HTML Structure and Elements

1. Introduction to HTML

HTML (HyperText Markup Language) is the standard language for creating web pages. It provides the structure of a webpage using elements defined by tags.

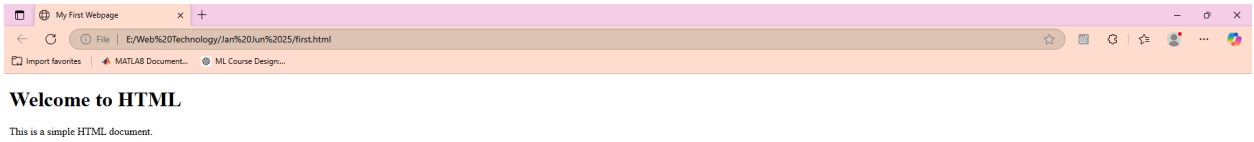
- **Key Features:**
    - Describes the structure of web content.
    - Uses a system of elements, which are building blocks for webpages.
    - Works in conjunction with CSS for styling and JavaScript for interactivity.

2. Basic Structure of an HTML Document

An HTML document starts with a <!DOCTYPE> declaration, followed by the <html> tag, which contains the entire page's content.

Example: 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 Webpage</title>
</head>
<body>
  <h1>Welcome to HTML</h1>
  <p>This is a simple HTML document.</p>
</body>
</html>
```



Explanation of Elements:

1. **<!DOCTYPE html>**: Declares the document type and version of HTML (HTML5 in this case).
2. **<html>**: Root element of the HTML document.
3. **<head>**: Contains metadata such as title, character encoding, and linked resources.
4. **<title>**: Sets the title displayed in the browser tab.
5. **<meta>**: Provides metadata; e.g., charset and viewport for responsive design.
    a. <meta charset="UTF-8"> is an essential meta tag that tells the browser to use UTF-8 (Unicode Transformation Format - 8-bit) encoding for the web page.
    b. By including <meta name="viewport" content="width=device-width, initial-scale=1.0">, you tell the browser to:
        i. **Use the device's width as the viewport width:** This makes the page adapt to the screen size of the device.
        ii. **Start with no initial zoom:** This presents the page at a readable size from the beginning.
6. **<body>**: Contains the visible content of the webpage.

3. HTML Elements

HTML elements consist of opening and closing tags with content in between.

Syntax:

<tagname>Content</tagname>

Types of Elements:

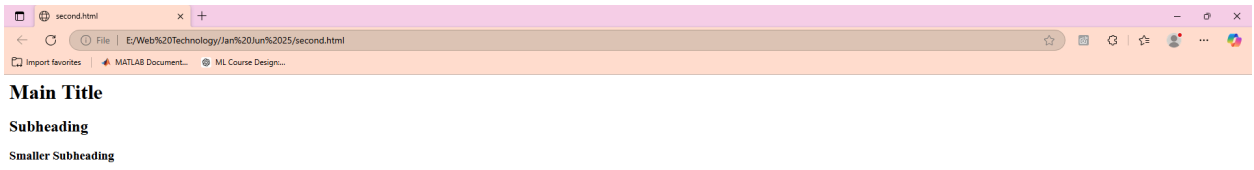
1. **Block-Level Elements:**
    - Start on a new line and take up full width.
    - Examples: , <p>, <h1> to <h6>, <section>, <article>.
2. **Inline Elements:**
    - Do not start on a new line; only take up as much width as necessary.
    - Examples: , <a>, <img>, , .

4. Common HTML Elements

Headings

- Define hierarchical structure.
- <h1> is the largest and most important, <h6> is the smallest.

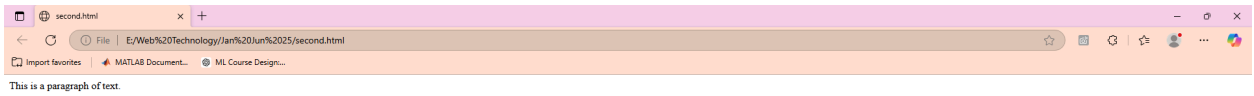
<h1>Main Title</h1>
<h2>Subheading</h2>
<h3>Smaller Subheading</h3>



Paragraphs

- Represent blocks of text.

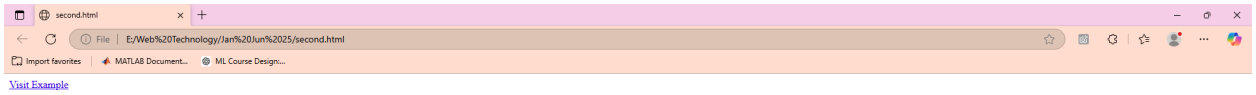
This is a paragraph of text.

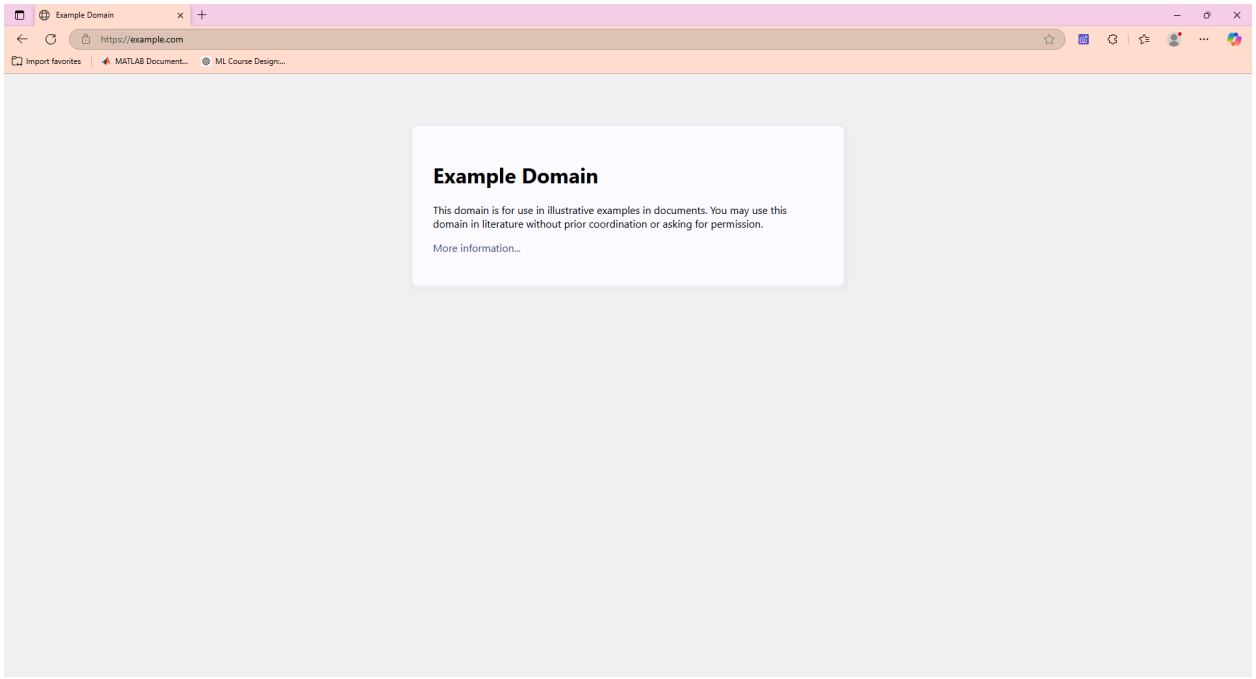


Links

- Create hyperlinks using the <a> tag.

Visit Example

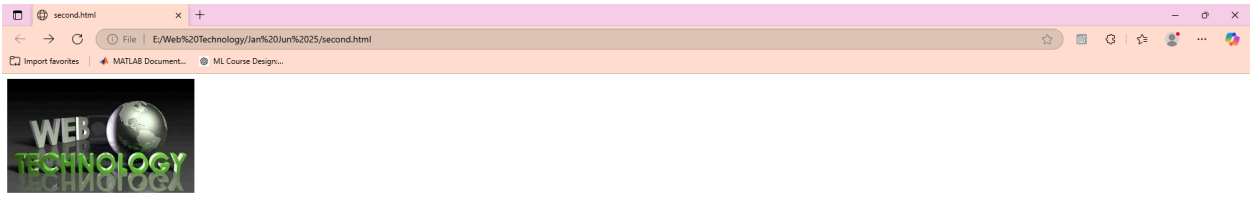


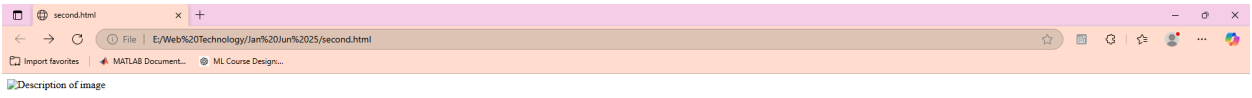


Images

- Display images using the <img> tag. Requires src and alt attributes.

<img src="image.jpg" alt="Description of image">



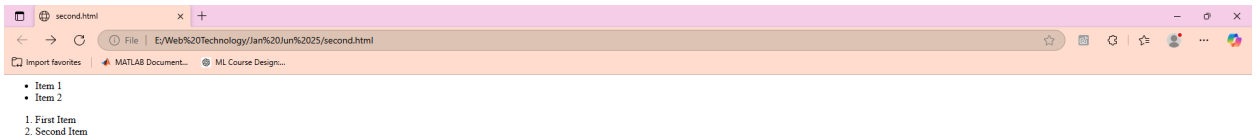


Lists

- Ordered (<ol>) and unordered (<ul>) lists:

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
```

```
</ul>

<ol>
  <li>First Item</li>
  <li>Second Item</li>
</ol>
```

