Call function from the included file  
    greet("Jane");  
?>
```

### **functions.php:**

```
<?php  
    // Function to greet a user  
    function greet($name) {  
        echo "Hello, $name!";  
    }  
?>
```

---

## 13-14: Working with Forms and Form Handling in PHP > **Building HTML Forms**


In this chapter, we'll explore how to build HTML forms in conjunction with PHP for handling user input. We'll provide code examples and illustrations to guide you through the process effectively.

## 1. Introduction to HTML Forms

HTML forms are essential for collecting user input on web pages. They allow users to enter data, which can then be submitted to a server for processing. HTML forms consist of various form elements such as text fields, checkboxes, radio buttons, dropdown lists, and buttons.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>HTML Form Example</title>
</head>
<body>
  <form action="process.php" method="post">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username">
    <br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password">
    <br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```



## 2. Explanation of HTML Form Elements

- **Form Tag** (<form>): Defines a form that collects user input.
- **Input Tag** (<input>): Defines an input field that allows users to enter data.
- **Label Tag** (<label>): Defines a label for an input field to provide a description or instruction.
- **Action Attribute**: Specifies the URL where the form data should be submitted.
- **Method Attribute**: Specifies the HTTP method (GET or POST) used to submit the form data.

## 3. Illustration - Building HTML Forms:

[Insert Image: Illustration demonstrating the structure of an HTML form]

#### 4. Handling Form Submission in PHP

After the user submits a form, the form data is sent to a PHP script for processing. In the PHP script, the form data can be accessed using the `$_POST` or `$_GET` superglobals, depending on the method used in the form.

Example:

```
<?php
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Process the form data (e.g., validate, authenticate)
?>
```

#### 5. Security Considerations

When working with forms and handling user input, it's essential to consider security measures to prevent vulnerabilities such as SQL injection and cross-site scripting (XSS). Sanitise and validate user input before using it in your application to ensure data integrity and protect against malicious attacks.

**Practice Example:**

##### 1. Simple Form with Input Fields:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple Form Example</title>
</head>
<body>
    <h2>Simple Form Example</h2>
    <form method="post" action="process_form.php">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name"><br><br>
```

```
<label for="email">Email:</label>
<input type="email" id="email" name="email"><br><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

## 2. Form with Textarea and Radio Buttons:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Form with Textarea and Radio Buttons</title>
</head>
<body>
  <h2>Form with Textarea and Radio Buttons</h2>
  <form method="post" action="process_form.php">
    <label for="message">Message:</label><br>
    <textarea id="message" name="message" rows="4"
cols="50"></textarea><br><br>
    <label>Gender:</label><br>
    <input type="radio" id="male" name="gender" value="male">
    <label for="male">Male</label><br>
    <input type="radio" id="female" name="gender" value="female">
    <label for="female">Female</label><br><br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

## 3. Form with Select Dropdown and Checkbox:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Form with Select Dropdown and Checkbox</title>
</head>
<body>
```

```
<h2>Form with Select Dropdown and Checkbox</h2>
<form method="post" action="process_form.php">
  <label for="country">Select Country:</label><br>
  <select id="country" name="country">
    <option value="USA">USA</option>
    <option value="UK">UK</option>
    <option value="Canada">Canada</option>
  </select><br><br>
  <input type="checkbox" id="subscribe" name="subscribe" value="yes">
  <label for="subscribe">Subscribe to Newsletter</label><br><br>
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

## 13-14: Working with Forms and Form Handling in PHP > **Processing Form Data with PHP**

In this chapter, we'll delve into how to process form data using PHP after it has been submitted by the user. We'll provide code examples and illustrations to guide you through the process effectively.

### 1. Introduction to Processing Form Data

After a user submits a form on a web page, the form data is sent to a server for processing. PHP is commonly used on the server-side to handle form data. Processing form data involves retrieving the submitted data, validating it, performing any necessary operations, and providing feedback to the user.

### 2. Retrieving Form Data in PHP

Form data can be retrieved in PHP using the `$_POST` or `$_GET` superglobals, depending on the form's method attribute (`POST` or `GET`).

Example:

```

<?php
    // Retrieve form data submitted via POST method
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Process the form data
    // (e.g., validate, authenticate, store in database)
?>

```

### 3. Form Validation

Form validation is crucial to ensure that the submitted data meets certain criteria or constraints. This helps maintain data integrity and security. Common validation tasks include checking for required fields, validating email addresses, and verifying the format of input data.

Example:

```

<?php
    // Validate username field
    if (empty($_POST['username'])) {
        $errors[] = "Username is required";
    }

    // Validate email field
    if (!filter_var($_POST['email'], FILTER_VALIDATE_EMAIL)) {
        $errors[] = "Invalid email address";
    }

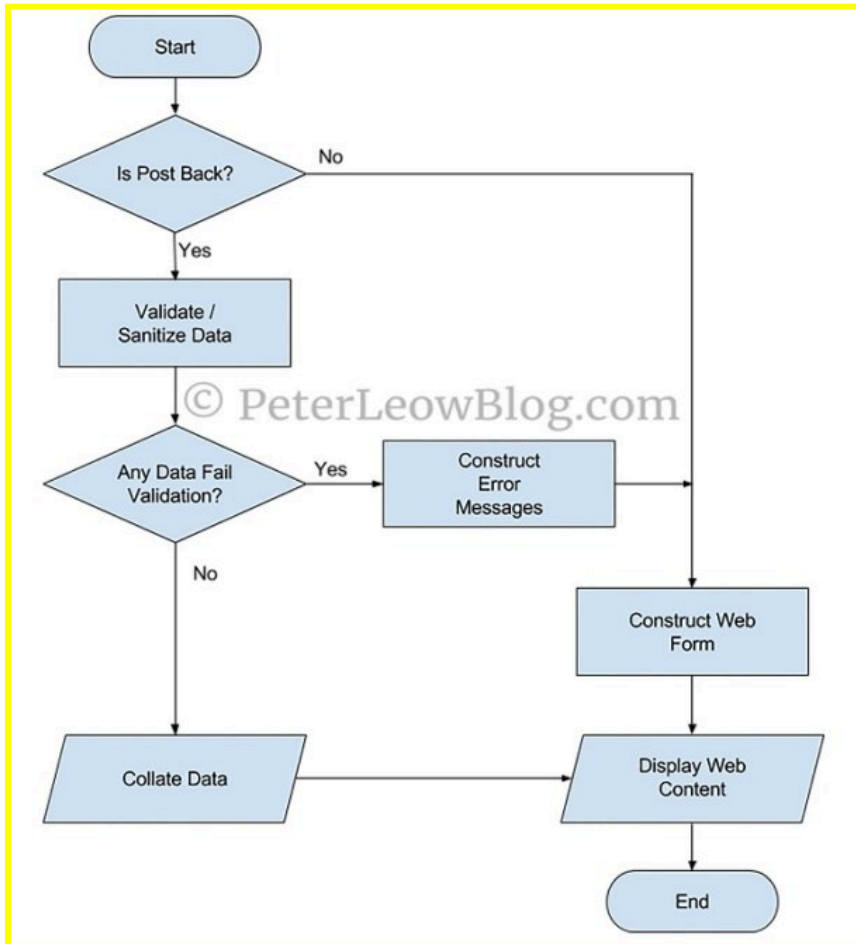
    // Check for any validation errors
    if (!empty($errors)) {
        // Display error messages to the user
        foreach ($errors as $error) {
            echo $error . "<br>";
        }
    } else {
        // Proceed with processing the form data
    }
?>

```



#### 4. Illustration - Processing Form Data with PHP:

[Insert Image: Illustration demonstrating the process of processing form data with PHP]



#### 5. Sanitising Form Data

Sanitising form data involves removing any potentially malicious or unwanted characters from the input data to prevent security vulnerabilities such as SQL injection and cross-site scripting (XSS) attacks. Common sanitization techniques include using functions like `htmlspecialchars()` and `mysqli_real_escape_string()`.

Example:

