Module 6 – HTML, CSS and JS in PHP

1. HTML Basics

1. **What is HTML? Explain its structure.**

   HTML (Hypertext Mark-up Language) is the standard language used to create and design the **structure of web pages**. It uses a system of tags to define elements like headings, paragraphs, links, images, and more.

   Structure of HTML:

```
<! DOCTYPE html>        <! -- Declaration of HTML version -->
<Html>                  <! -- Root element -->
  <Head>                <! -- Metadata about the document -->
    <Title>My Website</title>
  </head>
  <Body>                <! -- Visible content goes here -->
    <h1>Welcome! </h1>
    <p>this is a simple HTML page. </p>
  </body>
</html>
```

- **<! DOCTYPE html>** – Declares the document type (HTML5).
- **<Html>** – Root of the document.
- **<Head>** – Contains Meta info (title, styles, scripts).
- **<Body>** – Holds the content displayed on the webpage.

2. **Describe the purpose of HTML tags and provide examples of commonly used tags.**

HTML tags are used to define elements within a webpage. They usually come in pairs: an opening tag <tag> and a closing tag </tag>.

Purpose:

- Structure content
- Define meaning (e.g., heading vs paragraph)
- Allow browsers to interpret content appropriately

Common HTML Tags:

Tag	Purpose	Example
<h1>–<h6>	Headings (from biggest to smallest)	Main Title
	Paragraph	This is a paragraph.
<a>	Hyperlink	Visit
<img>	Image	<img src="image.jpg" alt="Image">

Tag	Purpose	Example
,	Unordered list and list items	Item 1
	Container (block-level)	Some content
	Inline container	Inline text

3. **What are the differences between block-level and Inline elements? Give examples of each.**

Block-level vs Inline Elements

▯ Block-level Elements

- Take up the **full width** available
- Start on a **new line**
- Used for larger structures

Examples:

-
- <p>
- <h1> to <h6>
- ,
- <section>, <article>

🔤 Inline Elements

- Take up only as much width as necessary
- Do **not** start on a new line
- Used for smaller pieces of content inside blocks

Examples:

-
- <a>
- <img>
- ,

4. **Explain the concept of semantic HTML and why it is important.**

Semantic HTML refers to the use of HTML tags that clearly describe the meaning of the content they enclose.

Instead of using for everything, semantic tags provide **context** and **improve accessibility**, SEO, and maintainability.

Examples of Semantic Tags:

- <header> – Defines a page or section header
- <nav> – Contains navigation links
- <main> – Main content of the page
- <article> – A self-contained piece of content
- <footer> – Footer of the section/page
- <section> – Groups related content

Why is it important?

- **Improves accessibility** for screen readers
- **Enhances SEO** (search engines understand content better)
- **Easier to read and maintain** code

2. CSS Fundamentals

1. What is CSS? How does it differ from HTML?

CSS (Cascading Style Sheets) is a language used to **style and layout HTML elements** on a web page—things like colors, fonts, spacing, positioning, and responsiveness.

Key Differences between HTML and CSS:

HTML	CSS
Defines **structure & content**	Defines **style & appearance**
Uses **tags/elements**	Uses **rules and selectors**
E.g., Hello	E.g., p { color: red; }
Tells what content is	Tells how content looks

2. Explain the three ways to apply CSS to a web page.

Three Ways to Apply CSS to a Web Page

Inline CSS

- Added directly to HTML elements using the style attribute.
- Not reusable, best for quick styles.

Html

```
<p style="color: blue ;"> this is blue text. </p>
```

Internal CSS

- Written in a <style> tag inside the <head> of the HTML document.

```
Html
<Head>
  <Style>
    P {
      Font-size: 18px;
    }
  </style>
</head>
```

External CSS

- Written in a separate .css file and linked using <link>.

```
Html
<link rel="stylesheet" href="styles.css">

Css
/* styles.css */
h1 {
  Color: green;
}
```

External CSS is the best practice for larger or multi-page websites.