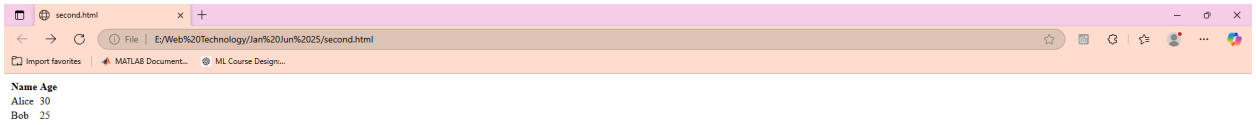


Tables

- Represent tabular data:

```
<table>
  <tr>
    <th>Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Alice</td>
    <td>30</td>
  </tr>
  <tr>
    <td>Bob</td>
    <td>25</td>
```

```
  </tr>
</table>
```



Forms

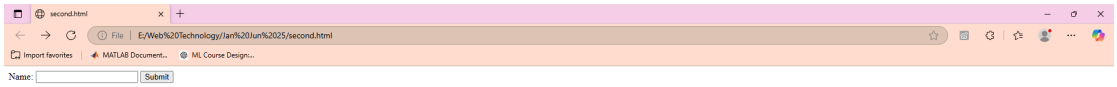
- Collect user input:

```
<form action="submit.php" method="post">
  <label for="name">Name:</label>
  <input type="text" id="name" name="name">
  <button type="submit">Submit</button>
</form>
```

<label for="name">Name:</label> is an HTML element used to create a label for a form control. The for attribute creates an explicit link between the label and the form control. This has several important benefits:

1. **Clickable label:** When the user clicks on the label text ("Name:" in this case), the associated form control (the element with id="name") will receive focus. This makes it easier for users to interact with form elements, especially on touch devices.

2. **Accessibility:** Screen readers and other assistive technologies use the for attribute to associate labels with form controls, making forms more accessible to users with disabilities.



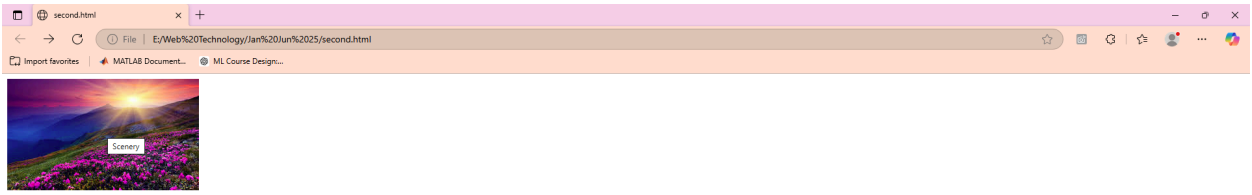
5. Attributes

HTML attributes provide additional information about elements. They are specified within the opening tag.

Examples of Common Attributes:

1. **Global Attributes**:
    - id: Unique identifier.
    - class: Specifies class name for styling.
    - style: Inline CSS styles.
    - title: Tooltip text.
2. **Specific Attributes:**
    - src (in <img>): Specifies the image URL.
    - href (in <a>): Specifies the hyperlink URL.
    - alt (in <img>): Provides alternative text for images.

<img src="image.jpg" alt="A beautiful scenery" title="Scenery">

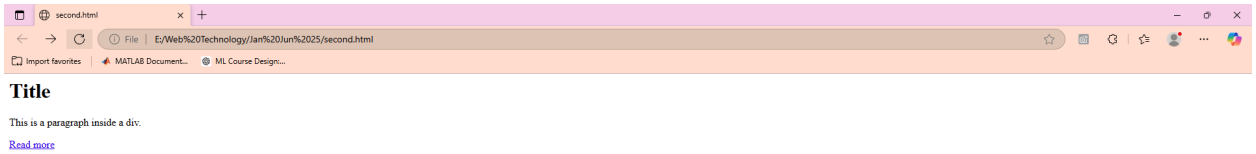


6. Nesting and Hierarchy

Elements can be nested to create a structure.

Example:

```
<div>
  <h1>Title</h1>
  <p>This is a paragraph inside a div.</p>
  <a href="#">Read more</a>
</div>
```



Rules for Nesting:

- Ensure tags are properly closed.
- Avoid overlapping tags.

7. Semantic HTML

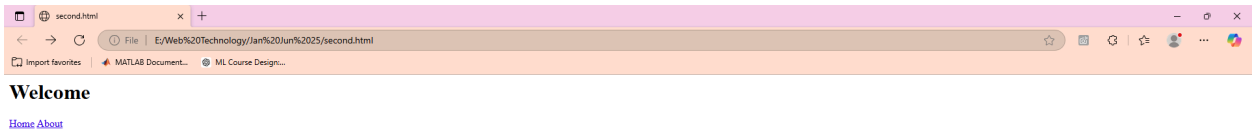
Semantic elements clearly describe their purpose and content.

Examples of Semantic Tags:

- <header>: Defines a header section.
- <footer>: Defines a footer section.
- <article>: Represents self-contained content.
- <section>: Defines sections of a document.
- <nav>: Represents navigation links.

```
<header>
  <h1>Welcome</h1>
  <nav>
    <a href="#home">Home</a>
    <a href="#about">About</a>
  </nav>
```

</header>



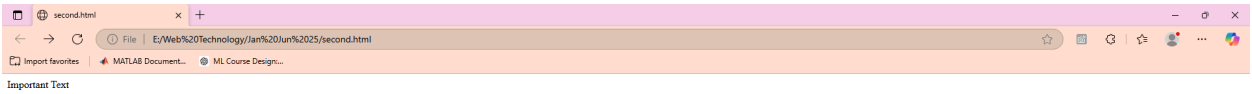
8. Non-Semantic HTML

Non-semantic elements do not explicitly define their meaning.

Examples:

- : Generic container for grouping.
- : Inline container for styling or grouping text.

```
<div class="container">
  <span class="highlight">Important Text</span>
</div>
```



9. Best Practices

1. Use semantic tags for better accessibility and SEO.
2. Close all tags properly.
3. Write readable and organized code with proper indentation.
4. Use meaningful id and class names.
5. Include alt text for all images.

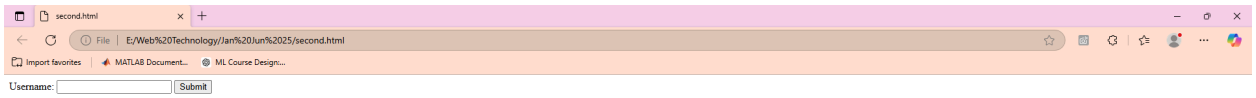
Forms and Validation

1. Introduction to Forms in HTML

Forms are a crucial part of web development, enabling user input collection and interaction with web applications. They are used for tasks like user registration, login, feedback submission, and more.

Basic Structure of an HTML Form:

```
<form action="/submit" method="post">
 <label for="username">Username:</label>
 <input type="text" id="username" name="username">
 <button type="submit">Submit</button>
</form>
```



- **<form>**: Defines the form container.
    - **action**: Specifies the server endpoint where the form data will be sent.
    - **method**: Defines the HTTP method (GET or POST) for data submission.
- **<label>**: Provides a description for input fields.
- **<input>**: Collects user input.

- **<button>**: Triggers form submission.

2. HTML Form Elements

Forms use a variety of elements to collect different types of data.

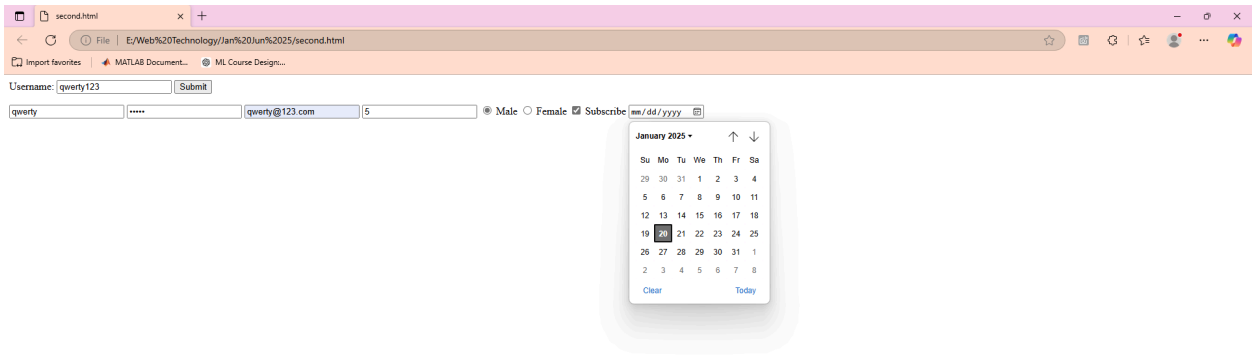
Input Types:

1. **Text Input**: <input type="text" name="name">
2. **Password Input**: <input type="password" name="password">
3. **Email Input**: <input type="email" name="email">
4. **Number Input**: <input type="number" name="age">
5. **Radio Buttons**:

   <input type="radio" name="gender" value="male"> Male

   <input type="radio" name="gender" value="female"> Female

6. **Checkboxes**: <input type="checkbox" name="subscribe" value="yes"> Subscribe
7. **Date Picker**: <input type="date" name="dob">



Other Form Elements:

1. **Text Area:** <textarea name="comments"></textarea>

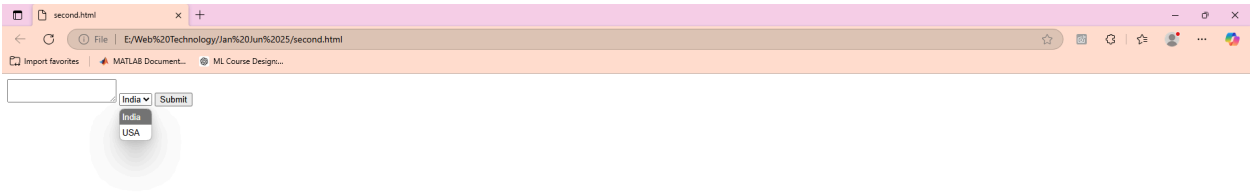
2. **Select Dropdown:**
   <select name="country">

   <option value="india">India</option>

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

   </select>

3. **Submit Button:** <button type="submit">Submit</button>



3. Form Validation

Validation ensures that user input adheres to the required format or rules before it is sent to the server. It can be divided into two types:

a. Client-Side Validation

Performed in the browser using HTML attributes, JavaScript, or external libraries. Ensures a better user experience.