```
<?php
    // Sanitize form data
    $username = htmlspecialchars($_POST['username']);
    $password = mysqli_real_escape_string($connection, $_POST['password']);

    // Proceed with processing the sanitized data
?>
```

## Practice Example:

### 1. Processing Text Input:

HTML form:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Text Input Form</title>
</head>
<body>
    <h2>Text Input Form</h2>
    <form method="post" action="process_text.php">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name"><br><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

PHP script (process\_text.php):

```
<?php
    if ($_SERVER["REQUEST_METHOD"] == "POST") {
        $name = $_POST["name"];
        echo "Hello, $name!";
    }
?>
```

### 2. Processing Checkbox Input:

## HTML form:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Checkbox Form</title>
</head>
<body>
  <h2>Checkbox Form</h2>
  <form method="post" action="process_checkbox.php">
    <input type="checkbox" id="subscribe" name="subscribe" value="yes">
    <label for="subscribe">Subscribe to Newsletter</label><br><br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

## PHP script (process\_checkbox.php):

```
<?php
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (isset($_POST["subscribe"]) && $_POST["subscribe"] == "yes") {
      echo "Thank you for subscribing!";
    } else {
      echo "You did not subscribe.";
    }
  }
?>
```

## 3. Processing Select Dropdown Input:

### HTML form:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Select Dropdown Form</title>
</head>
```

```
<body>
  <h2>Select Dropdown Form</h2>
  <form method="post" action="process_dropdown.php">
    <label for="country">Select Country:</label><br>
    <select id="country" name="country">
      <option value="USA">USA</option>
      <option value="UK">UK</option>
      <option value="Canada">Canada</option>
    </select><br><br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

PHP script (process\_dropdown.php):

```
<?php
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $country = $_POST["country"];
    echo "You selected $country.";
  }
?>
```

## 13-14: Working with Forms and Form Handling in PHP > **Form Validation and Security Measures**

In this chapter, we'll explore the importance of form validation and security measures when handling user input in PHP. We'll provide code examples and illustrations to demonstrate various techniques for ensuring data integrity and preventing security vulnerabilities.

### 1. Introduction to Form Validation

Form validation is the process of ensuring that the data submitted through a form meets certain criteria or constraints. It helps maintain data integrity, improves user experience, and prevents security vulnerabilities such as SQL injection and cross-site scripting (XSS) attacks.

### 2. Common Form Validation Tasks

- **Required Fields:** Check if required fields are not empty.

- **Email Validation:** Verify the format of email addresses.
- **Numeric Validation:** Ensure that numeric input is valid.
- **Length Validation:** Check the length of input data.
- **Regular Expressions:** Use regular expressions to validate complex patterns (e.g., phone numbers, ZIP codes).

Example:

```
<?php
// Validate email field
if (empty($_POST['email'])) {
    $errors[] = "Email is required";
} elseif (!filter_var($_POST['email'], FILTER_VALIDATE_EMAIL)) {
    $errors[] = "Invalid email address";
}

// Validate password field
if (strlen($_POST['password']) < 8) {
    $errors[] = "Password must be at least 8 characters long";
}

// Check for any validation errors
if (!empty($errors)) {
    // Display error messages to the user
    foreach ($errors as $error) {
        echo $error . "<br>";
    }
} else {
    // Proceed with processing the form data
}
?>
```

### 3. Sanitising Form Data

In addition to validation, it's crucial to sanitise form data to prevent security vulnerabilities. Sanitization involves removing potentially malicious or unwanted characters from the input data.

Example:

```

<?php
    // Sanitize form data
    $username = htmlspecialchars($_POST['username']);
    $password = mysqli_real_escape_string($connection, $_POST['password']);

