HTML & CSS

Complied by Kevin Comba

Table of Contents

HTML

- HTML, a shorthand for Hyper Text Markup Language, is one of the most fundamental building blocks of the Web.
- HTML was officially born in `1993` and since then it evolved into its current state, moving from simple text documents to powering rich Web Applications.
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

What is HTML5?

HTML5 is the newest version of HTML. The term refers to two things. One is the updated HTML language itself, which has new elements and attributes. The second is the larger set of technologies that work with this new version of HTML — like a new video format — and enable you to build more complex and powerful websites and apps.

HTML vs HTML5

HTML is the World Wide Web's core markup language. Originally designed to semantically describe scientific documents, it has since evolved to describe much more.

Most pages on the web today were built using HTML4. Although much improved since the first version of HTML written in 1993, HTML4 still had its limitations. It’s biggest was if web developers or designers wanted to add content or features to their site that weren’t supported in HTML. In that case, they would have to use non-standard proprietary technologies, like Adobe Flash, which required users to install browser plugins. Even then, some users wouldn’t be able to access that content or feature. Users on iPhones and iPads, for example, wouldn’t be able to since those devices don't support Flash.

Cue, HTML5. HTML5 was designed to cut out the need for those non-standard proprietary technologies. With this new version of HTML, you can create web applications that work offline, support high-definition video and animations, and know where you are geographically located.

To understand how HTML5 can do all that, let's look at what's new in this latest version of HTML.

What is new in HTML5?

HTML5 was designed with major objectives, including:

Making code easier to read for users and screen readers Reducing the overlap between HTML, CSS, and JavaScript Promoting design responsiveness and consistency across browsers Supporting multimedia without the need for Flash or other plugins Each of these objectives informed the changes in this new version of HTML. Let's focus on seven of those changes below.

New Semantic Elements

HTML5 introduced several new semantically meaningful tags. These include `<section>`, `<header>`, `<footer>` `<nav>`, `<mark>`, `<figure>`, `<aside>` `<figcaption>`, `<data>`, `<time>`, `<output>`, `<progress>`, `<meter>` and `<main>`. These make it easier to write cleaner code that clearly delineates style from content, which is particularly important to users with assistive technologies like screen readers.

See also

Writerside documentation

Typescript (https://TS.com)

HTML Introduction

```
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

- The `<!DOCTYPE html>` declaration defines that this document is an HTML5 document
- The `<html>` element is the root element of an HTML page
- The `<head>` element contains meta information about the HTML page
- The `<title>` element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The `<body>` element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The `<h1>` element defines a large heading
- The `<p>` element defines a paragraph

What is an HTML Element?

An HTML element is defined by a start tag, some content, and an end tag: `<tagname> Content goes here... </tagname>`

The HTML element is everything from the start tag to the end tag: `<h1>My First Heading</h1>`

HTML Page Structure

Below is a visualization of an HTML page structure:

image.png

Note: The content inside the section will be displayed in a browser. The content inside the element will be shown in the browser's title bar or in the page's tab.

HTML History

YEAR	VERSION
1989	Tim Berners-Lee invented www
1991	Tim Berners-Lee invented HTML
1993	Dave Raggett drafted HTML+
1995	HTML Working Group defined HTML 2.0
1997	W3C Recommendation: HTML 3.2
1999	W3C Recommendation: HTML 4.01
2000	W3C Recommendation: XHTML 1.0
2008	WHATWG HTML5 First Public Draft
2012	WHATWG HTML5 Living Standard
2014	W3C Recommendation: HTML5
2016	W3C Candidate Recommendation: HTML 5.1
2017	W3C Recommendation: HTML5.1 2nd Edition (http://www.w3.org/TR/html51/)
2017	W3C Recommendation: HTML5.2 (http://www.w3.org/TR/html52/)

HTML Basic

HTML Documents

All HTML documents must start with a document type declaration: <!DOCTYPE html>.

The HTML document itself begins with .

The visible part of the HTML document is between .

The <!DOCTYPE> Declaration

The <!DOCTYPE> declaration represents the document type, and helps browsers to display web pages correctly.

It must only appear once, at the top of the page (before any HTML tags).

The `<!DOCTYPE>` declaration is not case sensitive.

The `<!DOCTYPE>` declaration for HTML5 is:

HTML Elements

- An HTML element is defined by a start tag, some content, and an end tag.
- The HTML element is everything from the start tag to the end tag:

`<tagname>Content goes here...</tagname>` Examples of some HTML elements:

`<h1>My First Heading</h1>` `<p>My first paragraph.</p>`

Nested HTML Elements

HTML elements can be nested (this means that elements can contain other elements).

All HTML documents consist of nested HTML elements.

The following example contains four HTML elements (`<html>`, `<body>`, `<h1>` and `<p>`):

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Empty HTML Elements

HTML elements with no content are called empty elements.

The `<br>` tag defines a line break, and is an empty element without a closing tag:

```
<p>This is a <br> paragraph with a line break.</p>
```

HTML is Not Case Sensitive

HTML tags are not case sensitive: `<P>` means the same as `<p>`.

The HTML standard does not require lowercase tags, but **W3C recommends lowercase in HTML**, and demands lowercase for stricter document types like XHTML.

HTML Attributes

HTML attributes provide additional information about HTML elements.

1. All HTML elements can have **attributes**
2. Attributes provide **additional information** about elements
3. Attributes are always specified **in the start tag**
4. Attributes usually come in name/value pairs like: **name="value"**

The href Attribute

The `<a>` tag defines a hyperlink. The `href` attribute specifies the URL of the page the link goes to:

```
<a href="https://teach2give.com">Teach2give website</a>
```

The src Attribute

The `<img>` tag is used to embed an image in an HTML page. The src attribute specifies the path to the image to be displayed:

```
<img src="img_girl.jpg">
```

The width and height Attributes

The `<img>` tag should also contain the width and height attributes, which specify the width and height of the image (in pixels):

```
<img src="img_girl.jpg" width="500" height="600">
```

The alt Attribute

The required `alt` attribute for the `<img>` tag specifies an alternate text for an image, if the image for some reason cannot be displayed. This can be due to a slow connection, or an

error in the `src` attribute, or if the user uses a screen reader.

```
<img src="img_girl.jpg" alt="Girl with a jacket">
```

The style Attribute

The `style` attribute is used to add styles to an element, such as color, font, size,

```
<p style="color:red;">This is a red paragraph.</p>
```

The lang Attribute

You should always include the `lang` attribute inside the `<html>` tag, to declare the language of the Web page. This is meant to assist search engines and browsers.

The following example specifies English as the language:

```
<!DOCTYPE html>
<html lang="en">
<body>
...
</body>
</html>
```

HTML Elements Types

HTML (HyperText Markup Language) defines the structure of web pages. HTML elements are building blocks of HTML documents, consisting of opening and closing tags, attributes, and content. Below are various types of HTML elements, along with examples.

Sure! Here's a comprehensive list of **HTML elements**, including a full breakdown of input elements and all other relevant types in HTML.

1. Structural Elements

These elements define the overall structure of an HTML document.

<html>

Defines the root of an HTML document.

```
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Example</title>
  </head>
  <body>
    <h1>Welcome!</h1>
  </body>
</html>
```

<head>

Contains metadata about the HTML document, like title, character encoding, links to stylesheets, etc.

```
<head>
  <meta charset="UTF-8">
  <title>Page Title</title>
</head>
```

<body>

Defines the body of the HTML document containing content to be displayed on the webpage.

```
<body>
  <h1>Hello, World!</h1>
</body>
```

2. Text Formatting Elements

These elements are used to format and style text.

<h1> - <h6>

Defines header elements, with `<h1>` being the largest and `<h6>` being the smallest.

```
<h1>Main Header</h1>
<h2>Sub Header</h2>
<h3>Sub Header</h3>
<h4>Sub Header</h4>
<h5>Sub Header</h5>
<h6>Sub Header</h6>
```

<p>

Defines a paragraph of text.

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


Inserts a line break.

```
<p>This is line one.<br>This is line two.</p>
```


Makes text bold and indicates strong emphasis.

```
<p><strong>This text is important!</strong></p>
```


Italicizes text to indicate emphasis.

```
<p><em>This text is emphasized.</em></p>
```

<mark>

Highlights text.

```
<p><mark>Highlighted text</mark></p>
```

<small>

Defines smaller text.

```
<p><small>This is small text.</small></p>
```

<del>

Defines text that has been deleted or is no longer accurate.

```
<p><del>This text is deleted.</del></p>
```

<ins>

Defines text that has been inserted or added.

```
<p><ins>This is inserted text.</ins></p>
```

<code>

Defines a piece of computer code.

```
<code>console.log("Hello, World!");</code>
```

<pre>

Defines preformatted text, preserving whitespace and line breaks.

```
<pre>
This is preformatted text
    with indentation and line breaks
</pre>
```

3. Link Elements

These elements are used for hyperlinks and navigation.

<a>

Defines a hyperlink.

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

<link>

Defines relationships between the document and external resources, often used for linking to stylesheets.

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

4. List Elements

These elements create ordered or unordered lists of items.

Defines an unordered (bulleted) list.

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


Defines an ordered (numbered) list.

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


Defines a list item in either `<ul>` or `<ol>`.

```
<li>This is a list item.</li>
```

<dl>

Defines a description list.

```
<dl>
  <dt>Term 1</dt>
  <dd>Description of Term 1</dd>
</dl>
```

<dt>

Defines a term in a description list.

```
<dt>Term</dt>
```

<dd>

Defines the description of a term in a description list.

```
<dd>Description of the term.</dd>
```

5. Form Elements

Form elements allow users to enter data into a webpage.

<form>

Defines a form for collecting user input.