3. What are CSS selectors? List and describe the different types of selectors.

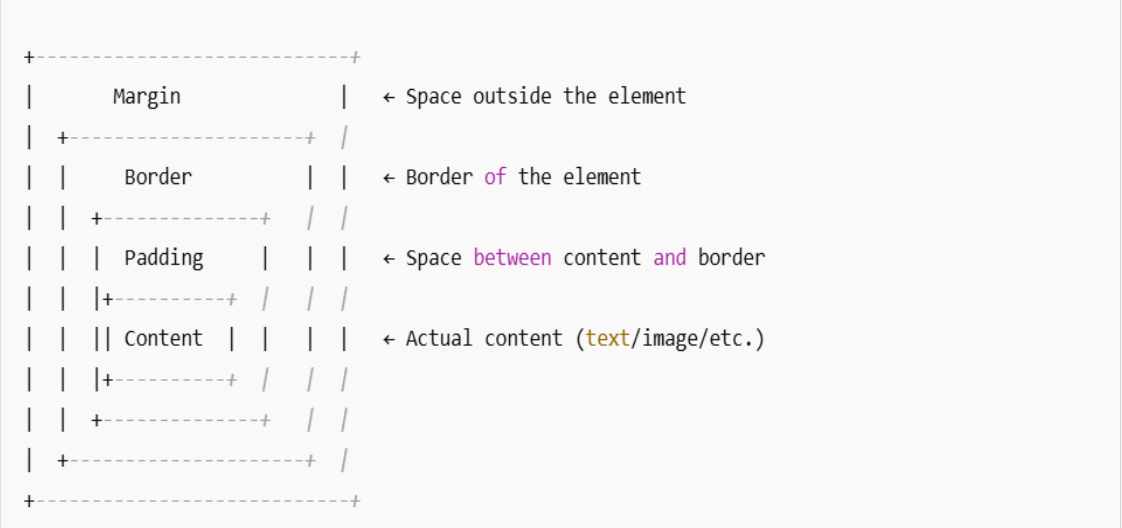
Types of CSS Selectors:

Selector Type	Example	Description
Universal	*	Selects all elements
Element	p	Selects all <p> elements
Class	.box	Selects elements with class "box"
ID	#main	Selects element with id "main"
Group	h1, p	Selects multiple types of elements
Descendant	div p	Selects <p> inside
Child	ul > li	Selects direct child of
Attribute	input[type="text"]	Selects input fields of type text
Pseudo-class	a:hover	Selects links on hover
Pseudo-element	p::first-line	Styles the first line of <p> text

4. What is the box model in CSS? Explain its components.

The **CSS Box Model** is a fundamental concept that describes how elements are **structured and spaced** on a page.

Each element is a box made up of:



Components:

- **Content:** The actual element (text, image, etc.)
- **Padding:** Space inside the border (around content)
- **Border:** Edge around the padding and content
- **Margin:** Space outside the element.

3. Responsive Web Design

1. **What is responsive web design? Why is it important?**

Responsive Web Design (RWD) is a design approach that ensures a website **adapts to different screen sizes and devices**—desktops, tablets, smartphones, etc.

Key Features:

- Flexible layouts
- Scalable images
- Dynamic content reflow using **CSS media queries**

- **Why It's Important:**

- **Improves user experience** on all devices
- Helps reach a **wider audience**
- Boosts **SEO performance** (Google prefers mobile-friendly sites)
- Reduces need for multiple codebases (no need for separate mobile and desktop sites)

2. **Explain the use of media queries in CSS. Provide an example.**

Media queries are CSS rules that apply styles **based on screen size, resolution, or device type**.

They allow you to **change styles dynamically** depending on the user's device.

Syntax:

@media (max-width: 600px) {

```
  Body {
    Background-color: lightgray;
  }
}
```

Explanation:

- This rule says: “If the screen is **600px wide or less**, apply these styles.”
- In this case, the page background will turn **light gray** on smaller screens (like phones).