- **HTML5 Validation Attributes:**
  1. **required**: Ensures the field is not left empty.
     <input type="text" name="name" required>

2. **pattern**: Specifies a regex pattern the input must match.
   <input type="text" name="zip" pattern="\d{5}" title="Enter a 5-digit ZIP code">
3. **min and max**: Define numeric input range.
   <input type="number" name="age" min="18" max="99">
4. **maxlength and minlength**: Set character length constraints.
   <input type="text" name="username" minlength="5" maxlength="15">
5. **type validation**: Ensures input matches the defined type (e.g., email, url).

Using JavaScript for Custom Validation:

```
<script>

 document.getElementById("myForm").addEventListener("submit", function(e) {

  const emailField = document.getElementById("email");

  if (!emailField.value.includes("@")) {

   e.preventDefault();

   alert("Please enter a valid email address.");

  }

 });

</script>
```

b. Server-Side Validation

Performed on the server to ensure security and data integrity, even if client-side validation is bypassed.

- **Validation Examples:**
  - Check if fields are empty.
  - Sanitize inputs to prevent SQL injection or XSS attacks.
  - Verify email addresses by sending confirmation links.

4. Enhancing Forms with CSS and Accessibility

- **Styling Forms with CSS:**

Add visual feedback using pseudo-classes:

```
input:focus {
```

```
 border-color: blue;

}

input:invalid {

 border-color: red;

}
```

- **Ensuring Accessibility:**
    - Use **<label>** tags properly.
    - Add ARIA attributes for better screen reader support.
    - Provide clear error messages.

5. Form Best Practices

1. Use semantic HTML for better readability and accessibility.
2. Minimize the number of fields to improve user experience.
3. Provide inline error messages for immediate feedback.
4. Ensure forms are mobile-friendly with responsive designs.

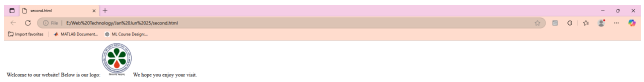
Some Examples:

```
<p>

 Welcome to our website! Below is our logo:

 <img src="logo.png" alt="Website Logo">

 We hope you enjoy your visit.

</p>
```

Welcome to our website! Below is our logo:

<img src="logo.png" alt="Website Logo" height="100">

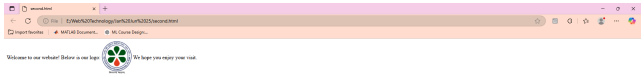
We hope you enjoy your visit.



Welcome to our website! Below is our logo:

<img src="logo.png" alt="Website Logo" height="100" style="vertical-align: middle;">

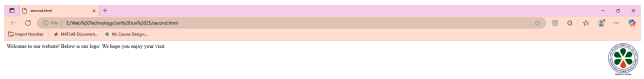
We hope you enjoy your visit.



Welcome to our website! Below is our logo:

<img src="logo.png" alt="Website Logo" height="100" style="float: right;">

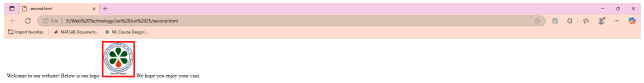
We hope you enjoy your visit.



Welcome to our website! Below is our logo:

<img src="logo.png" alt="Website Logo" height="100" style="border: 5px solid red;">

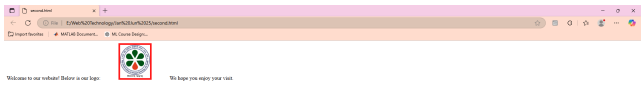
We hope you enjoy your visit.



Welcome to our website! Below is our logo:

<img src="logo.png" alt="Website Logo" height="100" style="border: 5px solid red; margin: 0px 50px">

We hope you enjoy your visit.

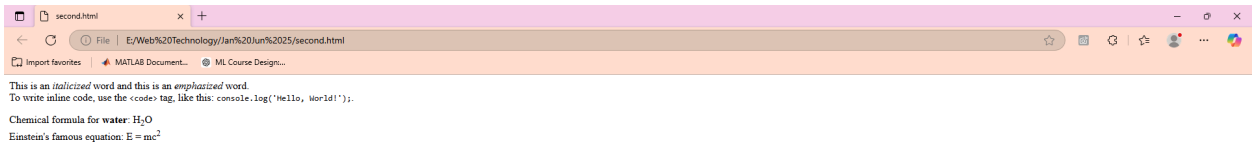


This is an italicized word and this is an emphasized word.

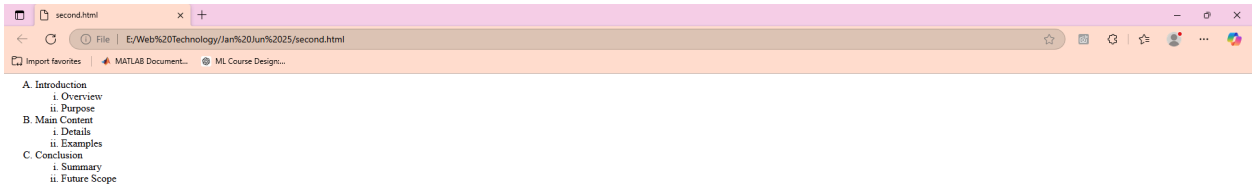
To write inline code, use the <code><code></code> tag, like this: <code>console.log('Hello, World!');</code>.

Chemical formula for water: H₂O

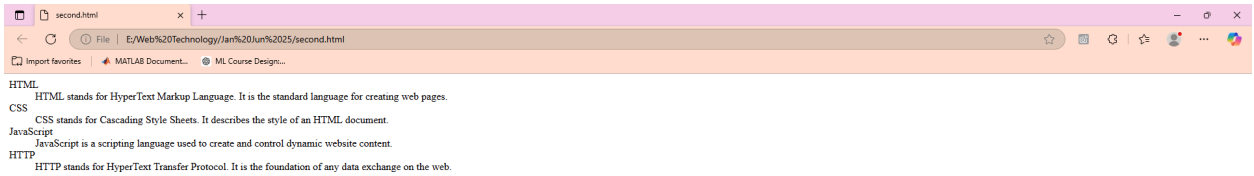
Einstein's famous equation: E = mc²



```
<ol type="A">
    <li>Introduction
      <ol type="i">
        <li>Overview</li>
        <li>Purpose</li>
      </ol>
    </li>
    <li>Main Content
      <ol type="i">
        <li>Details</li>
        <li>Examples</li>
      </ol>
    </li>
    <li>Conclusion
      <ol type="i">
        <li>Summary</li>
        <li>Future Scope</li>
      </ol>
    </li>
  </ol>
```

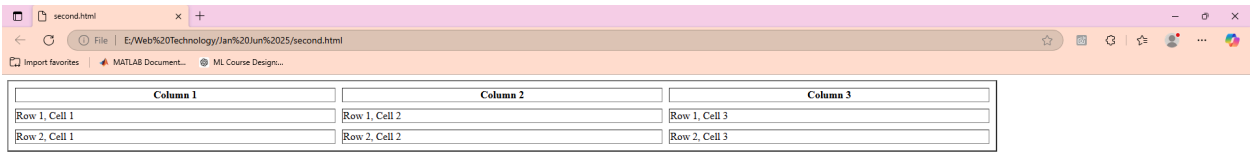



```
<dl>

    <dt>HTML</dt>

    <dd>HTML stands for HyperText Markup Language. It is the standard language for creating web pages.</dd>

    <dt>CSS</dt>

    <dd>CSS stands for Cascading Style Sheets. It describes the style of an HTML document.</dd>

    <dt>JavaScript</dt>

    <dd>JavaScript is a scripting language used to create and control dynamic website content.</dd>

    <dt>HTTP</dt>

    <dd>HTTP stands for HyperText Transfer Protocol. It is the foundation of any data exchange on the web.</dd>

</dl>
```




```
<table width="80%" border="2" cellspacing="10">
  <tr>
    <th>Column 1</th>
    <th>Column 2</th>
    <th>Column 3</th>
  </tr>
  <tr>
    <td>Row 1, Cell 1</td>
    <td>Row 1, Cell 2</td>
    <td>Row 1, Cell 3</td>
  </tr>
  <tr>
    <td>Row 2, Cell 1</td>
```

```
    <td>Row 2, Cell 2</td>
    <td>Row 2, Cell 3</td>
  </tr>
</table>
```



Introduction to CSS

What is CSS?