```
<form action="/submit" method="POST">
  <!-- Form fields here -->
</form>
```

<input>

Defines an input field for user data.

```
<input type="text" name="username" placeholder="Enter your username">
```

<textarea>

Defines a multiline input field.

```
<textarea name="message" rows="5" cols="40" placeholder="Enter your
message here"></textarea>
```

<button>

Defines a clickable button.

```
<button type="submit">Submit</button>
```

<select>

Defines a drop-down list.

```
<select name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
</select>
```

<option>

Defines an option in a drop-down list.

```
<option value="volvo">Volvo</option>
```

<optgroup>

Defines a group of related options in a drop-down list.

```
<optgroup label="Swedish Cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
</optgroup>
```

<label>

Defines a label for an input element.

```
<label for="username">Username:</label>
<input type="text" id="username" name="username">
```

<fieldset>

Defines a group of related elements in a form.

```
<fieldset>
  <legend>Personal Information</legend>
  <!-- Form fields here -->
</fieldset>
```

<legend>

Defines a title for a `<fieldset>` element.

```
<legend>Contact Information</legend>
```

6. Table Elements

These elements are used to display tabular data.

<table>

Defines a table.

```
<table>
  <tr>
    <th>Header 1</th>
    <th>Header 2</th>
  </tr>
  <tr>
    <td>Data 1</td>
    <td>Data 2</td>
  </tr>
</table>
```

<tr>

Defines a table row.

```
<tr>
  <td>Row 1, Column 1</td>
  <td>Row 1, Column 2</td>
</tr>
```

<th>

Defines a table header cell (text is bold and centered by default).

```
<th>Header</th>
```

<td>

Defines a table data cell.

```
<td>Data</td>
```

<caption>

Defines a title or description for a table.

```
<caption>Employee Table</caption>
```

<thead>

Groups the header content in a table.

```
<thead>
  <tr>
    <th>Name</th>
    <th>Age</th>
  </tr>
</thead>
```

<tbody>

Groups the body content in a table.

```
<tbody>
  <tr>
    <td>John</td>
    <td>25</td>
  </tr>
</tbody>
```

<tfoot>

Groups the footer content in a table.

```
<tfoot>
  <tr>
    <td colspan="2">Total</td>
  </tr>
</tfoot>
```

7. Media Elements

Used for embedding audio, video, and images.

Defines an image.

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

<audio>

Defines an audio file.

```
<audio controls>
  <source src="audio.mp3" type="audio/mp3">
  Your browser does not support the audio element.
</audio>
```

<video>

Defines a video file.

```
<video width="320" height="240" controls>
  <source src="video.mp4" type="video/mp4">
  Your browser does not support the video tag.
</video>
```

<source>

Defines multiple media resources for `<video>` and `<audio>`.

```
<video controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
</video>
```

<track>

Defines text tracks for `<video>` and `<audio>` elements (e.g., subtitles).

```
<video controls>
  <source src="movie.mp4" type="video/mp4">
  <track src="subtitles_en.vtt" kind="subtitles" srclang="en"
label="English">
</video>
```

<picture>

Defines a container for multiple image sources (responsive images).

```
<picture>
  <source srcset="image.webp" type="image/webp">
  <source srcset="image.jpg" type="image/jpeg">
  <img src="image.jpg" alt="A sample image">
</picture>
```

8. Semantic HTML Elements

Provide meaning to the structure and help improve accessibility.

<article>

Defines an independent, self-contained piece of content.

```
<article>
  <h2>Article Title</h2>
  <p>This is an article content.</p>
</article>
```

<section>

Defines a section of content, often with a heading.

```
<section>
  <h2>Section Title</h2>
  <p>This is a section.</p>
</section>
```

<nav>

Defines navigation links.

```
<nav>
  <a href="#home">Home</a>
```

```
  <a href="#about">About</a>
</nav>
```

<footer>

Defines a footer for a page or section.

```
<footer>
  <p>Footer content</p>
</footer>
```

<header>

Defines the header for a page or section.

```
<header>
  <h1>Welcome to My Website</h1>
</header>
```

<aside>

Defines content that is tangentially related to the content around it.

```
<aside>
  <h3>Related Links</h3>
  <ul>
    <li><a href="#">Link 1</a></li>
    <li><a href="#">Link 2</a></li>
  </ul>
</aside>
```

9. Script and Meta Elements

These are used for scripts and meta-information in the document.

<script>

Defines client-side JavaScript.

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

<meta>

Provides metadata about the document, such as charset, viewport settings, author, etc.

```
<meta charset="UTF-8">
<meta name="author" content="John Doe">
```

<style>

Defines internal CSS for the document.

```
<style>
  body { background-color: lightblue; }
</style>
```

<link>

Links to external resources like stylesheets.

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

<noscript>

Defines alternative content for browsers that do not support JavaScript.

```
<noscript>Your browser does not support JavaScript.</noscript>
```

HTML Semantic Elements

Semantic HTML elements are those that convey meaning about the content they contain. They describe their content more clearly than non-semantic elements like `<div>` and `<span>`. HTML5 introduced several semantic elements that make it easier for developers to structure web pages logically and for browsers and search engines to understand and interpret the content. These elements help with SEO (Search Engine Optimization) and accessibility, as well as making the code more readable and maintainable.

Here's a detailed guide to **Semantic HTML Elements** introduced in HTML5, with code samples.

1. <article>

The `<article>` element represents independent, self-contained content that could be distributed and reused, such as a blog post, news article, or forum post.

- **Use Case**: For any content that can stand alone or be syndicated independently.
- **Example**:

```
<article>
  <header>
    <h2>Understanding HTML5 Semantics</h2>
    <p><time datetime="2025-05-06">May 6, 2025</time> by John Doe</p>
  </header>
  <p>HTML5 introduced several new elements that make webpages more
meaningful and accessible. This is an article explaining those changes.
</p>
  <footer>
    <p>Tags: <a href="#">HTML5</a>, <a href="#">Semantics</a></p>
  </footer>
</article>
```

2. <section>

The `<section>` element represents a generic section of content, often with a heading. It's used to group thematically related content, such as a chapter in a document or a thematic group of content in a webpage.

- **Use Case**: Group related content, like an introduction, main content, or a section of an article.
- **Example**:

```
<section>
  <header>
    <h3>Introduction to HTML5 Semantics</h3>
  </header>
  <p>HTML5 semantics help make the structure of a webpage more
meaningful.</p>
</section>
```

3. <nav>

The `<nav>` element defines navigation links. It indicates to the browser and screen readers that the links inside this container represent a navigation menu.

- **Use Case**: For grouping primary navigation links, like a website's main menu.
- **Example**:

```
<nav>
  <ul>
    <li><a href="#home">Home</a></li>
    <li><a href="#about">About Us</a></li>
    <li><a href="#services">Services</a></li>
    <li><a href="#contact">Contact</a></li>
  </ul>
</nav>
```

4. <header>

The `<header>` element represents a group of introductory content or a set of navigational links. It typically contains things like the logo, navigation links, or introductory information about the webpage.

- **Use Case**: To wrap introductory content or navigation for a section or the entire page.
- **Example**:

```
<header>
  <h1>Welcome to My Website</h1>
  <nav>
    <ul>
      <li><a href="#home">Home</a></li>
      <li><a href="#about">About</a></li>
      <li><a href="#contact">Contact</a></li>
    </ul>
  </nav>
</header>
```

5. <footer>

The `<footer>` element defines a footer for a document or section. It typically contains information like copyright notices, contact information, or related links.

- **Use Case**: To wrap up a section or the entire document with footer-related content.
- **Example**:

```
<footer>
  <p>&copy; 2025 My Website</p>
  <p>Contact: <a href="mailto:info@mywebsite.com">info@mywebsite.com</a>
</p>
</footer>
```

6. <aside>

The `<aside>` element represents content that is tangentially related to the content around it, often presented as a sidebar. It can be used for side notes, additional information, or advertisements.

- **Use Case**: For related but non-essential content that could be moved without affecting the main content.
- **Example**:

```
<aside>
  <h4>Did You Know?</h4>
  <p>HTML5 introduced new semantic elements to help make webpages more accessible.</p>
</aside>
```

7. <main>

The `<main>` element represents the dominant content of the `<body>`, excluding things like headers, footers, and sidebars. There should be only one `<main>` element per document.

- **Use Case**: For wrapping the primary content of a webpage or document.
- **Example**:

```
<main>
  <h2>Featured Articles</h2>
  <article>
    <h3>HTML5 Features</h3>
    <p>This article talks about the new features in HTML5.</p>
  </article>
  <article>
    <h3>CSS Grid Layout</h3>
    <p>Learn how to use CSS Grid to create complex layouts easily.</p>
```

```
  </article>
</main>
```

8. <figure>

The `<figure>` element represents self-contained content, often with a `<figcaption>`. It is used to group media like images, diagrams, or videos and their respective captions.

- **Use Case**: For grouping an image or video with a caption.
- **Example**:

```
<figure>
  <img src="html5-logo.png" alt="HTML5 Logo">
  <figcaption>The HTML5 Logo</figcaption>
</figure>
```

9. <figcaption>

The `<figcaption>` element defines a caption for the content inside a `<figure>`.

- **Use Case**: To provide a caption or description for an image, video, or other media in a `<figure>`.
- **Example**:

```
<figure>
  <img src="js-frameworks.jpg" alt="JavaScript Frameworks">
  <figcaption>Popular JavaScript Frameworks in 2025</figcaption>
</figure>
```

10. <mark>

The `<mark>` element represents text that has been highlighted or marked for reference or emphasis. It is typically rendered with a yellow background color.

- **Use Case**: To highlight text for emphasis, usually used in search results.
- **Example**:

```
<p>This is <mark>important</mark> information about HTML5 semantics.</p>
```

11. <progress>

The `<progress>` element represents the completion of a task, like a progress bar. It is used for displaying the progress of a process such as file downloads or uploads.

- **Use Case**: To show a visual indication of progress.
- **Example**:

```
<progress value="70" max="100">70%</progress>
```

12. <meter>

The `<meter>` element represents a scalar measurement within a known range, such as a disk usage bar or a temperature gauge.

- **Use Case**: For displaying a measurement within a known range.
- **Example**:

```
<meter value="0.7" min="0" max="1">70%</meter>
```

13. <details> and <summary>

The `<details>` element creates a disclosure widget from which the user can obtain additional information or controls. The `<summary>` element is used to define a heading or label for the `<details>` element.

- **Use Case**: For creating collapsible content (e.g., FAQs, or additional information).

- **Example**:

```
<details>
  <summary>Click to view more information</summary>
  <p>This is the additional information that can be toggled open and
closed.</p>
</details>
```

14. <time>

The `<time>` element represents a specific period in time or a date/time value. It's useful for marking dates, times, or durations in content.

- **Use Case**: To represent a date, time, or duration.
- **Example**:

```
<p>We will meet on <time datetime="2025-05-06">May 6, 2025</time> at
10:00 AM.</p>
```

15. <wbr>

The `<wbr>` element specifies where the browser can optionally break a word for wrapping if necessary. This helps improve text layout and readability in narrow spaces.

- **Use Case**: To suggest line break opportunities in long words.
- **Example**:

```
<p>This is a longword that might break at <wbr>appropriate places.</p>
```

HTML Blogs Layouts

Below are four different blog layouts using HTML5 semantic elements. These layouts represent various structures that a blog page might have. Each layout makes use of semantic elements introduced in HTML5, such as `<header>`, `<footer>`, `<section>`, `<article>`, `<aside>`, and others.

Below are four different blog layouts using **HTML5 semantic elements**. These layouts represent various structures that a blog page might have. Each layout makes use of semantic elements introduced in HTML5, such as `<header>`, `<footer>`, `<section>`, `<article>`, `<aside>`, and others.

1. Basic Blog Layout

This layout is a simple blog page with a header, main content area, a sidebar, and a footer. It includes a list of blog articles.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Blog - Basic Layout</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 0; padding: 0;
background-color: #f4f4f4; }
    header, footer { background-color: #333; color: white; padding:
1rem; text-align: center; }
    main { display: flex; padding: 20px; }
    .content { flex: 3; padding: 20px; }
    .sidebar { flex: 1; padding: 20px; background-color: #eee; }
    article { background-color: white; margin-bottom: 20px; padding:
15px; border-radius: 5px; }
    h1, h2 { color: #333; }
    footer { font-size: 0.9rem; }
  </style>
</head>
```

```
<body>
  <header>
    <h1>My Blog</h1>
    <nav>
      <a href="#">Home</a> | <a href="#">About</a> | <a
href="#">Contact</a>
    </nav>
  </header>

  <main>
    <section class="content">
      <article>
        <header>
          <h2>Blog Post 1</h2>
          <p><time datetime="2025-05-06">May 6, 2025</time> by <a
href="#">John Doe</a></p>
        </header>
        <p>This is the content of the first blog post.</p>
      </article>
      <article>
        <header>
          <h2>Blog Post 2</h2>
          <p><time datetime="2025-05-05">May 5, 2025</time> by <a
href="#">Jane Smith</a></p>
        </header>
        <p>This is the content of the second blog post.</p>
      </article>
    </section>

    <aside class="sidebar">
      <h3>About Me</h3>
      <p>Hi, I'm John Doe, a passionate writer and blogger.</p>
      <h3>Recent Posts</h3>
      <ul>
        <li><a href="#">Blog Post 1</a></li>
        <li><a href="#">Blog Post 2</a></li>
        <li><a href="#">Blog Post 3</a></li>
      </ul>
```

```
    </aside>
  </main>

  <footer>
    <p>&copy; 2025 My Blog. All rights reserved.</p>
  </footer>
</body>
</html>
```

2. Blog with Featured Image and Sidebar

This layout features a more detailed blog post with a large featured image, content area, and sidebar. The sidebar includes recent posts, a search bar, and a social media link.

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Blog - Featured Post</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f9f9f9;
        }

        header,
        footer {
            background-color: #444;
            color: white;
            padding: 1rem;
            text-align: center;
        }
```

```
main {
    display: flex;
    padding: 20px;
}

.content {
    flex: 2;
    padding: 20px;
}

.sidebar {
    flex: 1;
    padding: 20px;
    background-color: #e9e9e9;
}

article {
    background-color: white;
    margin-bottom: 20px;
    padding: 20px;
    border-radius: 8px;
}

h1,
h2 {
    color: #333;
}

img {
    width: 100%;
    border-radius: 5px;
}

.recent-posts {
    margin-top: 30px;
}
```

```
        /* body {
            display: flex;
            flex-direction: column;
            min-height: 100vh; /* Ensures body takes at least full
viewport height */
        /* } */

        /* main { */
        /* flex-grow: 1; Allows main content to expand and push footer
down */
        /* } */

        /* footer { */
        /* font-size: 0.9rem; This was the content of your original
selection */
        /* No additional positioning properties are needed for the
footer with this flexbox approach */
        /* } */
    </style>
</head>

<body>
<header>
    <h1>Blog with Featured Image</h1>
    <nav>
        <a href="#">Home</a> | <a href="#">Categories</a> | <a
href="#">Contact</a>
    </nav>
</header>

<main>
    <section class="content">
        <article>
            <header>
                <h2>Exploring the Beauty of HTML5</h2>
                <p><time datetime="2025-05-06">May 6, 2025</time> by <a
href="#">John Doe</a></p>
            </header>
```

```
            <img src="https://via.placeholder.com/800x400" alt="HTML5
Image">
            <p>HTML5 introduced many important features that
revolutionized web development. Let’s dive into what
                makes HTML5 stand out and how it empowers developers.
</p>
        </article>
    </section>

    <aside class="sidebar">
        <h3>Recent Posts</h3>
        <ul>
            <li><a href="#">Understanding JavaScript</a></li>
            <li><a href="#">Introduction to CSS Grid</a></li>
            <li><a href="#">Building Responsive Websites</a></li>
        </ul>

        <h3>Follow Us</h3>
        <a href="#">Facebook</a> | <a href="#">Twitter</a> | <a
href="#">Instagram</a>
    </aside>
</main>

<footer>
    <p>&copy; 2025 My Blog. All rights reserved.</p>
</footer>
</body>
</html>
```

3. Multi-Column Blog Layout with Footer Navigation

This layout includes multiple columns in the content area, with each column representing different categories or articles. It also has footer navigation links.

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

```
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Multi-Column Blog</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 0; padding: 0;
background-color: #fff; }
    header, footer { background-color: #222; color: white; padding:
1rem; text-align: center; }
    main { display: flex; justify-content: space-between; padding: 20px;
}
    .main-content { flex: 3; }
    .sidebar { flex: 1; padding: 20px; background-color: #f4f4f4; }
    .column { background-color: #fff; padding: 20px; margin: 10px 0;
border-radius: 8px; }
    footer { font-size: 0.9rem; display: flex; justify-content: space-
around; padding: 20px 0; }
    footer a { color: white; text-decoration: none; }
  </style>
</head>
<body>
  <header>
    <h1>My Multi-Column Blog</h1>
    <nav>
      <a href="#">Home</a> | <a href="#">About</a> | <a
href="#">Contact</a>
    </nav>
  </header>

  <main>
    <section class="main-content">
      <section class="column">
        <h2>Latest Articles</h2>
        <article>
          <header>
            <h3>Understanding CSS Flexbox</h3>
            <p><time datetime="2025-05-06">May 6, 2025</time> by <a
href="#">John Doe</a></p>
          </header>
```

```
          <p>Learn the basics of CSS Flexbox and how to use it to create
responsive layouts.</p>
        </article>
      </section>

      <section class="column">
        <h2>Popular Posts</h2>
        <article>
          <header>
            <h3>Exploring CSS Grid</h3>
            <p><time datetime="2025-05-05">May 5, 2025</time> by <a
href="#">Jane Smith</a></p>
          </header>
          <p>CSS Grid is a powerful tool for creating complex layouts.
Learn how to master it!</p>
        </article>
      </section>
    </section>

    <aside class="sidebar">
      <h3>About Me</h3>
      <p>Hello, I am a web developer who loves teaching HTML, CSS, and
JavaScript. Follow my blog for regular updates.</p>
    </aside>
  </main>