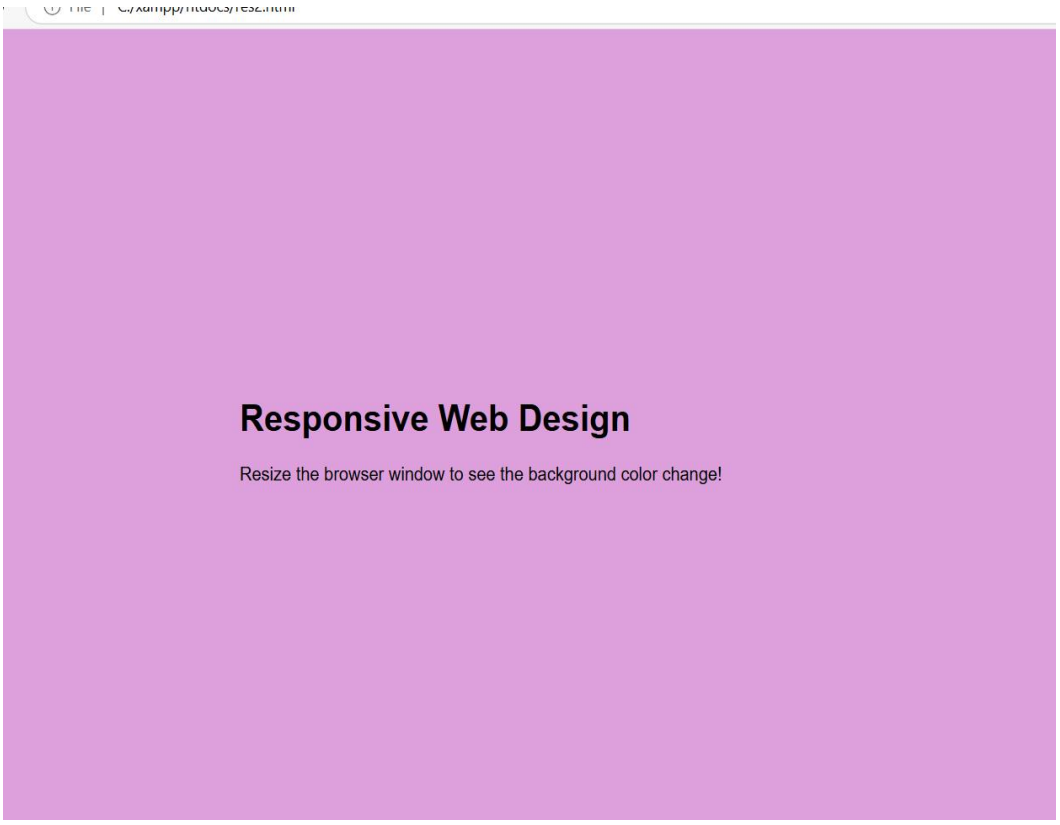
Example:

res2.html > html > head > style

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Responsive Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: plum;
      margin: 100px;
      padding: 200px;
    }

  </style>
</head>
<body>
  <h1>Responsive Web Design</h1>
  <p>Resize the browser window to see the background color change!</p>
</body>
</html>

```



4. **What are the benefits of using a mobile-first approach in web design?**

Benefits of Mobile-First Design Approach:

A **mobile-first** approach means designing for **small screens first**, then scaling up for larger screens.

Why Use It:

Benefit	Description
Performance	Smaller screens = less content = faster load times
Better UX	Prioritizes touch-friendly and readable interfaces

Benefit	Description
Progressive Enhancement	Builds up features for larger devices step by step
Future-proofing	Mobile users dominate the web; ensures accessibility

---

In Summary:

- Responsive Design = fluid, flexible, adaptive layouts
- Media Queries = apply CSS rules based on device/screen conditions
- Mobile-First = start small, scale up — leads to better performance and UX

4. PHP Integration

1. **How can PHP be used to dynamically generate HTML content? Provide examples.**

PHP is a server-side scripting language, which means it runs on the server and can generate dynamic content. The main way PHP dynamically generates HTML is by embedding PHP code inside an HTML page. PHP can manipulate variables, process user input, or interact

with databases and generate different HTML output accordingly.

Example:

```
<? Php
$fruits = ["Apple", "Banana", "Cherry", "Date"];
?>
<! DOCTYPE html>
<html Lang="en">
<Head>
   <Meta charset="UTF-8">
   <Title>Fruit List</title>
</head>
<Body>
   <h1>Fruit List</h1>
   <Ul>
      <? Php
      For each ($fruits as $fruit) {
         Echo "<li>$fruit</li>";
      }
      ?>
   </ul>
</body>
</html>
```

In this case, PHP loops through the array `$fruits` and generates a list of items inside an unordered list (`<ul>`).

2. **Explain how to include CSS files in a PHP-generated HTML page.**

You can include a **CSS file** in a PHP-generated HTML page just like you would in a regular HTML page, using the `<link>` tag in the `<head>` section of the page. The only difference is that you will be working within a PHP file.

Example: Including a CSS file in a PHP page

```
<? Php
// PHP script starts here
?>
<! DOCTYPE html>
<html Lang="en">
<Head>
   <Meta charset="UTF-8">
   <Title>My Styled Page</title>
   <! -- Link to an external CSS file -->
   <link rel="stylesheet" href="styles.css">
</head>
<Body>
   <h1>Hello, World! </h1>
   <p>this is a PHP page styled using an external CSS file.
</p>
```

</body>
</html>

In this example, the CSS file (styles.css) is linked using the <link> tag. The CSS file should be in the same folder as the PHP file, or you can specify a path to the file.

3. **What are the advantages of using PHP to manage HTML forms?**

Using PHP to manage HTML forms comes with several advantages. Here are some key benefits:

Advantages:

1. **Dynamic Form Handling**: PHP allows you to **dynamically generate and handle forms** based on user input or other conditions. For example, depending on previous form choices, you can dynamically adjust the fields displayed in the form.
2. **Form Validation**: PHP can validate form data before it's submitted to a database or processed. For example, you can check if a user has entered a valid email address, or if required fields are filled out.

   Example: PHP Form Validation

If ($_SERVER ["REQUEST_METHOD"] == "POST") {

```
$name = $_POST ['name'];
$email = $_POST ['email'];

If (empty ($name) || empty ($email)) {
    Echo "Name and Email are required!";
} else
{
    Echo "Form submitted successfully!”
}
}
```

3. Security: PHP can help ensure **security** by sanitizing form data before processing it. For example, it can prevent **SQL Injection** attacks by using prepared statements with a database.

Example: Sanitizing Input

```
Php
$name = htmlspecialchars (trim($_POST['name'])); // Prevent XSS
```

4. Database Integration: PHP allows forms to interact with a database. After validating and sanitizing the input, you can store form data in a database like MySQL. For example, when a user submits a registration form, PHP can save their information to a database.

5. User Feedback: PHP can provide **real-time feedback** after form submission. For example, it can display a message confirming successful submission or show an error if something went wrong.

6. Session Handling: PHP can maintain user session information. After submitting a form, PHP can redirect users to a confirmation page, or it can save certain form data in a session to use on another page.

Example of PHP Form Handling with Validation:

```
<? Php
If ($_SERVER ["REQUEST_METHOD"] == "POST") {
   $name = htmlspecialchars (trim ($_POST ['name']));
   $email = htmlspecialchars (trim ($_POST ['email']));
   $message = htmlspecialchars (trim ($_POST ['message']));

   If (empty ($name) || empty ($email) || empty ($message)) {
      Echo "All fields are required!”
   } else if (! filter_var ($email, FILTER_VALIDATE_EMAIL)) {
      Echo "Invalid email format!”
   } else {
```

```
        Echo "Thank you, $name. Your message has been sent!”
    }
}
?>

<! DOCTYPE html>
<html Lang="en">
<Head>
    <Meta charset="UTF-8">
    <Title>Contact Form</title>
</head>
<Body>
    <h1>Contact Us</h1>
    <form method="POST">
        <label for="name">Name :< /label>
        <input type="text" name="name" id="name">
        <Br>
        <label for="email">Email :< /label>
        <input type="email" name="email" id="email">
```

```
        <Br>
        <label for="message">Message :< /label>
        <text area name="message" id="message"></text area>
        <Br>
        <input type="submit" value="Send Message">
    </form>
</body>
</html>
```

In this example:

- The form collects name, email, and message.
- PHP processes the form data, validates the input, and provides feedback.