- **CSS (Cascading Style Sheets)**: A language used to describe the presentation of a document written in HTML or XML.
- It controls the layout, colors, fonts, and overall visual appearance of web pages.
- CSS separates **content** (HTML) from **presentation** (CSS).

Why CSS?

1. **Separation of concerns**:
    - HTML: Defines the structure/content.
    - CSS: Handles the design and styling.
2. **Consistency**: Enables uniform styling across multiple pages.
3. **Efficiency**: Changes in one CSS file can update styles across the entire website.
4. **Improved Accessibility**: Allows for responsive design and compatibility with various devices.
5. **Better Maintenance**: Styling can be updated without touching the HTML.

Features of CSS

1. **Styling**:
    - Text styles (font size, color, weight).
    - Layout styles (positioning, margins, padding).
2. **Media Queries**: Makes web pages responsive to different screen sizes.
3. **Animations**: Add dynamic transitions and effects.
4. **Browser Compatibility**: Supported by all major web browsers.

How CSS Works

1. CSS applies styles to HTML elements through **selectors** and **properties**.
2. A CSS rule consists of:
    - **Selector**: Targets the HTML element(s) to style.
    - **Declaration**: Contains the property and its value.

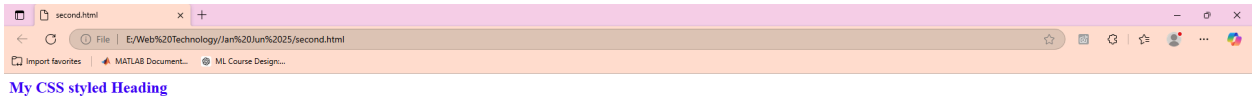
Example of a CSS Rule:

```
h1 {

  color: blue;
```

font-size: 24px;

}

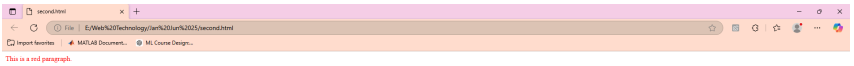
- h1: Selector.
- color and font-size: Properties.
- blue and 24px: Values.



Types of CSS

1. **Inline CSS**: Written directly within an HTML element's style attribute.

Example:
This is a red paragraph.

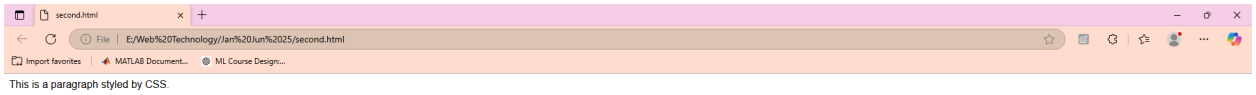


- **Pros**: Easy for quick changes.
- **Cons**: Hard to maintain and lacks separation of concerns.

2. **Internal CSS**: Written inside a <style> tag within the <head> section of an HTML document.

Example:

```
<style>
  p {
    font-family: Arial, sans-serif;
  }
</style>
```



- **Pros**: Useful for single-page styling.
- **Cons**: Does not apply across multiple pages.

3. **External CSS**: Written in a separate .css file and linked to HTML using a <link> tag.

Example:
<link rel="stylesheet" href="styles.css">

- **Pros**: Best for large websites; reusable and maintainable.
- **Cons**: Requires an additional file.

CSS Syntax

Structure:
selector {

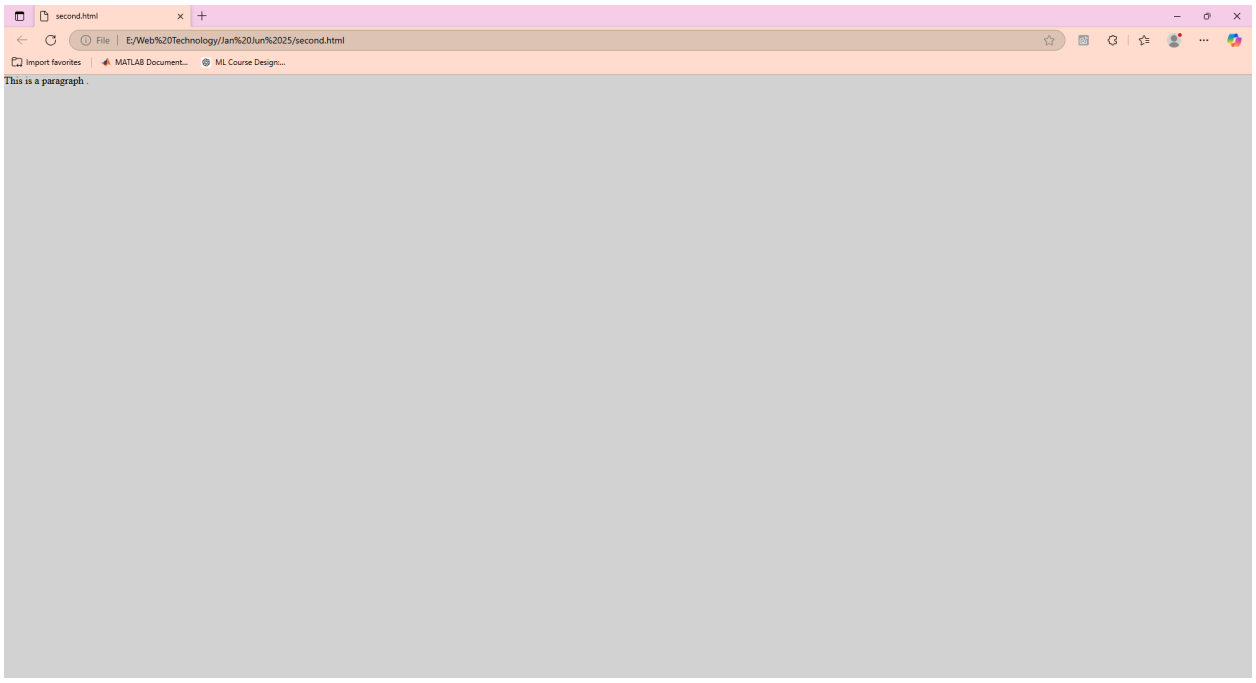
property: value;

}

Example:
body {

background-color: lightgray;

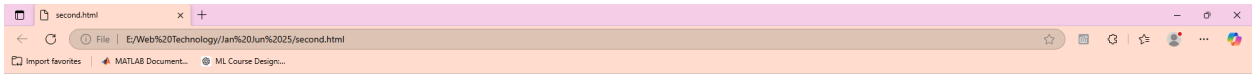
```
  margin: 0;
  font-size: 16px;
}
```



CSS Selectors

Universal Selector (*): Applies to all elements.

```
* {
  margin: 100;
  padding: 0;
}
```



This is a paragraph .

Type Selector: Targets HTML elements.

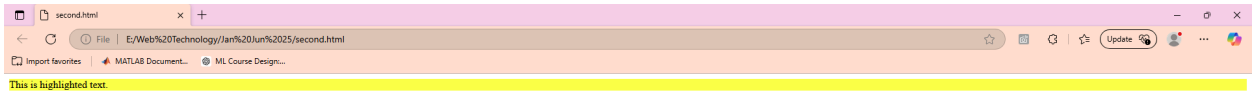
```
h1 {
  color: green;
}
```

Class Selector (.): Targets elements with a specific class.

```
.highlight {
  background-color: yellow;
}
```

Usage in HTML:

```
<p class="highlight">This is highlighted text.</p>
```

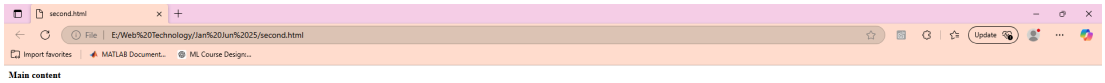


ID Selector (#): Targets an element with a specific ID.

```
#main {

  font-weight: bold;

}
```

Usage in HTML:

```
<div id="main">Main content</div>
```



Group Selector: Applies the same styles to multiple selectors.

```
h1, h2, p {
  color: blue;
}
```

CSS Properties

1. **Text Properties**:
    - color: Sets text color.
    - font-size: Defines text size.
    - text-align: Aligns text (e.g., left, right, center).

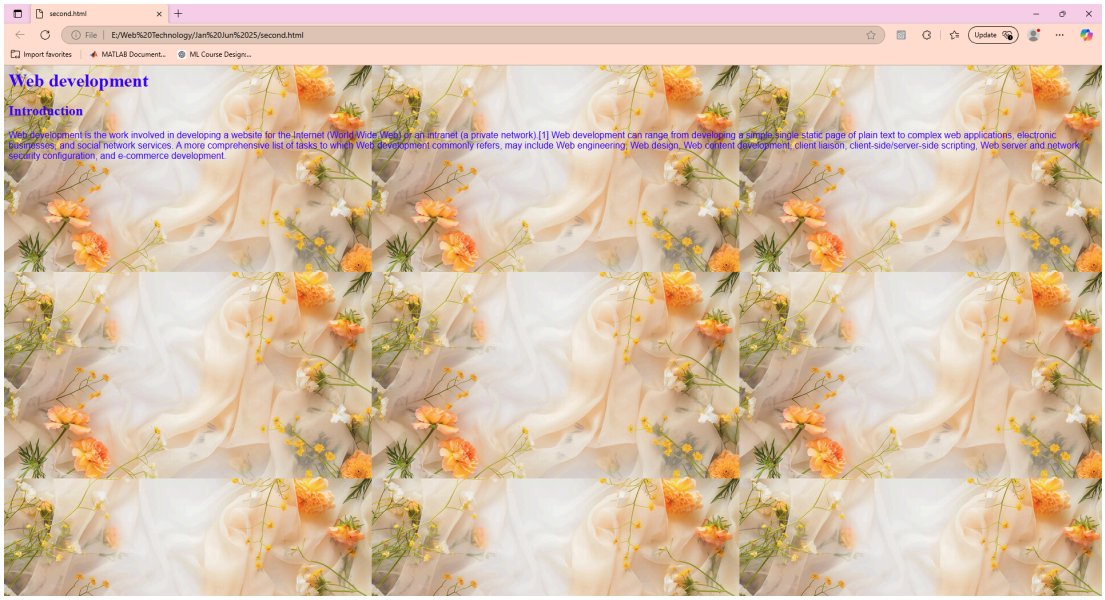
font-family: Sets the font.

```
p {

  font-family: "Arial", sans-serif;

}
```

2. **Background Properties**:
    - background-color: Sets the background color.

background-image: Adds a background image.