  <footer>
    <nav>
      <a href="#">Privacy Policy</a> | <a href="#">Terms of Service</a>
| <a href="#">Contact</a>
    </nav>
  </footer>
</body>
</html>
```

4. Blog Layout with Hero Section and Footer Widgets

This layout features a large hero section with an introduction and an emphasis on a featured article, along with a footer that includes multiple widget areas.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Blog with Hero Section</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 0; padding: 0;
background-color: #f4f4f4; }
    header { background-color: #333; color: white; padding: 2rem; text-
align: center; }
    header h1 { margin: 0; }
    .hero { background-color: #333; color: white; padding: 5rem 0; text-
align: center; }
    .hero h2 { margin: 0; font-size: 2.5rem; }
    main { padding: 20px; display: flex; flex-direction: column; align-
items: center; }
    article { background-color: white; margin: 20px 0; padding: 20px;
border-radius: 8px; width: 80%; max-width: 900px; }
    footer { background-color: #333; color: white; padding: 1rem 0;
text-align: center; font-size: 0.9rem; }
    footer .footer-widgets { display: flex; justify-content: space-
around; padding: 20px 0; }
    footer .widget { width: 30%; }
  </style>
</head>
<body>
  <header>
    <h1>My Blog</h1>
    <nav>
      <a href="#">Home</a> | <a href="#">Categories</a> | <a
href="#">Contact</a>
    </nav>
  </header>
```

```
  <section class="hero">
    <h2>Welcome to My Blog</h2>
    <p>Explore articles on web development, design, and more.</p>
  </section>

  <main>
    <article>
      <header>
        <h2>Featured Article: The Future of Web Development</h2>
        <p><time datetime="2025-05-06">May 6, 2025</time> by <a
href="#">John Doe</a></p>
      </header>
      <p>The future of web development is constantly evolving. Learn
about the new trends that will shape the web in the coming years...</p>
    </article>
  </main>

  <footer>
    <nav>
      <a href="#">Privacy Policy</a> | <a href="#">Terms of Service</a>
| <a href="#">Contact</a>
    </nav>
    <div class="footer-widgets">
      <div class="widget">
        <h3>Categories</h3>
        <ul>
          <li><a href="#">Web Development</a></li>
          <li><a href="#">Design</a></li>
          <li><a href="#">Programming</a></li>
        </ul>
      </div>
      <div class="widget">
        <h3>Follow Us</h3>
        <ul>
          <li><a href="#">Facebook</a></li>
          <li><a href="#">Twitter</a></li>
          <li><a href="#">Instagram</a></li>
```

```
        </ul>
      </div>
    </div>
  </footer>
</body>
</html>
```

HTML Forms

An HTML form is used to collect user input. The user input is most often sent to a server for processing.

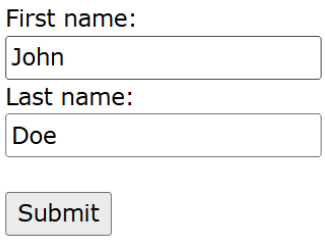

image_2.png

The Element

The HTML `<form>` element is used to create an HTML form for user input:

```
<form>
.
form elements
.
</form>
```

The `<form>` element is a container for different types of input elements, such as: `text fields`, `checkboxes`, `radio buttons`, `submit buttons`, etc.

The Element

The HTML `<input>` element is the most used form element.

An `<input>` element can be displayed in many ways, depending on the type attribute.

Here are some examples:

Type	Description
`<input type="text">`	Displays a single-line text input field
`<input type="radio">`	Displays a radio button (for selecting one of many choices)
`<input type="checkbox">`	Displays a checkbox (for selecting zero or more of many choices)
`<input type="submit">`	Displays a submit button (for submitting the form)
`<input type="button">`	Displays a clickable button

Text Fields

The `<input type="text">` defines a single-line input field for text input. A form with input fields for text:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

This is how the HTML code above will be displayed in a browser:

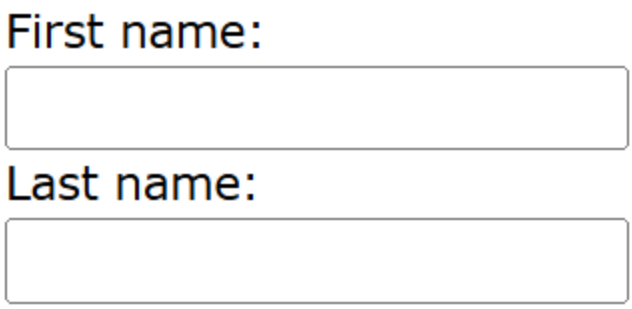

image_3.png

The Element

Notice the use of the `<label>` element in the example above.

The `<label>` tag defines a label for many form elements.

The `<label>` element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focuses on the input element.

The element also helps users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the element, it toggles the radio button/checkbox.

The for attribute of the tag should be equal to the id attribute of the element to bind them together.

Radio Buttons

The defines a radio button.

Radio buttons let a user select ONE of a limited number of choices.

```
<p>Choose your favorite Web language:</p>

<form>
  <input type="radio" id="html" name="fav_language" value="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" value="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_language"
value="JavaScript">
  <label for="javascript">JavaScript</label>
</form>
```

Checkboxes

The `<input type="checkbox">` defines a checkbox.

Checkboxes let a user select ZERO or MORE options of a limited number of choices.

```
<form>
  <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
  <label for="vehicle1"> I have a bike</label><br>
  <input type="checkbox" id="vehicle2" name="vehicle2" value="Car">
  <label for="vehicle2"> I have a car</label><br>
  <input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
  <label for="vehicle3"> I have a boat</label>
</form>
```

The Submit Button

The `<input type="submit">` defines a button for submitting the form data to a form-handler.

The form-handler is typically a file on the server with a script for processing input data.

The form-handler is specified in the form's `action` attribute.

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

This is how the HTML code above will be displayed in a browser:

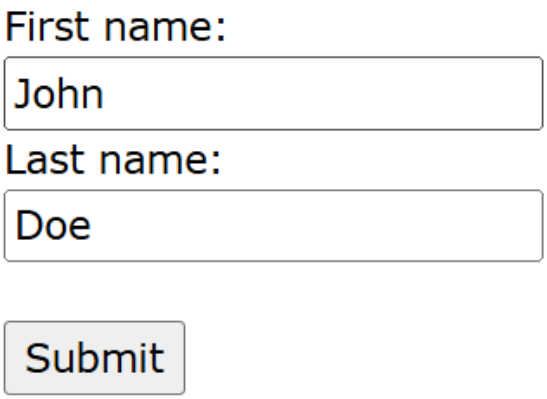

image_4.png

The Name Attribute for

Notice that each input field must have a `name` attribute to be submitted.

If the `name` attribute is omitted, the value of the input field will not be sent at all.

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" value="John"><br><br>
  <input type="submit" value="Submit">
</form>
```

HTML Form Elements

The HTML element can contain one or more of the following form elements:

```
- <input>
- <label>
- <select>
- <textarea>
- <button>
- <fieldset>
- <legend>
- <datalist>
- <output>
- <option>
- <optgroup>
```

The Element:

One of the most used form elements is the `<input>` element.

The `<input>` element can be displayed in several ways, depending on the type attribute.

```
<label for="fname">First name:</label>
<input type="text" id="fname" name="fname">
```

other `<input>` Types are:

```
<input type="button">
<input type="checkbox">
<input type="color">
<input type="date">
<input type="datetime-local">
<input type="email">
<input type="file">
<input type="hidden">
<input type="image">
<input type="month">
```

```
<input type="number">
<input type="password">
<input type="radio">
<input type="range">
<input type="reset">
<input type="search">
<input type="submit">
<input type="tel">
<input type="text">
<input type="time">
<input type="url">
<input type="week">
```

The Element

The `<select>` element defines a drop-down list:

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

To define a pre-selected option, add the `selected` attribute to the option: `<option value="fiat" selected>Fiat</option>`

Visible Values:

Use the `size` attribute to specify the number of visible values:

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars" size="3">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
```

```
  <option value="audi">Audi</option>
</select>
```

Allow Multiple Selections:

Use the `multiple` attribute to allow the user to select more than one value:

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars" size="4" multiple>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

The Element

The element defines a multi-line input field (a text area):

```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

The rows attribute specifies the visible number of lines in a text area.

The cols attribute specifies the visible width of a text area.

This is how the HTML code above will be displayed in a browser:

The Element

The element defines a clickable button:

```
<button type="button" onclick="alert('Hello World!')">Click Me!</button>
```

The and Elements

The element is used to group related data in a form.

The element defines a caption for the element.

```
<form action="/action_page.php">
  <fieldset>
    <legend>Personalia:</legend>
    <label for="fname">First name:</label><br>
    <input type="text" id="fname" name="fname" value="John"><br>
    <label for="lname">Last name:</label><br>
    <input type="text" id="lname" name="lname" value="Doe"><br><br>
    <input type="submit" value="Submit">
  </fieldset>
</form>
```

This is how the HTML code above will be displayed in a browser:

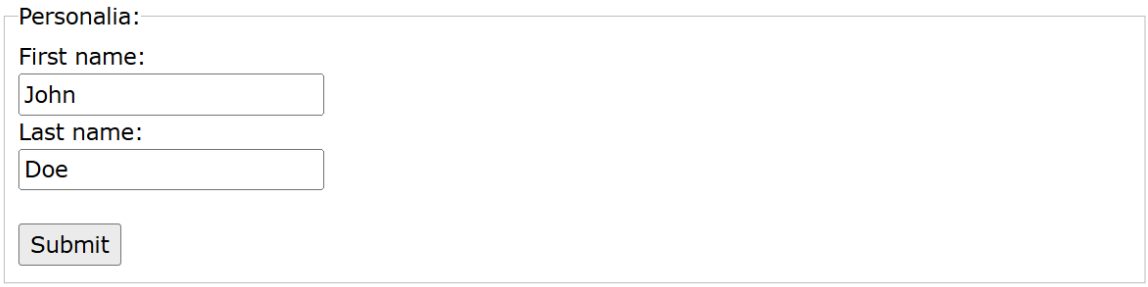

image_5.png

HTML Block VS Inline Elements

Every HTML element has a default display value, depending on what type of element it is.

The two most common display values are block and inline.

Block-level Elements

A block-level element always starts on a new line, and the browsers automatically add some space (a margin) before and after the element.

A block-level element always takes up the full width available (stretches out to the left and right as far as it can).

Two commonly used block elements are:

and .

The

element defines a paragraph in an HTML document.

The element defines a division or a section in an HTML document.

Here are the block-level elements in HTML:

```
- <address>
- <article>
- <aside>
- <blockquote>
- <canvas>
- <dd>
- <div>
- <dl>
- <dt>
- <fieldset>
- <figcaption>
- <figure>
- <footer>
- <form>
- <h1>-<h6>
```

```
- <header>
- <hr>
- <li>
- <main>
- <nav>
- <noscript>
- <ol>
- <p>
- <pre>
- <section>
- <table>
- <tfoot>
- <ul>
- <video>
```

Inline Elements

An inline element does not start on a new line.

An inline element only takes up as much width as necessary. Here are the inline elements in HTML: `<a>` `<abbr>` `<acronym>` `<b>` `<bdo>` `<big>` `<br>` `<button>` `<cite>` `<code>` `<dfn>` `<em>` `<i>` `<img>` `<input>` `<kbd>` `<label>` `<map>` `<object>` `<output>` `<q>` `<samp>` `<script>` `<select>` `<small>` `<span>` `<strong>` `<sub>` `<sup>` `<textarea>` `<time>` `<tt>` `<var>`

Note: An inline element cannot contain a block-level element!

CSS

CSS is the language we use to style a Web page.

What is CSS?

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files

Why Use CSS?

CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes

CSS Syntax

A CSS rule consists of a selector and a declaration block.

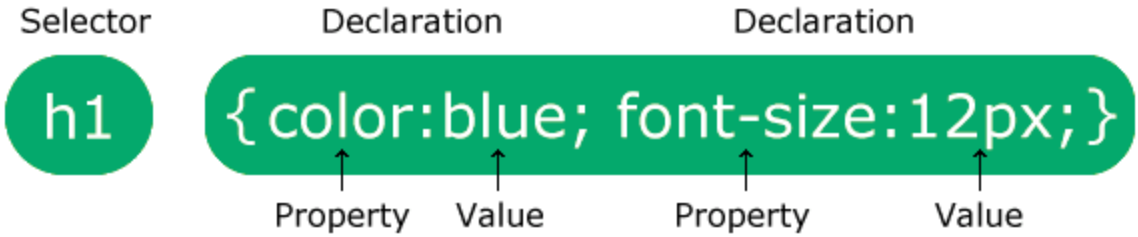

image_6.png

- The selector points to the HTML element you want to style.
- The declaration block contains one or more declarations separated by semicolons.
- Each declaration includes a CSS property name and a value, separated by a colon.

- Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

CSS Icons

How To Add Icons

The simplest way to add an icon to your HTML page, is with an icon library, such as Font Awesome.

Font Awesome Icons

To use the Font Awesome icons, go to fontawesome.com, sign in, and get a code to add in the section of your HTML page:

`<script src="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.7.2/js/all.min.js" crossorigin="anonymous"></script>` ***Note: No downloading or installation is required!***

```
<!DOCTYPE html>
<html>
<head>
    <!-- <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.7.2/css/all.min.css">
         -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.7.2/js/all.min.js"></script>
</head>
<body>

<i class="fas fa-cloud"></i>
<i class="fas fa-heart"></i>
<i class="fas fa-car"></i>
<i class="fas fa-file"></i>
<i class="fas fa-bars"></i>

</body>
</html>
```

Google Icons

To use the Google icons, add the following line inside the section of your HTML page:

`<link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons">`

Note: No downloading or installation is required!

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/icon?
family=Material+Icons">
</head>
<body>

<i class="material-icons">cloud</i>
<i class="material-icons">favorite</i>
<i class="material-icons">attachment</i>
<i class="material-icons">computer</i>
<i class="material-icons">traffic</i>

</body>
</html>
```

CSS Colors

CSS Color Names

Colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values In CSS, a color can be specified by using a predefined color name:

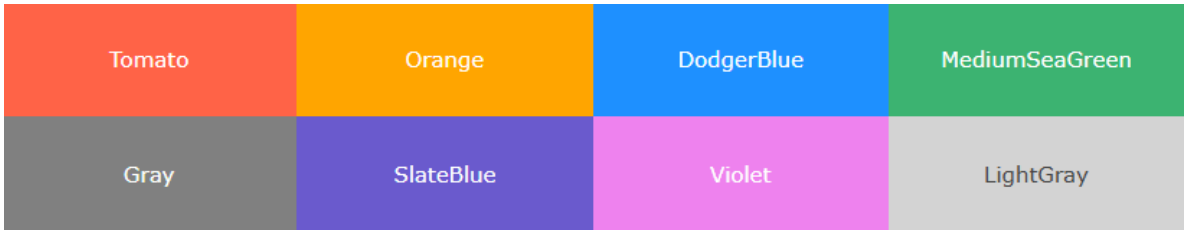

image_7.png

CSS Background Color

You can set the background color for HTML elements:

image_8.png

```
<h1 style="background-color:DodgerBlue;">Hello World</h1>
<p style="background-color:Tomato;">Lorem ipsum...</p>
```

CSS Text Color

You can set the color of text:

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

image_10.png

```
<h1 style="color:Tomato;">Hello World</h1>
<p style="color:DodgerBlue;">Lorem ipsum...</p>
<p style="color:MediumSeaGreen;">Ut wisi enim...</p>
```

CSS Border Color

You can set the color of borders:

Hello World

Hello World

Hello World

image_9.png

```
<h1 style="border:2px solid Tomato;">Hello World</h1>
<h1 style="border:2px solid DodgerBlue;">Hello World</h1>
<h1 style="border:2px solid Violet;">Hello World</h1>
```

CSS Color Values

In CSS, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values:

Same as color name "Tomato":

rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

Same as color name "Tomato", but 50% transparent:

rgba(255, 99, 71, 0.5)

hsla(9, 100%, 64%, 0.5)

image_11.png

```
<h1 style="background-color:rgb(255, 99, 71);">...</h1>
<h1 style="background-color:#ff6347;">...</h1>
<h1 style="background-color:hsl(9, 100%, 64%);">...</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.5);">...</h1>
<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">...</h1>
```

CSS Fonts

Generic Font Families In CSS there are five generic font families:

1. `Serif fonts` have a small stroke at the edges of each letter. They create a sense of formality and elegance.
2. `Sans-serif` fonts have clean lines (no small strokes attached). They create a modern and minimalistic look.
3. `Monospace fonts` - here all the letters have the same fixed width. They create a mechanical look.
4. `Cursive fonts` imitate human handwriting.
5. `Fantasy fonts` are decorative/playful fonts.

Some font examples

Generic Font Family	Examples of Font Names
Serif	Times New Roman Georgia Garamond
Sans-serif	Arial Verdana Helvetica
Monospace	Courier New Lucida Console Monaco
Cursive	Brush Script MT Lucida Handwriting
Fantasy	Copperplate Papyrus

image_14.png

Web Safe Fonts

Fallback Fonts However, there are no 100% completely web safe fonts. There is always a chance that a font is not found or is not installed properly.

Therefore, it is very important to always use fallback fonts.

```
p {
font-family: Tahoma, Verdana, sans-serif;
}
```

Best Web Safe Fonts for HTML and CSS The following list are the best web safe fonts for HTML and CSS:

1. Arial (sans-serif)
2. Verdana (sans-serif)
3. Tahoma (sans-serif)
4. Trebuchet MS (sans-serif)
5. Times New Roman (serif)
6. Georgia (serif)
7. Garamond (serif)
8. Courier New (monospace)
9. Brush Script MT (cursive)

Google Fonts

If you do not want to use any of the standard fonts in HTML, you can use Google Fonts.

Google Fonts are free to use, and have more than 1000 fonts to choose from.

How To Use Google Fonts

```
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?
family=Sofia">
<style>
body {
  font-family: "Sofia", sans-serif;
}
</style>
</head>
```

Use Multiple Google Fonts

To use multiple Google fonts, just separate the font names with a pipe character (|), like this:

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?
family=Audiowide|Sofia|Trirong">
<style>
h1.a {
  font-family: "Audiowide", sans-serif;
}

h1.b {
  font-family: "Sofia", sans-serif;
}

h1.c {
  font-family: "Trirong", serif;
}
</style>
</head>
<body>

<h1 class="a">Audiowide Font</h1>
```

```
<h1 class="b">Sofia Font</h1>

<h1 class="c">Trirong Font</h1>

</body>
</html>
```

Margins Vs Padding

CSS Margins

The CSS `margin` properties are used to create space around elements, outside of any defined borders.

With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (`top, right, bottom, and left`).

CSS has properties for specifying the margin for each side of an element:

- margin-top
- margin-right
- margin-bottom
- margin-left

All the margin properties can have the following values:

- auto - the browser calculates the margin
- length - specifies a margin in px, pt, cm, etc.
- % - specifies a margin in % of the width of the containing element
- inherit - specifies that the margin should be inherited from the parent element

```
p {
    margin-top: 100px;
    margin-bottom: 100px;
    margin-right: 150px;
    margin-left: 80px;
}
```

Margin - Shorthand Property

The margin property is a shorthand property for the following individual margin properties:

```
p {
  margin: 25px 50px 75px 100px;
}
```

- top margin is 25px
- right margin is 50px
- bottom margin is 75px
- left margin is 100px

CSS Padding

Padding is used to create space around an element's content, inside of any defined borders.

Padding - Individual Sides

CSS has properties for specifying the padding for each side of an element:

1. padding-top
2. padding-right
3. padding-bottom
4. padding-left

All the padding properties can have the following values:

- length - specifies a padding in px, pt, cm, etc.
- % - specifies a padding in % of the width of the containing element
- inherit - specifies that the padding should be inherited from the parent element

Padding - Shorthand Property

The padding property is a shorthand property for the following individual padding properties:

```
div {
  padding: 25px 50px 75px 100px;
}
```

- top padding is 25px
- right and left paddings are 50px
- bottom padding is 75px

The CSS Box Model

In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: content, padding, borders and margins. The image below illustrates the box model:

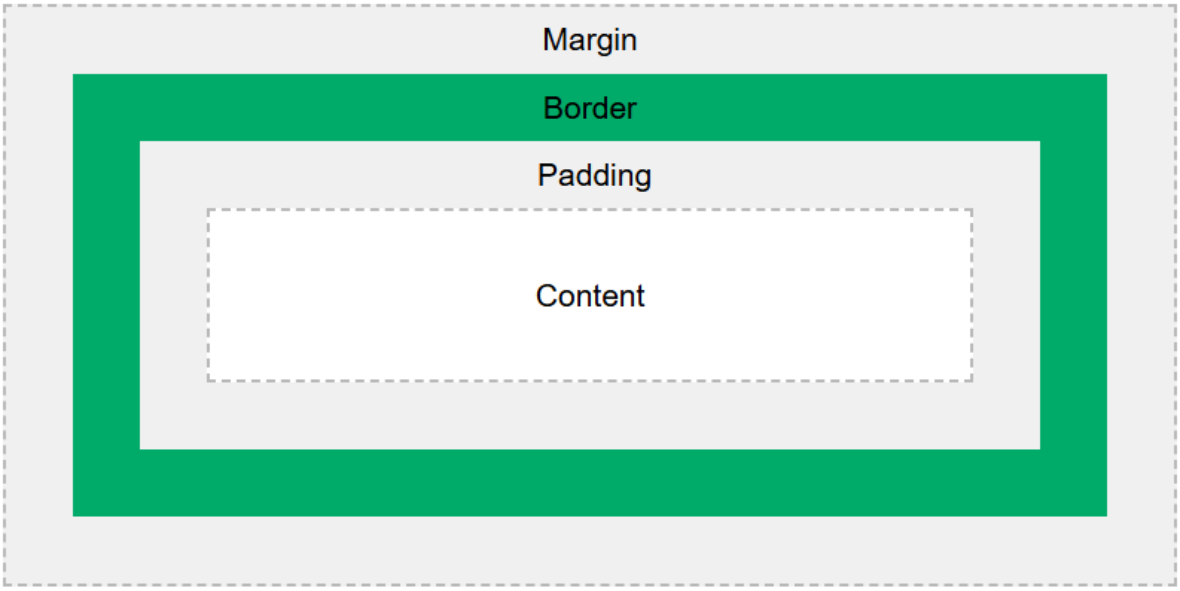

image_13.png

Explanation of the different parts:

- Content - The content of the box, where text and images appear
- Padding - Clears an area around the content. The padding is transparent
- Border - A border that goes around the padding and content
- Margin - Clears an area outside the border. The margin is transparent

CSS Border Style

The `border-style` property specifies what kind of border to display.

The following values are allowed:

- `dotted` - Defines a dotted border
- `dashed` - Defines a dashed border
- `solid` - Defines a solid border
- `double` - Defines a double border
- `groove` - Defines a 3D grooved border. The effect depends on the border-color value
- `ridge` - Defines a 3D ridged border. The effect depends on the border-color value
- `inset` - Defines a 3D inset border. The effect depends on the border-color value
- `outset` - Defines a 3D outset border. The effect depends on the border-color value
- `none` - Defines no border
- `hidden` - Defines a hidden border

The border-style property can have from one to four values (for the top border, right border, bottom border, and the left border).

```
p.dotted {border-style: dotted;}
p.dashed {border-style: dashed;}
p.solid {border-style: solid;}
p.double {border-style: double;}
p.groove {border-style: groove;}
p.ridge {border-style: ridge;}
p.inset {border-style: inset;}
p.outset {border-style: outset;}
p.none {border-style: none;}
p.hidden {border-style: hidden;}
p.mix {border-style: dotted dashed solid double;}
```

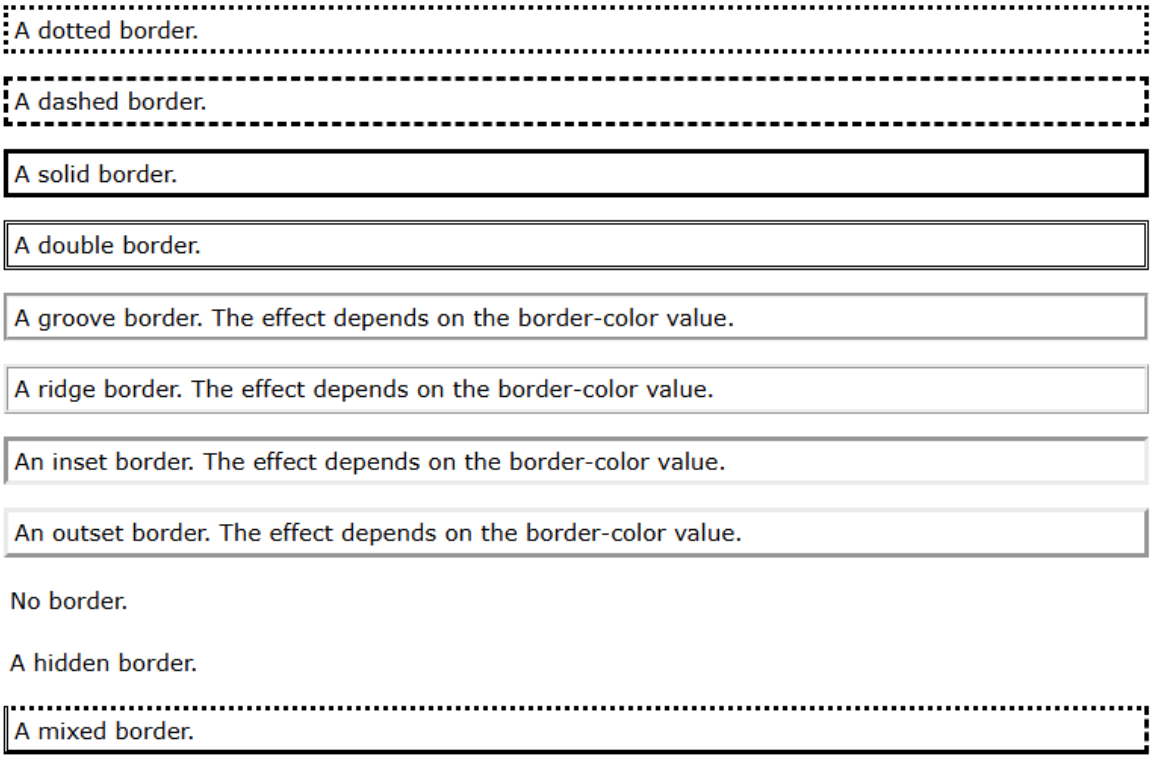

image_12.png

CSS Rounded Borders

```css
p {
  border: 2px solid red;
  border-radius: 5px;
}
```

CSS Border - Shorthand Property

Like you saw in the previous page, there are many properties to consider when dealing with borders.

To shorten the code, it is also possible to specify all the individual border properties in one property.

The `border` property is a shorthand property for the following individual border properties:

1. `border-width`
2. `border-style` (required)
3. `border-color`

Example:

```
p {
  border: 5px solid red;
}
```

Left Border

```
p {
  border-left: 6px solid red;
}
```

Bottom Border

```
p {
  border-bottom: 6px solid red;
}
```

CSS Selectors

A CSS selector selects the HTML element(s) you want to style.

CSS Selectors

CSS selectors are used to "find" (or select) the HTML elements you want to style.

We can divide CSS selectors into five categories:

1. Simple selectors (select elements based on name, id, class)
2. Combinator selectors (select elements based on a specific relationship between them)
3. Pseudo-class selectors (select elements based on a certain state)
4. Pseudo-elements selectors (select and style a part of an element)
5. Attribute selectors (select elements based on an attribute or attribute value)

The CSS element Selector

The element selector selects HTML elements based on the element name.

```
p {
  text-align: center;
  color: red;
}
```

The CSS id Selector

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

```
#para1 {
  text-align: center;
  color: red;
}
```

Note: An id name cannot start with a number!

The CSS class Selector

The class selector selects HTML elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the class name.

```
.center {
  text-align: center;
  color: red;
}
```

The CSS Universal Selector

The universal selector (*) selects all HTML elements on the page.