    // Proceed with processing the sanitized data
?>

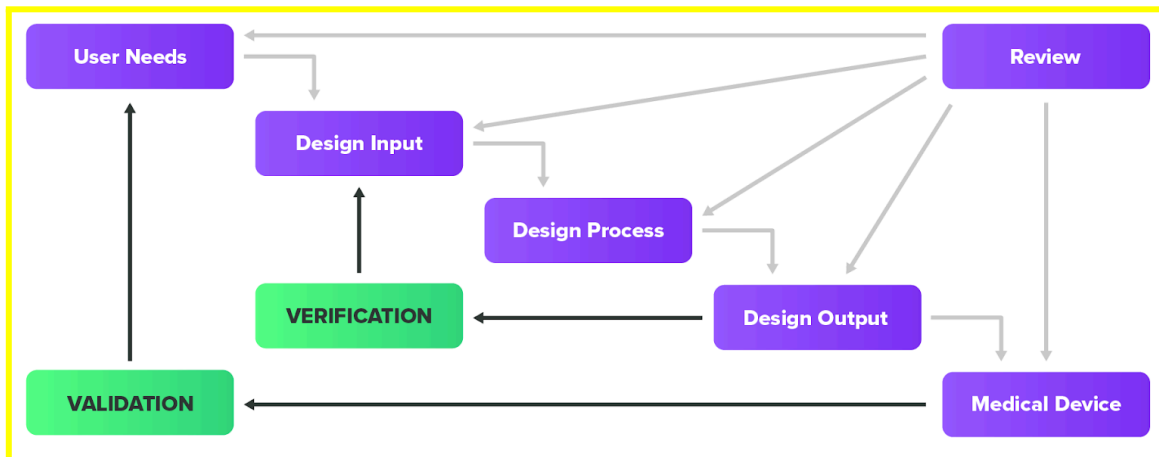
```

#### 4. Security Measures

- Preventing SQL Injection: Use prepared statements or parameterized queries to prevent SQL injection attacks.
- Cross-Site Scripting (XSS) Prevention: Escape output data using functions like `htmlspecialchars()` to prevent XSS attacks.
- CSRF Protection: Implement measures such as CSRF tokens to prevent Cross-Site Request Forgery attacks.

#### 5. Illustration - Form Validation and Security Measures:

[Insert Image: Illustration demonstrating form validation and security measures]



### 15-16: Introduction to MySQL Database > **Understanding Relational Databases**

In this chapter, we'll explore the concept of relational databases, with a focus on MySQL. We'll provide code examples and illustrations to help you understand the fundamentals effectively.

## 1. Introduction to Relational Databases

Relational databases are a type of database that organizes data into tables, where each table consists of rows and columns. The relationships between tables are established through keys, such as primary keys and foreign keys.

## 2. Key Concepts of Relational Databases

- **Tables:** The basic structure for storing data in a relational database.
- **Rows:** Each row in a table represents a record or entry.
- **Columns:** Columns represent attributes or fields of the data.
- **Primary Key:** A unique identifier for each row in a table.
- **Foreign Key:** A key that establishes a link between two tables.

## 3. Example of Relational Database Structure

Consider a simple relational database for managing employees and departments. We can have two tables: `employees` and `departments`.

### Employees Table:

id	name	department_id
1	Alice	1
2	Bob	2
3	Charlie	1

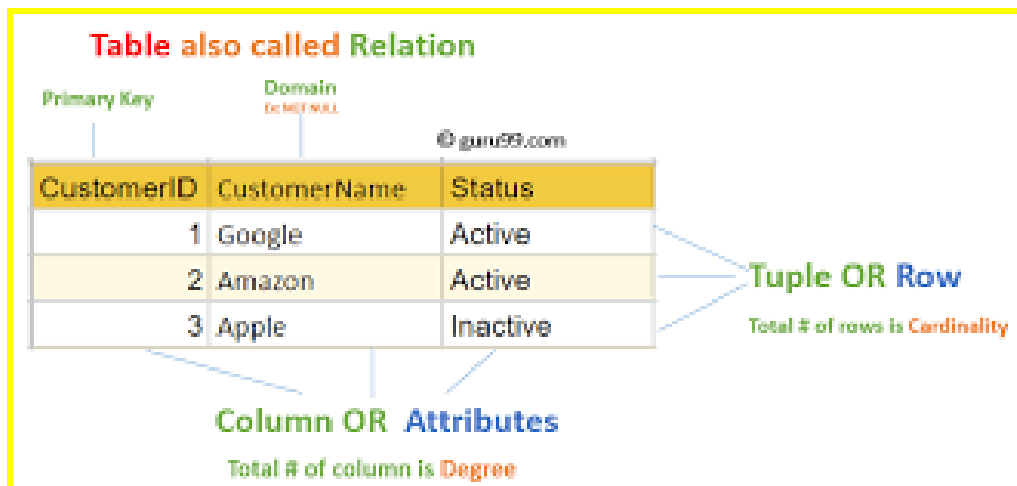
### Departments Table:

id	name
1	Engineering
2	Sales

In this example, the `employees` table has a foreign key `department_id` that references the `id` column in the `departments` table.

#### 4. Illustration - Relational Database Structure:

[Insert Image: Illustration demonstrating the structure of a relational database with tables, rows, and columns]



#### 5. Benefits of Relational Databases

- Data Integrity: Relational databases enforce data integrity through constraints such as primary keys and foreign keys.
- Flexibility: Tables can be related to each other, allowing complex data relationships.
- Querying: SQL (Structured Query Language) is used to query and manipulate data in relational databases, providing a powerful and standardised way to interact with data.

### 15-16: Introduction to MySQL Database > Basics of SQL: Creating Tables, Inserting Data, and Querying Data

In this chapter, we'll cover the fundamentals of SQL (Structured Query Language) with a focus on creating tables, inserting data, and querying data in MySQL. We'll provide code examples and illustrations to guide you through the process effectively.

#### 1. Introduction to SQL



SQL is a standard language for interacting with relational databases. It allows users to perform various operations such as creating tables, inserting, updating, and deleting data, and querying data for analysis and reporting.

## 2. Creating Tables in MySQL

In MySQL, tables are created using the `CREATE TABLE` statement. Each table consists of columns, each with a specific data type, and optionally, constraints such as primary keys and foreign keys.

Example:

```
CREATE TABLE users (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  username VARCHAR(50) NOT NULL,  
  email VARCHAR(100) NOT NULL UNIQUE,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

## 3. Inserting Data into Tables

Once a table is created, data can be inserted into it using the `INSERT INTO` statement. Values for each column are provided in the order in which they appear in the table definition.

Example:

```
INSERT INTO users (username, email) VALUES ('john_doe', 'john@example.com');  
INSERT INTO users (username, email) VALUES ('jane_smith', 'jane@example.com');
```

## 4. Querying Data from Tables

To retrieve data from a table, we use the `SELECT` statement. We can specify which columns to retrieve, filter the results based on conditions, and order the results as needed.

Example:

```
SELECT * FROM users;
```

## 5. Illustration - Basics of SQL:

[Insert Image: Illustration demonstrating the process of creating tables, inserting data, and querying data in MySQL]

```
mysql> INSERT INTO movies VALUE ("The Empire Strikes Back", "epic space opera", "Irvin Kershner", 1980);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM movies;
```

title	genre	director	release_year
Joker	psychological thriller	Todd Phillips	2019
The Empire Strikes Back	epic space opera	Irvin Kershner	1980

```
2 rows in set (0.00 sec)
```

## Practice Example:

### 1. Creating Tables:

```
CREATE TABLE users (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  username VARCHAR(50) NOT NULL,  
  email VARCHAR(100) NOT NULL UNIQUE,  
  password VARCHAR(255) NOT NULL  
);
```

```
CREATE TABLE posts (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  title VARCHAR(100) NOT NULL,  
  content TEXT,  
  user_id INT,  
  FOREIGN KEY (user_id) REFERENCES users(id)  
);
```