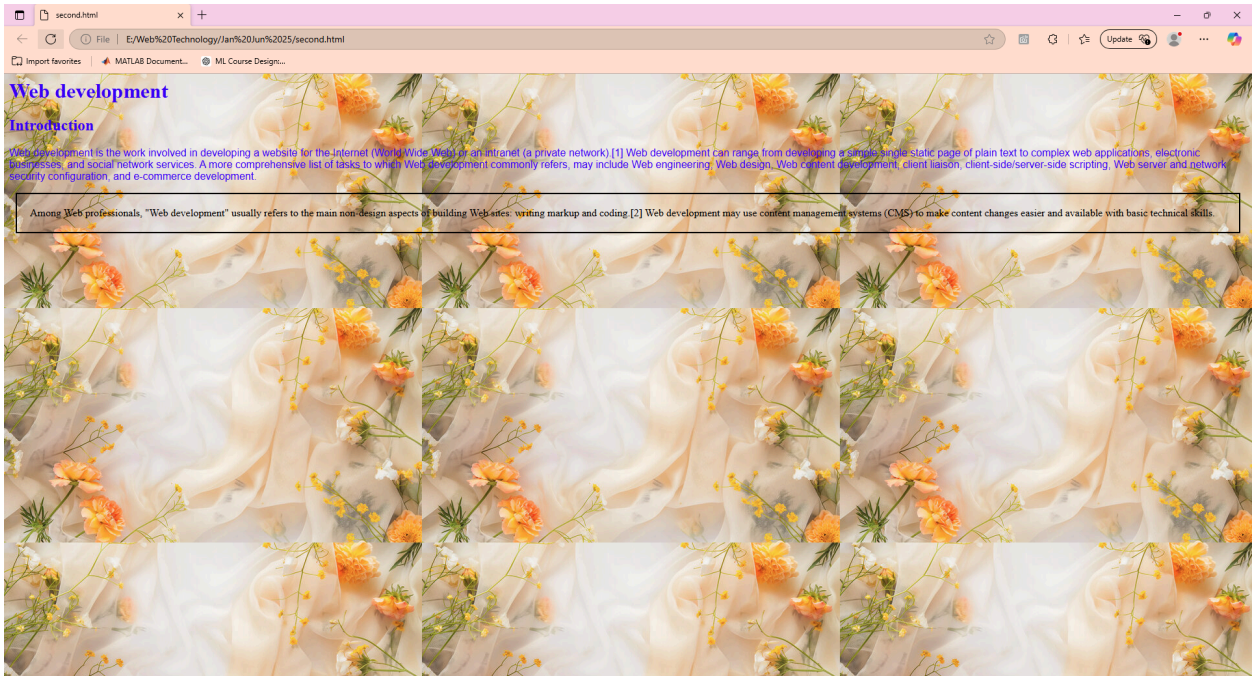
```
body {

  background-image: url("background.jpg");

}
```



3. **Box Model Properties**:
    - margin: Space outside an element.
    - padding: Space inside an element, between content and border.

border: Adds a border around an element.

```
div {

  margin: 10px;

  padding: 20px;

  border: 2px solid black;

}
```



CSS Box Model

- The box model consists of:
    1. **Content**: The content of the box.
    2. **Padding**: Clears an area around the content.
    3. **Border**: Surrounds the padding (optional).
    4. **Margin**: Clears an area outside the border.

Practical Examples

Basic Styling:

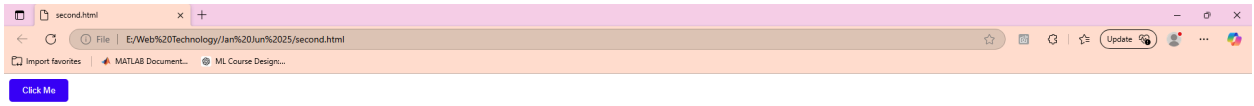
```
<style>
  h1 {
    color: navy;
    text-align: center;
  }
  p {
    font-size: 14px;
```

```
    line-height: 1.5;
  }
</style>
<h1>Welcome to CSS</h1>
<p>This is an introduction to CSS styling.</p>
```

Styling a Button:

```
<style>
  .btn {
    background-color: blue;
    color: white;
    padding: 10px 20px;
    border: none;
    border-radius: 5px;
```

```
    cursor: pointer;
  }
  .btn:hover {
    background-color: darkblue;
  }
</style>
<button class="btn">Click Me</button>
```



Advantages of CSS

1. Faster loading time by reducing inline styles.
2. Enables responsive and consistent designs.
3. Provides flexibility in presenting web content.

Box Model, Positioning, and Layouts in CSS

1. The CSS Box Model

The CSS Box Model is a fundamental concept that defines how elements are displayed and spaced on a web page. Every HTML element is treated as a rectangular box in the box model, consisting of the following components:

Components of the Box Model

1. **Content**:
    - The content inside the box (e.g., text, image, etc.).
    - Controlled by properties like width and height.
2. **Padding**:
    - The space between the content and the border.
    - Transparent by default.

padding: 10px;

3. **Border**:
    - A boundary around the padding (or content if padding is not applied).
    - Can be styled with border-width, border-style, and border-color.

border: 2px solid black;

4. **Margin**:
    - The space outside the border, separating the element from others.
    - Transparent by default.

margin: 15px;

Visual Representation

+-------------------------+

| Margin |

+-------------------------+

| Border |

+-------------------------+

| Padding |

+--------------------------+

| Content |

+--------------------------+

Box-Sizing Property

The box-sizing property determines whether the width and height include padding and border or not.

- content-box (default): Width/height includes only content.
- border-box: Width/height includes content, padding, and border.

Example:

```
div {
  box-sizing: border-box;
  width: 200px;
  padding: 10px;
  border: 5px solid black;
}
```

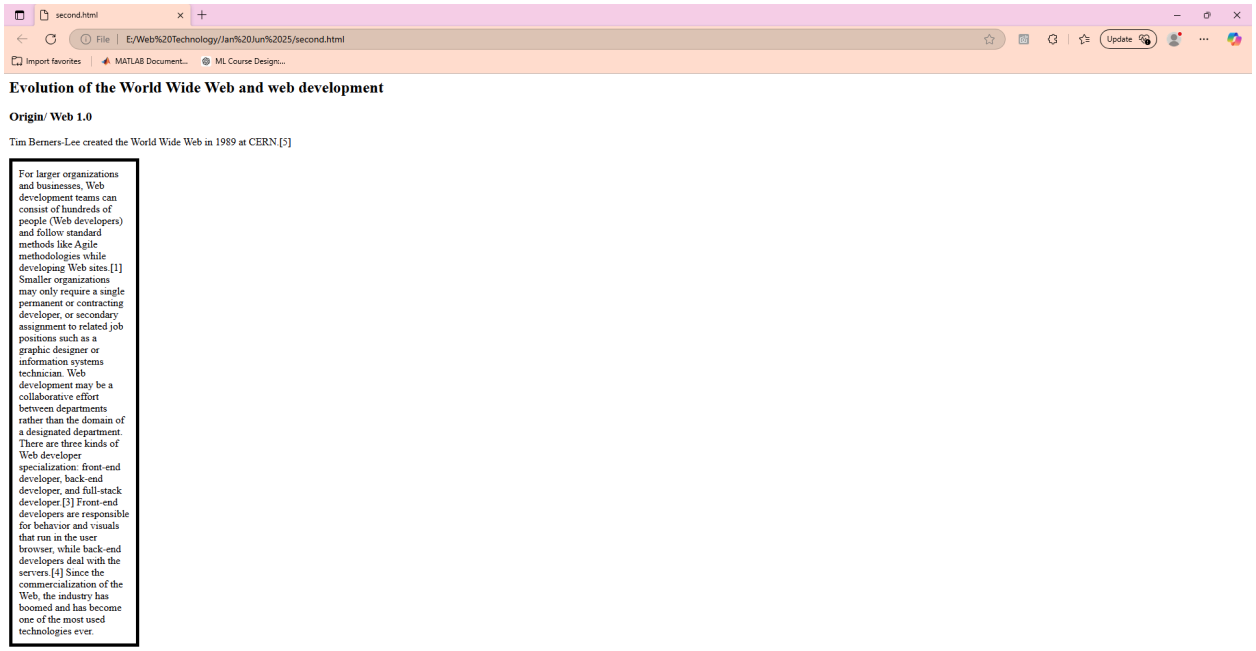
2. CSS Positioning

Positioning in CSS defines how elements are placed in relation to their containing elements or the viewport.