```
* {
  text-align: center;
  color: blue;
}
```

CSS Combinators

A CSS selector can contain more than one simple selector. Between the simple selectors, we can include a combinator.

There are four different combinators in CSS:

1. Descendant combinator (space)
2. Child combinator (>)
3. Next sibling combinator (+)
4. Subsequent-sibling combinator (~)

Descendant Combinator

The descendant combinator matches all elements that are descendants of a specified element.

The following example selects all

elements inside elements:

```
div p {
  background-color: yellow;
}
```

Child Combinator (>)

The child combinator selects all elements that are the children of a specified element.

The following example selects all

elements that are children of a element:

```
div > p {
  background-color: yellow;
}
```

Next Sibling Combinator (+)

The next sibling combinator is used to select an element that is directly after another specific element.

Sibling elements must have the same parent element, and "adjacent" means "immediately following".

The following example selects the first

element that are placed immediately after elements:

```
div + p {
  background-color: yellow;
}
```

Subsequent-sibling Combinator (~)

The subsequent-sibling combinator selects all elements that are next siblings of a specified element.

The following example selects all

elements that are next siblings of elements:

```
div ~ p {
  background-color: yellow;
}
```

Demo Example

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

```
/* General styles */
body {
    font-family: Arial, sans-serif;
    margin: 20px;
    background-color: #f4f4f4;
}

h1, h2 {
    color: #333;
}

p {
    color: #555;
    padding: 10px;
}

div {
    padding: 20px;
    background-color: #e0e0e0;
    margin: 10px 0;
}

/* Descendant Combinator */
div p {
    background-color: yellow;
}

/* Child Combinator */
div > p {
    background-color: lightblue;
}

/* Next Sibling Combinator */
div + p {
    background-color: lightgreen;
}

/* Subsequent Sibling Combinator */
```

```
        div ~ p {
            background-color: lightcoral;
        }
    </style>
</head>
<body>
<h1>CSS Combinators</h1>
<h2>Descendant Combinator</h2>
<p>The following <code>div p</code> selects all <code>&lt;p&gt;</code>
elements inside <code>&lt;div&gt;</code> elements:</p>
<div>
    <p>This paragraph is inside a div and is selected by the descendant
combinator.</p>
</div>
<p>This paragraph is not inside a div and will not be affected by the
descendant combinator.</p>

<h2>Child Combinator</h2>
<p>The following <code>div &gt; p</code> selects all <code>&lt;p&gt;
</code> elements that are direct children of a <code>&lt;div&gt;</code>
element:</p>
<div>
    <p>This paragraph is a child of a div and is selected by the child
combinator.</p>
    <div>
        <p>This paragraph is inside a nested div and will not be
affected by the child combinator.</p>
    </div>
</div>

<h2>Next Sibling Combinator</h2>
<p>The following <code>div + p</code> selects the first <code>&lt;p&gt;
</code> element that is placed immediately after a <code>&lt;div&gt;
</code> element:</p>
<div>This is a div element.</div>
<p>This paragraph is immediately after the div and is selected by the
next sibling combinator.</p>
<p>This paragraph is not directly after the div, so it will not be
```

```
affected by the next sibling combinator.</p>

<h2>Subsequent Sibling Combinator</h2>
<p>The following <code>div ~ p</code> selects all <code>&lt;p&gt;</code>
elements that are siblings of a <code>&lt;div&gt;</code> element:</p>
<div>This is a div element.</div>
<p>This paragraph is a sibling of the div and is selected by the
subsequent sibling combinator.</p>
<p>This paragraph is also a sibling of the div and will be selected by
the subsequent sibling combinator.</p>
<div>This is another div element.</div>
<p>This paragraph is after the second div and is selected as a
subsequent sibling.</p>
</body>
</html>
```

Summary

- Descendant Combinator: The first paragraph inside the `<div>` has a `yellow background`.
- Child Combinator: The first `<p>` inside the first `<div>` has a `light blue background`, but the nested `<p>` inside another `<div>` is *unaffected*.
- Next Sibling Combinator: The first `<p>` immediately after a `<div>` gets a `light green background`, but *other paragraphs are unaffected*.
- Subsequent Sibling Combinator: All paragraphs that are siblings of a `<div>` get a `light coral background`, regardless of whether they are directly after the `<div>`.

CSS Pseudo-classes

What are Pseudo-classes?

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

1. Style an element when a user moves the mouse over it
2. Style visited and unvisited links differently
3. Style an element when it gets focus
4. Style valid/invalid/required/optional form elements

Syntax

The syntax of pseudo-classes:

```
selector:pseudo-class {
  property: value;
}
```

Anchor Pseudo-classes

Links can be displayed in different ways:

```
/* unvisited link */
a:link {
  color: #FF0000;
}

/* visited link */
a:visited {
  color: #00FF00;
}
```

```css
/* mouse over link */
a:hover {
  color: #FF00FF;
}

/* selected link */
a:active {
  color: #0000FF;
}
```

Pseudo-classes and HTML Classes

Pseudo-classes can be combined with HTML classes:

Example:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>CSS Pseudo-classes Example</title>
    <style>
        /* General styles */
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
            background-color: #f4f4f4;
        }

        h1, h2 {
            color: #333;
        }

        p {
            color: #555;
```

```
}

/* Basic link styles */
a {
    text-decoration: none;
    font-weight: bold;
    font-size: 16px;
}

/* Anchor Pseudo-classes */
/* unvisited link */
a:link {
    color: #FF0000;
}

/* visited link */
a:visited {
    color: #00FF00;
}

/* mouse over link */
a:hover {
    color: #FF00FF;
}

/* selected link */
a:active {
    color: #0000FF;
}

/* Pseudo-classes with HTML classes */
.button:hover {
    background-color: #4CAF50;
    color: white;
}

.input-field:focus {
    background-color: #e0f7fa;
```

```
            border: 2px solid #00acc1;
        }

        .required:invalid {
            background-color: #ffebee;
            border: 2px solid #f44336;
        }

        .required:valid {
            background-color: #c8e6c9;
            border: 2px solid #4caf50;
        }

        .input-field {
            padding: 8px;
            margin: 10px 0;
            width: 200px;
            border: 1px solid #ddd;
        }

        .button {
            padding: 10px 15px;
            background-color: #f0f0f0;
            border: 1px solid #ccc;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <h1>CSS Pseudo-classes</h1>
    <h2>What are Pseudo-classes?</h2>
    <p>A pseudo-class is used to define a special state of an element.
</p>
    <ul>
        <li>Style an element when a user moves the mouse over it</li>
        <li>Style visited and unvisited links differently</li>
        <li>Style an element when it gets focus</li>
        <li>Style valid/invalid/required/optional form elements</li>
```

```
    </ul>

    <h2>Anchor Pseudo-classes</h2>
    <p>Links can be displayed in different ways:</p>
    <ul>
        <li><a href="#">Unvisited Link</a></li>
        <li><a href="#" class="visited">Visited Link</a></li>
    </ul>

    <h2>Combining Pseudo-classes with HTML Classes</h2>
    <form>
        <input type="text" class="input-field required"
placeholder="Enter your name" required>
        <input type="text" class="input-field" placeholder="Optional
input">
        <button type="submit" class="button">Submit</button>
    </form>
</body>
</html>
```

CSS Pseudo-elements

A CSS pseudo-element is used to style specific parts of an element.

For example, it can be used to:

- Style the first letter or line, of an element
- Insert content before or after an element
- Style the markers of list items
- Style the viewbox behind a dialog box

The ::first-line Pseudo-element

The ::first-line pseudo-element is used to add a special style to the first line of a text.

```
p::first-line {
  color: #ff0000;
  font-variant: small-caps;
}
```

Note: The ::first-line pseudo-element can only be applied to block-level elements.

The following properties apply to the `::first-line` pseudo-element:

- font properties
- color properties
- background properties
- word-spacing
- letter-spacing
- text-decoration
- vertical-align

- text-transform
- line-height
- clear

The following example formats the first line of the text in all

elements:

The ::first-letter Pseudo-element

The ::first-letter pseudo-element is used to add a special style to the first letter of a text.

The following example formats the first letter of the text in all

elements:

```
p::first-letter {
  color: #ff0000;
  font-size: xx-large;
}
```

Note: The ::first-letter pseudo-element can only be applied to block-level elements.

CSS - The ::before Pseudo-element

The `::before` pseudo-element can be used to insert some content before the content of an element.

The following example inserts an image before the content of each `<h1>` element:

```
h1::before {
  content: url(smiley.gif);
}
```

CSS - The ::after Pseudo-element

The `::after` pseudo-element can be used to insert some content after the content of an element.

The following example inserts an image after the content of each `<h1>` element:

```
h1::after {
  content: url(smiley.gif);
}
```

Learn CSS Pseudo-elements with below Example

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Learn CSS Pseudo-elements | Blog Example</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            background-color: #f7f7f7;
            color: #333;
        }

        header {
            background-color: #333;
            color: white;
            padding: 20px;
            text-align: center;
        }

        header h1 {
            font-size: 2.5em;
        }

        .content {
            width: 80%;
            max-width: 1000px;
```

```
        margin: 20px auto;
        background-color: #fff;
        padding: 20px;
        border-radius: 8px;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    }

    h2 {
        color: #444;
    }

    p {
        color: #555;
        font-size: 16px;
        line-height: 1.6;
        margin-bottom: 20px;
    }

    ul {
        list-style: none;
        padding-left: 0;
    }

    ul li {
        margin-bottom: 10px;
        font-size: 16px;
    }

    code {
        background-color: #f9f9f9;
        padding: 5px 10px;
        border-radius: 4px;
        font-size: 14px;
    }

    /* Pseudo-element examples */
    .post p::first-letter {
        color: #ff6347;
```

```
            font-size: 2em;
            font-weight: bold;
        }

        .post p::first-line {
            color: #4682b4;
            font-variant: small-caps;
        }

        h1::before {
            content: "📖 ";
            font-size: 1.5em;
        }

        h1::after {
            content: " 📖";
            font-size: 1.5em;
        }

        .post ul::marker {
            color: #ff6347;
            font-size: 20px;
        }

        ::selection {
            background-color: #ff6347;
            color: white;
        }

        /* Example with class-specific pseudo-element */
        .highlight::first-letter {
            color: #32cd32;
            font-size: 2em;
        }
    </style>
</head>
<body>
```

```
<header>
    <h1>Learn CSS Pseudo-elements - Blog Example</h1>
</header>

<div class="content">
    <h2>What are CSS Pseudo-elements?</h2>
    <p>CSS pseudo-elements are used to style specific parts of an element. They allow you to manipulate the content of an element without changing the underlying HTML structure. Below are a few examples showing how pseudo-elements can be applied in a blog post.</p>

    <h2>Example 1: ::first-letter</h2>
    <p class="post">This is a blog post where the first letter of the first paragraph is styled with the <code>::first-letter</code> pseudo-element. Notice how the first letter of this paragraph is red and larger in size.</p>