### 2. Inserting Data:

```
INSERT INTO users (username, email, password) VALUES ('john_doe',  
'john@example.com', 'password123');
```

```
INSERT INTO posts (title, content, user_id) VALUES ('First Post', 'This is the content of the first post.', 1);
```

### 3. Querying Data:

```
SELECT * FROM users;
```

```
SELECT username, email FROM users WHERE id = 1;
```

```
SELECT posts.title, posts.content, users.username FROM posts  
JOIN users ON posts.user_id = users.id;
```

---

## 17-18: Connecting PHP to MySQL >Establishing Database Connections in PHP

In this chapter, we'll explore how to establish database connections between PHP and MySQL, which is essential for interacting with MySQL databases from PHP scripts. We'll provide code examples and illustrations to guide you through the process effectively.

### 1. Introduction to Database Connections

Database connections enable PHP scripts to communicate with MySQL databases, allowing data retrieval, manipulation, and storage. Establishing a database connection involves providing connection parameters such as hostname, username, password, and database name.

### 2. Connecting to MySQL Database in PHP

PHP provides several functions for connecting to MySQL databases. The most commonly used function is `mysqli_connect()`.

Example:

```
<?php
    // Database connection parameters
    $hostname = "localhost";
    $username = "username";
    $password = "password";
    $database = "dbname";

    // Establishing a database connection
    $connection = mysqli_connect($hostname, $username, $password, $database);

    // Check connection
    if (!$connection) {
        die("Connection failed: " . mysqli_connect_error());
    } else {
        echo "Connected successfully";
    }
?>
```

### 3. Closing Database Connection

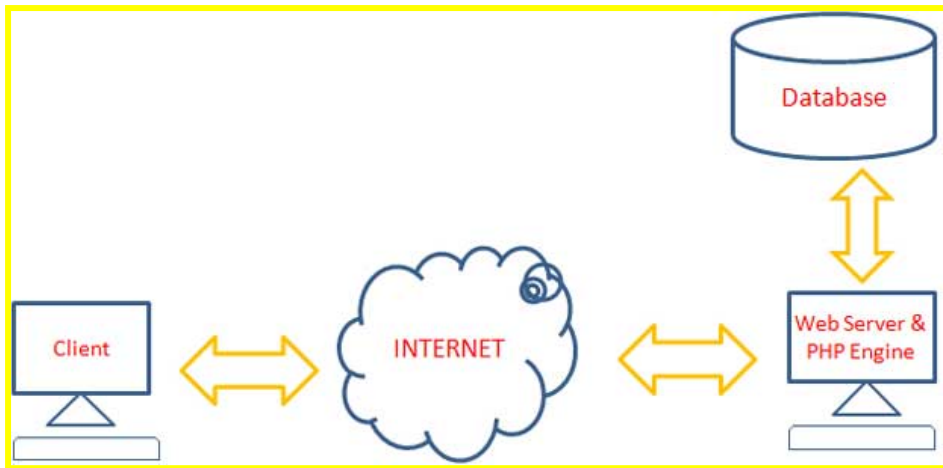
After executing database operations, it's essential to close the database connection to free up resources and prevent potential security risks.

Example:

```
<?php
    // Closing the database connection
    mysqli_close($connection);
?>
```

### 4. Illustration - Establishing Database Connections in PHP:

[Insert Image: Illustration demonstrating the process of establishing database connections in PHP]



## 5. Best Practices

- Use Secure Credentials: Avoid hardcoding database credentials directly into PHP scripts. Instead, store them in a separate configuration file outside the web root directory.
- Error Handling: Implement error handling mechanisms to gracefully handle database connection errors and failures.
- Connection Pooling: Consider using connection pooling to improve the performance and scalability of database connections in high-traffic applications.

## 17-18: Connecting PHP to MySQL > Performing CRUD Operations (Create, Read, Update, Delete) with PHP and MySQL

In this chapter, we'll delve into performing CRUD (Create, Read, Update, Delete) operations with PHP and MySQL, essential for interacting with databases effectively. We'll provide comprehensive code examples and illustrations to guide you through each operation seamlessly.