Types of Positioning

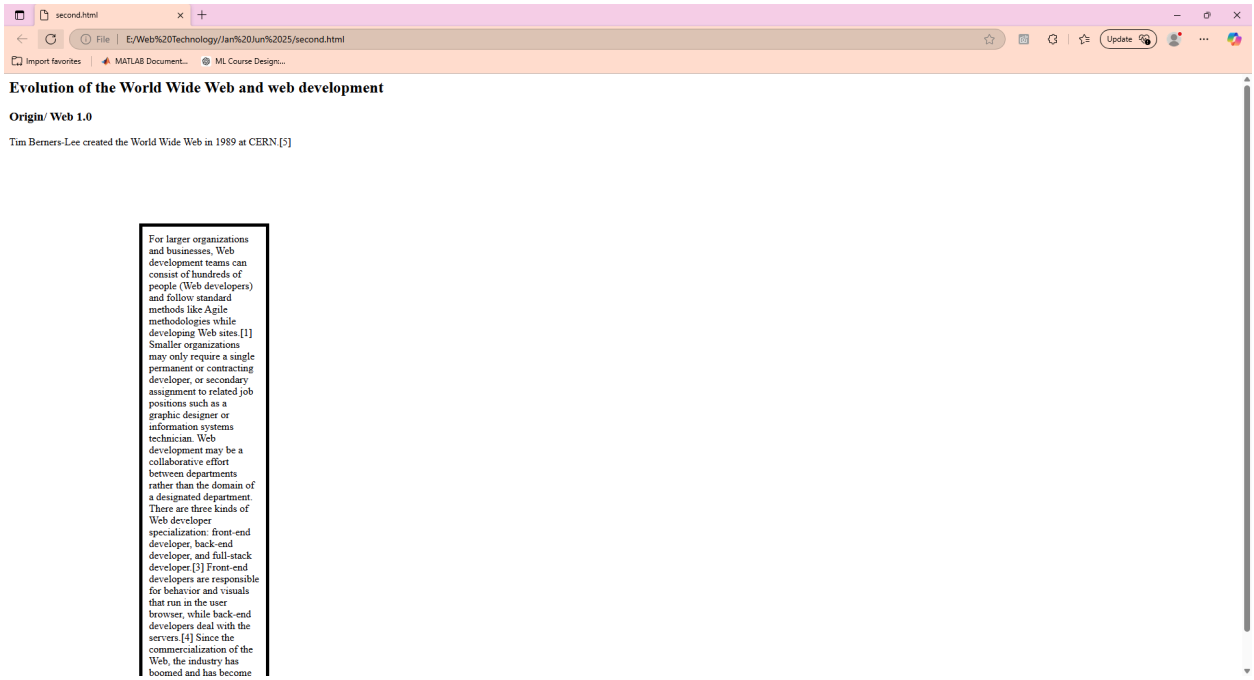
1. **Static (Default)**: Elements are placed in the natural flow of the document.

```
div {
  position: static;
}
```



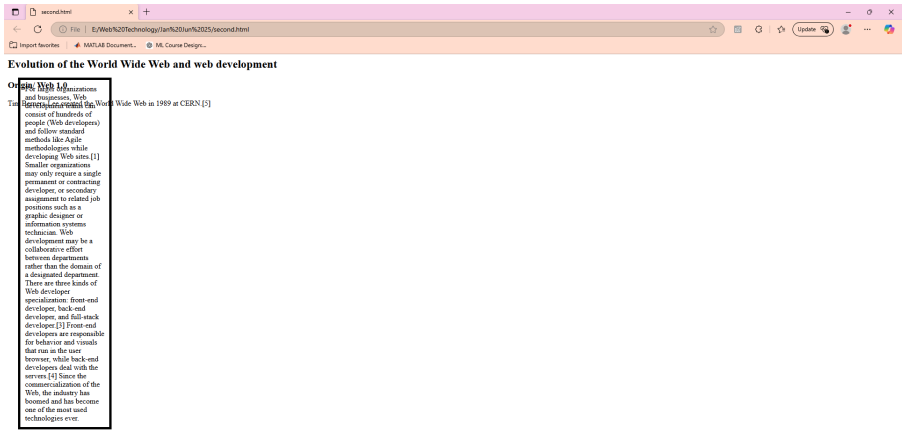
2. **Relative**:
    - Element is positioned relative to its normal position.
    - Can use top, right, bottom, and left.

```
div {
  position: relative;
  top: 10px;
  left: 20px;
}
```



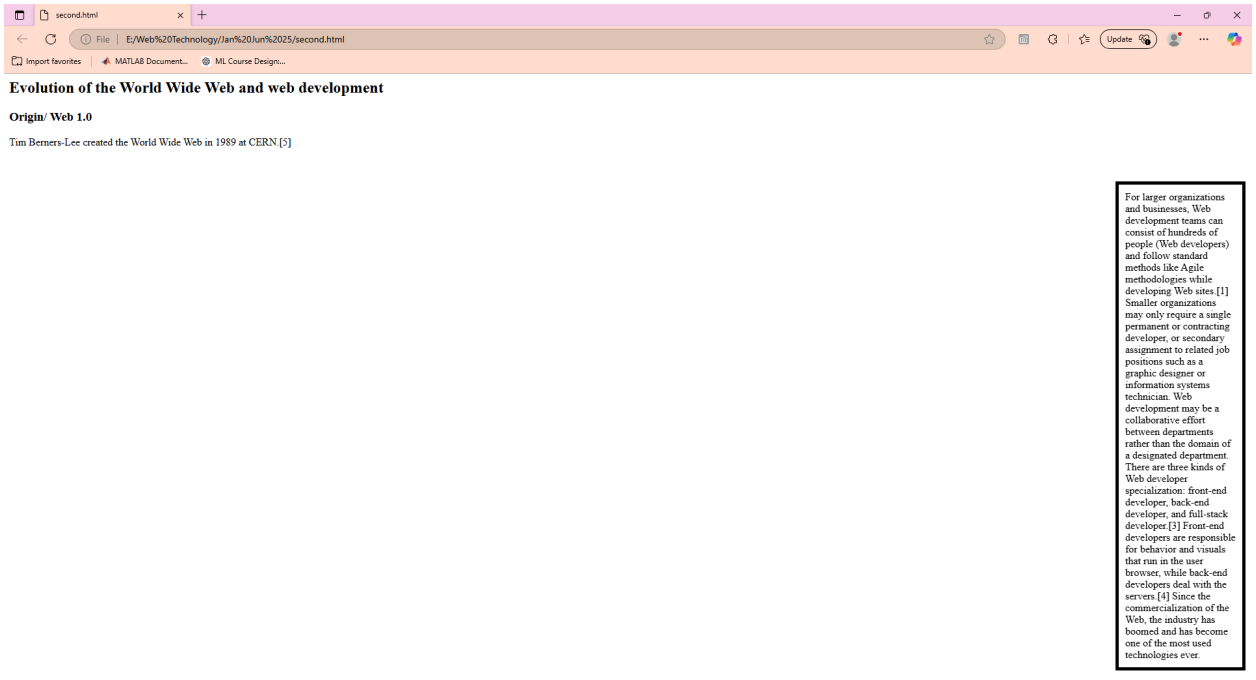
3. **Absolute**: Element is positioned relative to the nearest positioned ancestor (non-static) or the viewport if no ancestor is positioned.

```
div {
  position: absolute;
  top: 50px;
  left: 30px;
}
```



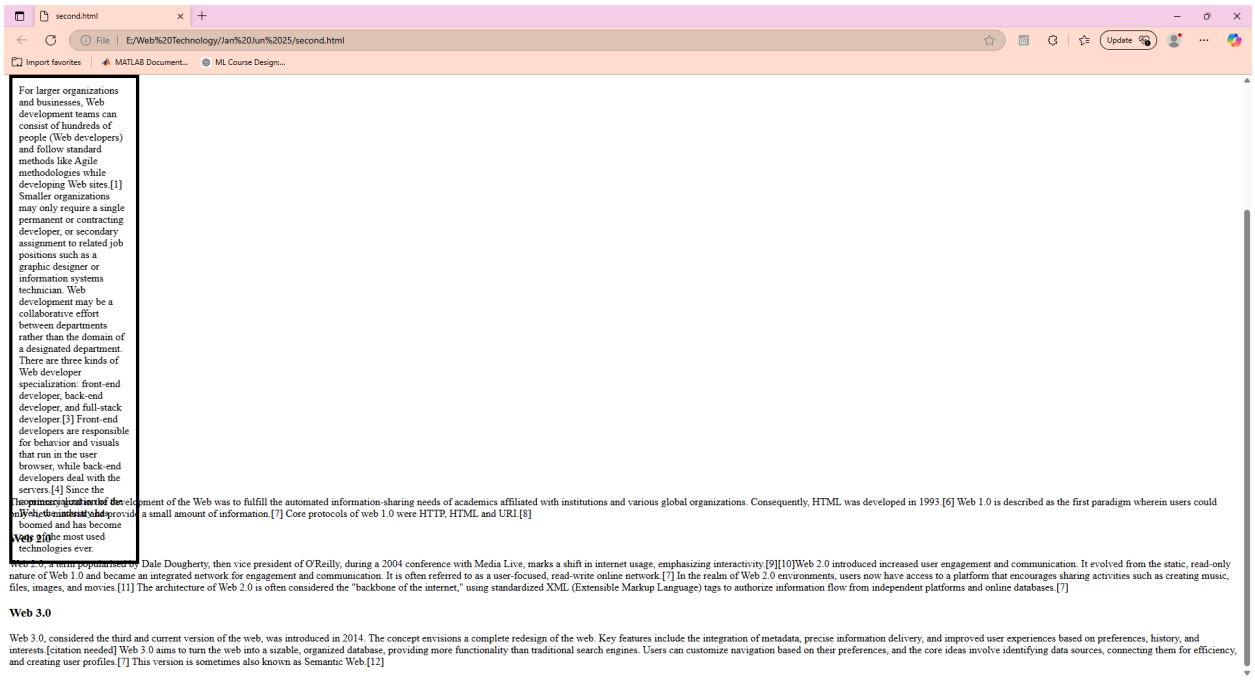
4. **Fixed**: Element is positioned relative to the viewport and does not move when scrolling.

```
div {
  position: fixed;
  bottom: 10px;
  right: 10px;
}
```



5. **Sticky**: Element toggles between relative and fixed based on scroll position.

```
div {
  position: sticky;
  top: 0;
}
```



3. CSS Layouts

Layouts are used to structure the placement of elements on a webpage. CSS provides several methods to create layouts.

3.1 Block and Inline Elements

1. **Block Elements**:
    - Start on a new line and take up the full width available.
    - Example: , <p>, <h1>.

Style example:

```
div {
```

display: block;

}

2. **Inline Elements**:
    - Do not start on a new line and take only as much width as necessary.
    - Example: , <a>, .

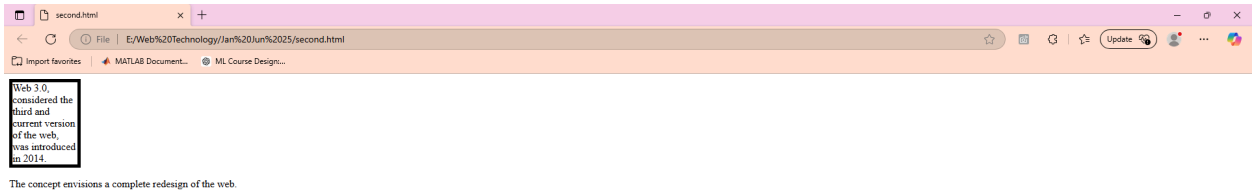
3.2 CSS Display Property

The display property controls how elements are displayed.

- block, inline, inline-block, flex, grid, none, etc.

Example:

```
span {

  display: inline-block;

  width: 100px;

border: 5px solid black;

}
```



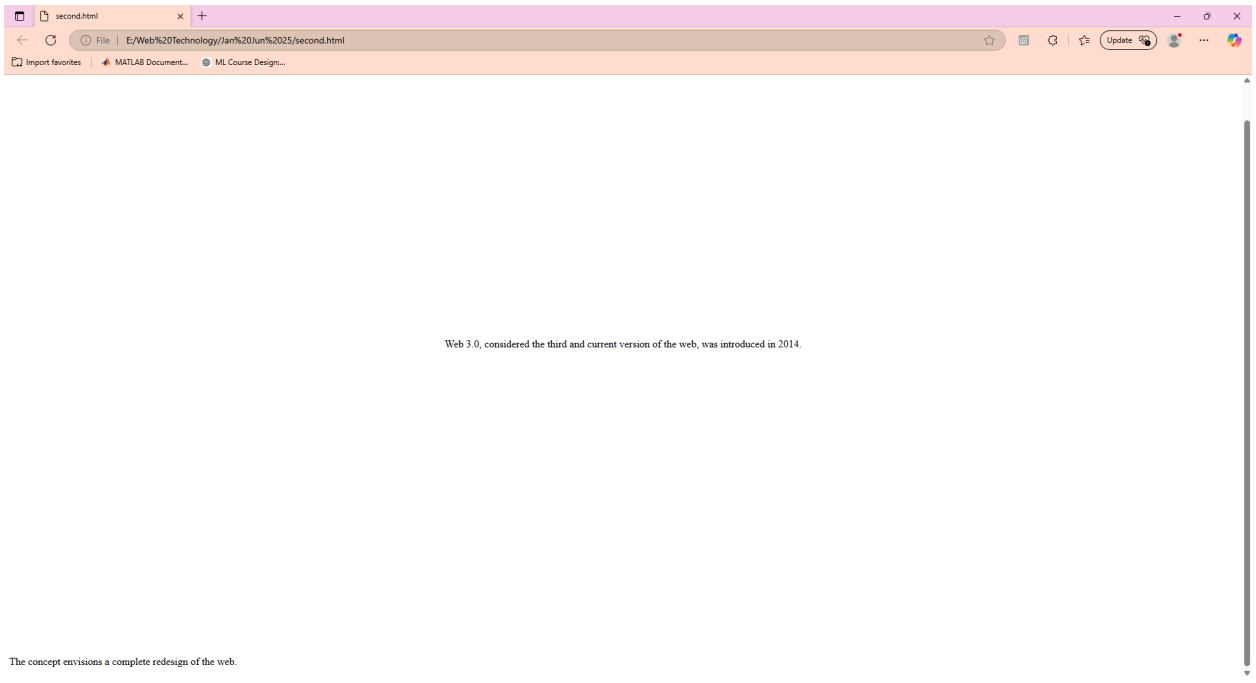
3.3 CSS Flexbox

A layout model designed for distributing space among items in a container.

1. **Key Properties**:
    - display: flex: Enables Flexbox on a container.
    - justify-content: Aligns items horizontally (e.g., center, space-between).
    - align-items: Aligns items vertically (e.g., center, flex-start).
    - flex-direction: Defines the direction of items (row, column).

Example:

```
.container {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}
```



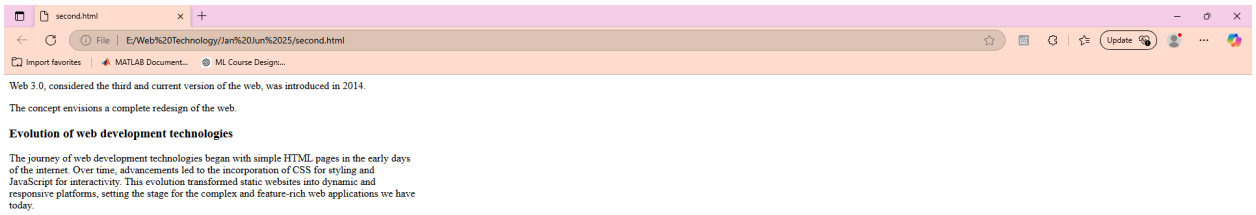
3.4 CSS Grid

A powerful two-dimensional layout system.

1. **Key Properties**:
    - display: grid: Enables Grid on a container.
    - grid-template-columns: Defines the number and size of columns.
    - grid-template-rows: Defines the number and size of rows.
    - gap: Adds spacing between rows and columns.

Example:

```
.grid-container {

  display: grid;

  grid-template-columns: repeat(3, 1fr);

  gap: 10px;

}
```

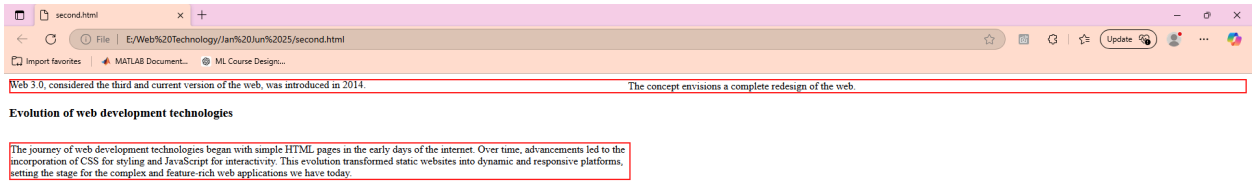


3.5 Float Layout

- Float is used for positioning elements to the left or right.

Example:

```
.box {
  float: left;
  width: 50%;
}
```



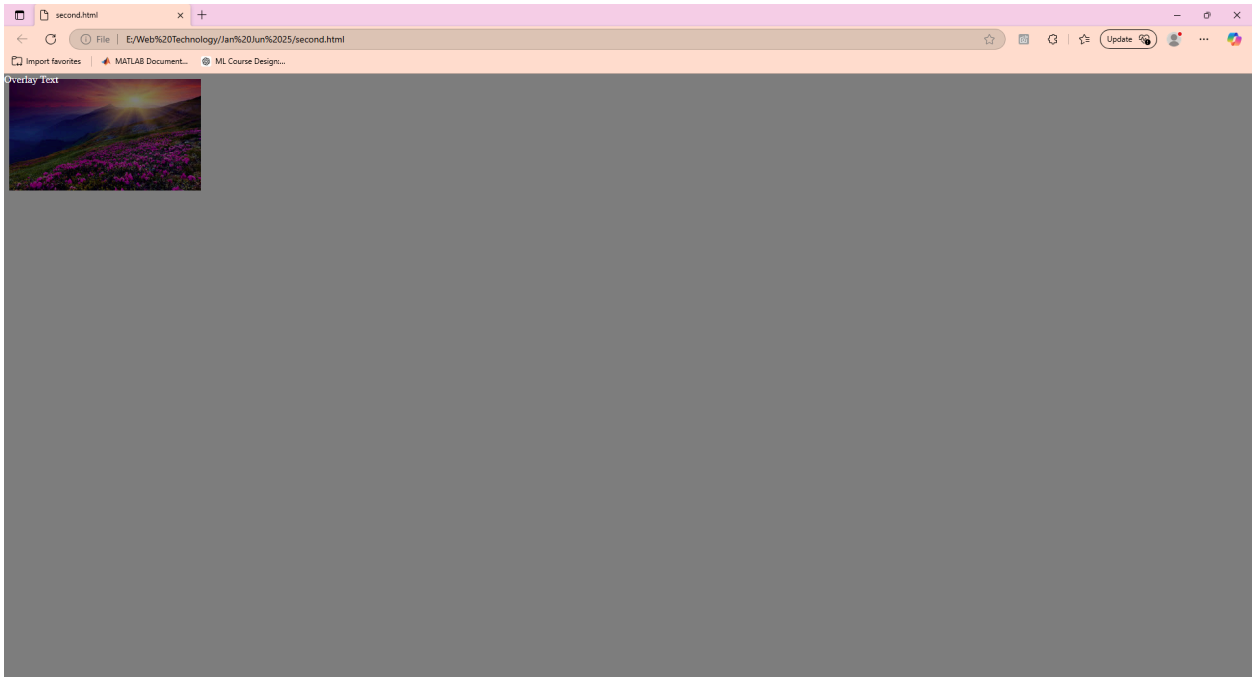
3.6 Position-Based Layout

Using position and z-index for advanced layering and alignment.