### 1. Introduction to CRUD Operations

CRUD operations form the cornerstone of database interaction. They facilitate creating, reading, updating, and deleting data within a database. Understanding and implementing these operations are vital for managing data effectively in PHP applications.

### 2. Performing CRUD Operations in PHP

### a. Create Operation (INSERT)

To add new records to a database, we use the `INSERT INTO` SQL statement. This operation involves specifying the table and values to be inserted.

Example:

```
<?php
// Database connection
$connection = mysqli_connect("localhost", "username", "password", "database");

// Insert data into a table
$sql = "INSERT INTO users (username, email) VALUES ('john_doe', 'john@example.com)";
mysqli_query($connection, $sql);
?>
```

### b. Read Operation (SELECT)

Retrieving records from a database involves using the `SELECT` SQL statement. This operation fetches data based on specified criteria and conditions.

Example:

```
<?php
// Database connection
$connection = mysqli_connect("localhost", "username", "password", "database");

// Retrieve data from a table
$sql = "SELECT * FROM users";
$result = mysqli_query($connection, $sql);

// Fetch and display data
while ($row = mysqli_fetch_assoc($result)) {
    echo "Username: " . $row['username'] . ", Email: " . $row['email'] . "<br>";
}
?>
```

### c. Update Operation (UPDATE)

Modifying existing records in a database involves using the `UPDATE` SQL statement. This operation updates data based on specified conditions.

Example:

```

<?php
    // Database connection
    $connection = mysqli_connect("localhost", "username", "password", "database");

    // Update data in a table
    $sql = "UPDATE users SET email = 'new_email@example.com' WHERE username = 'john_doe'";
    mysqli_query($connection, $sql);
?>

```

#### d. Delete Operation (DELETE)

Removing records from a database entails using the `DELETE` SQL statement. This operation deletes data based on specified conditions.

Example:

```

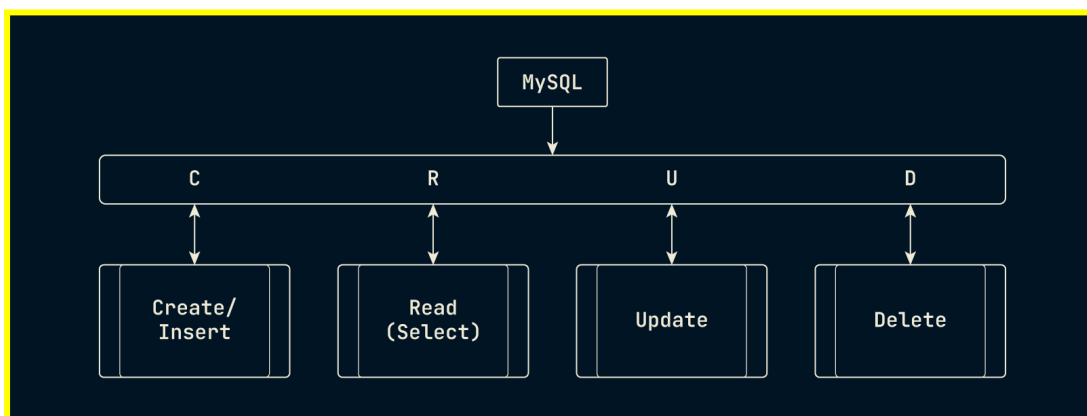
<?php
    // Database connection
    $connection = mysqli_connect("localhost", "username", "password", "database");

    // Delete data from a table
    $sql = "DELETE FROM users WHERE username = 'john_doe'";
    mysqli_query($connection, $sql);
?>

```

### 3. Illustration - Performing CRUD Operations with PHP and MySQL:

[Insert Image: Illustration demonstrating the process of performing CRUD operations with PHP and MySQL]



---

**19-20: Final Project and Review >Capstone Project: Design and Develop a Dynamic Web Application**

**19-20: Final Project and Review >Review of Course Concepts and Skills**

**19-20: Final Project and Review >Presenting and Showcasing Final Projects**

---

---