Example:

```
  <style>
    .overlay {
      position: absolute;
      top: 0;
      left: 0;
      width: 100%;
```

```
        height: 100%;
        background-color: rgba(0, 0, 0, 0.5);
            }
      </style>
<body>
  <div class="container">
    <img src="image.jpg" alt="Example Image" class="image">
    <div class="overlay">Overlay Text</div>
  </div>
</body>
```



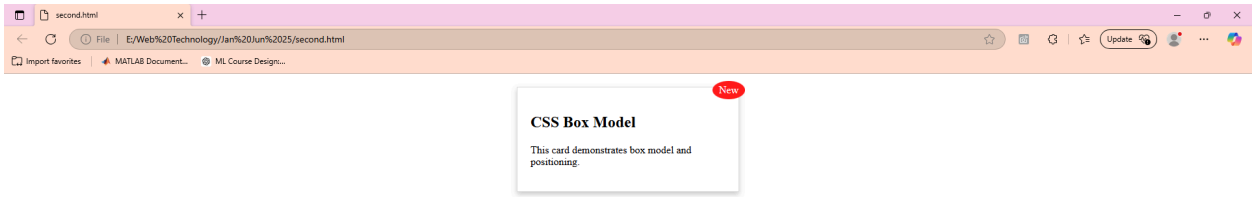
4. Combining Box Model, Positioning, and Layout

Example: Creating a card layout with positioning and box model:

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
  <style>
    .card {
      width: 300px;
      padding: 20px;
      margin: 20px auto;
      border: 1px solid #ddd;
      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
      position: relative;
    }
    .badge {
      position: absolute;
      top: -10px;
      right: -10px;
      background-color: red;
      color: white;
      padding: 5px 10px;
      border-radius: 50%;
    }
  </style>
</head>
```

```
<body>
  <div class="card">
    <div class="badge">New</div>
    <h2>CSS Box Model</h2>
    <p>This card demonstrates box model and positioning.</p>
  </div>
</body>
</html>
```



Responsive Design with Media Queries

Introduction to Responsive Design

Responsive Web Design (RWD) is a design approach that makes web pages **adapt** to different screen sizes, devices, and orientations. It ensures that a website looks **good** and functions **properly** on desktops, tablets, and mobile devices without requiring a separate mobile site.

Key Components of Responsive Design

1. **Fluid Grids** – Layouts use percentage-based widths rather than fixed units like pixels.
2. **Flexible Images** – Images resize proportionally within their container to fit different screens.
3. **Media Queries** – CSS technique to apply different styles based on screen width, height, resolution, or device type.

What are Media Queries?

Media queries are a feature of CSS that allow content to respond to different screen sizes and resolutions by applying specific styles based on conditions like:

- Screen width (most common)
- Screen height
- Device type (mobile, tablet, desktop)
- Orientation (landscape/portrait)
- Resolution (retina displays, high DPI screens)

Syntax of a Media Query

@media (condition) {

```
  /* CSS rules */
}
```

Example:

```
@media (max-width: 600px) {
  body {
    background-color: lightgray;
  }
}
```

This rule changes the background color when the screen width is **600px or smaller**.

Common Media Query Breakpoints

Breakpoints are specific screen widths where the layout changes. Standard breakpoints include:

Device Type	Breakpoint (px)
Extra small devices (phones)	max-width: 600px
Small devices (tablets)	max-width: 768px
Medium devices (laptops)	max-width: 992px
Large devices (desktops)	max-width: 1200px

Example of Multiple Breakpoints

```
/* Mobile (up to 600px) */
@media (max-width: 600px) {
  body {
    background-color: yellow;
  }
```

```
}

/* Tablets (up to 768px) */

@media (max-width: 768px) {

  body {

    background-color: orange;

  }

}

/* Laptops (up to 992px) */

@media (max-width: 992px) {

  body {

    background-color: lightblue;

  }

}

/* Desktops (above 1200px) */

@media (min-width: 1200px) {

  body {

    background-color: green;

  }

}
```

Using Media Queries for Layout Adjustments

1. Responsive Typography

Adjust text size dynamically for different screens.

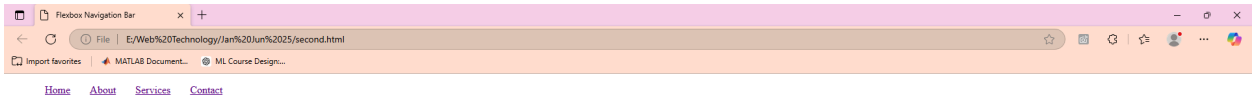
```
@media (max-width: 768px) {
  h1 {
    font-size: 24px;
  }
}
@media (min-width: 769px) {
  h1 {
    font-size: 36px;
  }
}
```

2. Responsive Navigation Menu

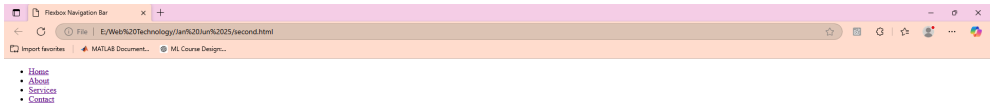
Convert a horizontal menu into a vertical menu for smaller screens.

```
/* Default navigation */
nav ul {
        display: flex;
        list-style: none;
        }
nav ul li {
        margin: 0 15px;
     }
<nav>
```

```
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Services</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </nav>
```




```
/* Mobile view: Stack menu items */
@media (max-width: 600px) {
  nav ul {
    flex-direction: column;
  }
}
```

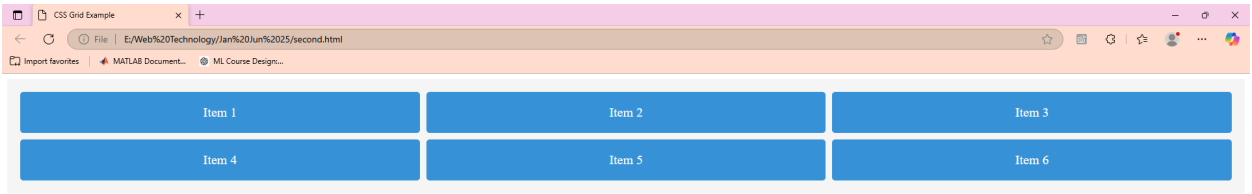


3. Responsive Grid Layout

Use display: grid and display: flex to adjust layouts based on screen size.

```
<head>
  <style>
    /* Container with Grid */
    .container {
      display: grid;
      grid-template-columns: 1fr 1fr 1fr; /* Three equal columns */
      gap: 10px; /* Space between grid items */
      padding: 20px;
      background-color: #f5f5f5;
    }
    /* Grid Items */
    .item {
      background-color: #3498db;
```

```
            color: white;
            font-size: 20px;
            text-align: center;
            padding: 20px;
            border-radius: 5px;
        }
    </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 class="item">Item 4</div>
        <div class="item">Item 5</div>
        <div class="item">Item 6</div>
    </div>
</body>
```




```
/* On smaller screens, use 1 column */
@media (max-width: 768px) {
  .container {
    grid-template-columns: 1fr;
  }
}
```

Advanced Media Query Features

1. Orientation-based Queries

Change styles based on whether the device is in **portrait** or **landscape** mode.

```
@media (orientation: landscape) {
  body {
```

```
    background-color: blue;
  }
}
@media (orientation: portrait) {
  body {
    background-color: pink;
  }
}
```

2. High-Resolution (Retina) Displays

```
@media (min-resolution: 2dppx) {
  body {
    background-image: url('high-res-image.png');
  }
}
```

3. Combining Multiple Conditions

Apply styles when multiple conditions are met.

```
@media (min-width: 600px) and (max-width: 1200px) {
  body {
    font-size: 18px;
  }
}
```

Best Practices for Responsive Design

1. **Use a Mobile-First Approach** – Start with mobile styles and use media queries to enhance for larger screens.
2. **Avoid Fixed Widths** – Use max-width, min-width, and percentages instead of fixed pixel values.
3. **Test on Multiple Devices** – Use tools like **Chrome DevTools** or **online simulators**.
4. **Use Flexbox and CSS Grid** – These layout techniques make responsiveness easier.
5. **Optimize Images for Different Screens** – Use srcset for high-resolution images.

Tools for Responsive Design Testing

- **Google Chrome DevTools** (F12 → Toggle Device Toolbar)
- **Responsive Design Mode** (Firefox Developer Tools)
- **Online Tools**: Am I Responsive?, BrowserStack