    <h2>Example 2: ::first-line</h2>
    <p class="post">This is another paragraph where the <code>::first-line</code> pseudo-element applies a different style to the first line. The first line is now blue and in small caps. This shows how you can target the first line of text inside a block-level element.</p>

    <h2>Example 3: ::before and ::after</h2>
    <p>The <code>::before</code> and <code>::after</code> pseudo-elements are used to insert content before or after an element. In the example below, the heading has a book emoji before and after the text.
</p>
    <h1>CSS Pseudo-elements in Practice</h1>

    <h2>Example 4: ::marker</h2>
    <p>The <code>::marker</code> pseudo-element is used to style the markers (bullets or numbers) of list items. In this example, the list items are styled with a custom color for their markers.</p>
    <ul>
        <li>Learn about pseudo-elements</li>
        <li>Style specific parts of an element</li>
        <li>Insert content with ::before and ::after</li>
```

```
    </ul>

    <h2>Example 5: ::selection</h2>
    <p>The <code>::selection</code> pseudo-element styles the portion of text selected by the user. Try selecting this paragraph, and you'll see the background color changes to a tomato red.</p>
    <p>Try highlighting this text to see the <code>::selection</code> effect in action. The background will turn red with white text when selected!</p>

    <h2>Combining Pseudo-elements with HTML Classes</h2>
    <p class="highlight">This paragraph has a special class <code>highlight</code>, and the first letter is styled differently using the <code>::first-letter</code> pseudo-element, making it green and larger.</p>
</div>

</body>
</html>
```

Summary

1. Header: The header uses ::before and ::after to add a book emoji (📖) before and after the title of the blog post, making the title more engaging.
2. Pseudo-elements Applied to Paragraphs: ::first-letter: In the first example, the first letter of the paragraph is styled with a large, bold red font to give a distinctive look to the beginning of each post.

::first-line: In the second example, the first line of text in the paragraph is styled with a blue color and small-caps to create visual interest.

3. List Markers with ::marker: The list markers (bullet points) in the unordered list are styled with the ::marker pseudo-element, changing their color to red for a more unique look.

4. Selection with ::selection: The ::selection pseudo-element is applied to style the portion of the text that the user selects with their mouse or keyboard. In this case, it changes the background to a tomato red and the text color to white when selected.

5. Combining Pseudo-elements with Classes: The paragraph with the highlight class demonstrates how pseudo-elements can be applied in conjunction with specific classes. The first letter of this paragraph is made green and larger using the ::first-letter pseudo-element, highlighting its distinct style.

CSS Attribute Selectors

It is possible to style HTML elements that have specific attributes or attribute values.

Demo Example

Below is a simple school website designed to teach CSS Attribute Selectors. The page contains examples of how attribute selectors work in CSS, with various interactive and practical examples based on a school-themed site.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Learn CSS Attribute Selectors | School Website</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #fafafa;
            margin: 0;
            padding: 0;
            color: #333;
        }

        header {
            background-color: #4CAF50;
            color: white;
            padding: 20px;
            text-align: center;
        }

        header h1 {
            font-size: 2.5em;
        }
```

```
        .content {
            width: 80%;
            max-width: 1000px;
            margin: 20px auto;
            padding: 20px;
            background-color: white;
            border-radius: 8px;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
        }

        h2 {
            color: #333;
            margin-bottom: 10px;
        }

        p {
            color: #555;
            font-size: 16px;
            line-height: 1.6;
        }

        code {
            background-color: #f4f4f4;
            padding: 5px 10px;
            border-radius: 4px;
            font-size: 14px;
        }

        .student-list {
            list-style: none;
            padding-left: 0;
        }

        .student-list li {
            margin: 10px 0;
            font-size: 16px;
            padding: 8px;
            background-color: #f1f1f1;
```

```
            border-radius: 4px;
        }

        .student-list li.highlight {
            background-color: #ffeb3b;
        }

        .student-list li[class*="grade-A"] {
            background-color: #81c784;
            color: white;
        }

        .student-list li[data-attendance="100%"] {
            background-color: #64b5f6;
            color: white;
        }

        a {
            color: #4CAF50;
            text-decoration: none;
        }

        a:hover {
            text-decoration: underline;
        }

        .example {
            background-color: #f9f9f9;
            padding: 10px;
            border-radius: 6px;
            margin: 10px 0;
        }

        .example code {
            display: block;
            margin-top: 10px;
        }
    </style>
```

```
</head>
<body>

<header>
    <h1>Learn CSS Attribute Selectors - School Website</h1>
</header>

<div class="content">
    <h2>What are CSS Attribute Selectors?</h2>
    <p>CSS attribute selectors allow you to target HTML elements based on the presence or value of their attributes. They are extremely useful when you want to apply styles to specific elements without needing to use classes or IDs.</p>

    <p>Attribute selectors use the following syntax:</p>
    <code>selector[attribute="value"] { property: value; }</code>

    <h2>Basic Attribute Selectors</h2>
    <p>Here are some examples of how to use attribute selectors to target elements in your HTML:</p>

    <div class="example">
        <h3>1. Select Elements with a Specific Attribute</h3>
        <p>To select all elements with a specific attribute:</p>
        <code>input[type="text"] { background-color: lightyellow; }
</code>
        <p>This will apply a light yellow background to all text input fields.</p>
    </div>

    <h2>Example 1: School Student List</h2>
    <p>In the student list below, we'll use attribute selectors to style students based on their grades and attendance:</p>

    <ul class="student-list">
        <li class="grade-A" data-attendance="100%">John Doe - Grade A - 100% Attendance</li>
        <li class="grade-B" data-attendance="90%">Jane Smith - Grade B -
```

```
90% Attendance</li>
        <li class="grade-A" data-attendance="100%">Alice Johnson - Grade
A - 100% Attendance</li>
        <li class="grade-C" data-attendance="85%">Bob Brown - Grade C -
85% Attendance</li>
        <li class="grade-A" data-attendance="95%">Eve White - Grade A -
95% Attendance</li>
    </ul>

    <h2>Using Attribute Selectors to Style Students</h2>
    <div class="example">
        <h3>2. Select Elements Based on Partial Attribute Values</h3>
        <p>To select elements with an attribute value that contains a
certain substring:</p>
        <code>li[class*="grade-A"] { font-weight: bold; color: green; }
</code>
        <p>This targets all elements whose class contains "grade-A"
(i.e., students with grade A) and styles them with bold green text.</p>
    </div>

    <div class="example">
        <h3>3. Select Elements with Specific Data Attributes</h3>
        <p>To select all elements with a specific data attribute value:
</p>
        <code>li[data-attendance="100%"] { background-color: lightblue;
}</code>
        <p>This will highlight students with 100% attendance by applying
a light blue background.</p>
    </div>

    <h2>Advanced Attribute Selectors</h2>
    <p>Attribute selectors are not limited to exact matches. You can use
various types of matching based on the attribute value:</p>

    <ul>
        <li><code>selector[attribute="value"]</code>: Matches elements
with the exact value of the attribute.</li>
        <li><code>selector[attribute*="value"]</code>: Matches elements
```

```
where the attribute contains the specified substring.</li>
        <li><code>selector[attribute^="value"]</code>: Matches elements
where the attribute starts with the specified value.</li>
        <li><code>selector[attribute$="value"]</code>: Matches elements
where the attribute ends with the specified value.</li>
        <li><code>selector[attribute~="value"]</code>: Matches elements
where the attribute contains a space-separated list of values, and one
of them is the specified value.</li>
    </ul>

    <h2>Conclusion</h2>
    <p>CSS attribute selectors provide a powerful way to target elements
based on their attributes, making your styles more flexible and reducing
the need for extra classes or IDs. Try experimenting with these
selectors on your own projects to see how they can make your CSS more
dynamic!</p>
</div>

</body>
</html>
```

The Structure:

1. **Header**:
   - The header introduces the website and the concept of CSS attribute selectors with a school theme. It includes a green background to match the educational theme.
2. **Content Section**:
   - This section explains CSS attribute selectors, with examples of how they can be applied to real-world scenarios, like styling students based on their grades or attendance.
3. **Attribute Selector Examples**:
   - **Basic Attribute Selector**: The `input[type="text"]` selector is used to style text input fields by changing their background color.

- **Student List**: A list of students is presented with classes that represent their grades (A, B, C) and data attributes representing their attendance.
- The list items are styled based on their class and data attributes using various CSS attribute selectors:
  - `[class*="grade-A"]`: Selects students with "grade-A" in their class name and applies bold green text to them.
  - `[data-attendance="100%"]`: Highlights students with 100% attendance by changing their background color.

4. **Advanced Attribute Selectors**:

- The page introduces more complex attribute selectors such as `^=`, `$=`, `*=`, and `~=`, explaining how they work with examples.
- This section demonstrates the versatility of CSS attribute selectors for matching attribute values in different ways.

CSS Media Queries

The `@media` rule, introduced in `CSS2`, made it possible to define different style rules for different media types. Media queries can be used to check many things, such as:

1. width and height of the viewport
2. orientation of the viewport (`landscape or portrait`)
3. resolution

CSS Media Types

Value	Description
all	Used for all media type devices
print	Used for print preview mode
screen	Used for computer screens, tablets, smart-phones etc.

CSS Common Media Features

Here are some commonly used media features:

Value	Description
orientation	Orientation of the viewport. Landscape or portrait
max-height	Maximum height of the viewport
min-height	Minimum height of the viewport
height	Height of the viewport (including scrollbar)
max-width	Maximum width of the viewport
min-width	Minimum width of the viewport
width	Width of the viewport (including scrollbar)

Media Query Syntax

```
@media not|only mediatype and (media feature) and (media feature) {
  CSS-Code;
}
```

Responsive Blogging site

- desktop view

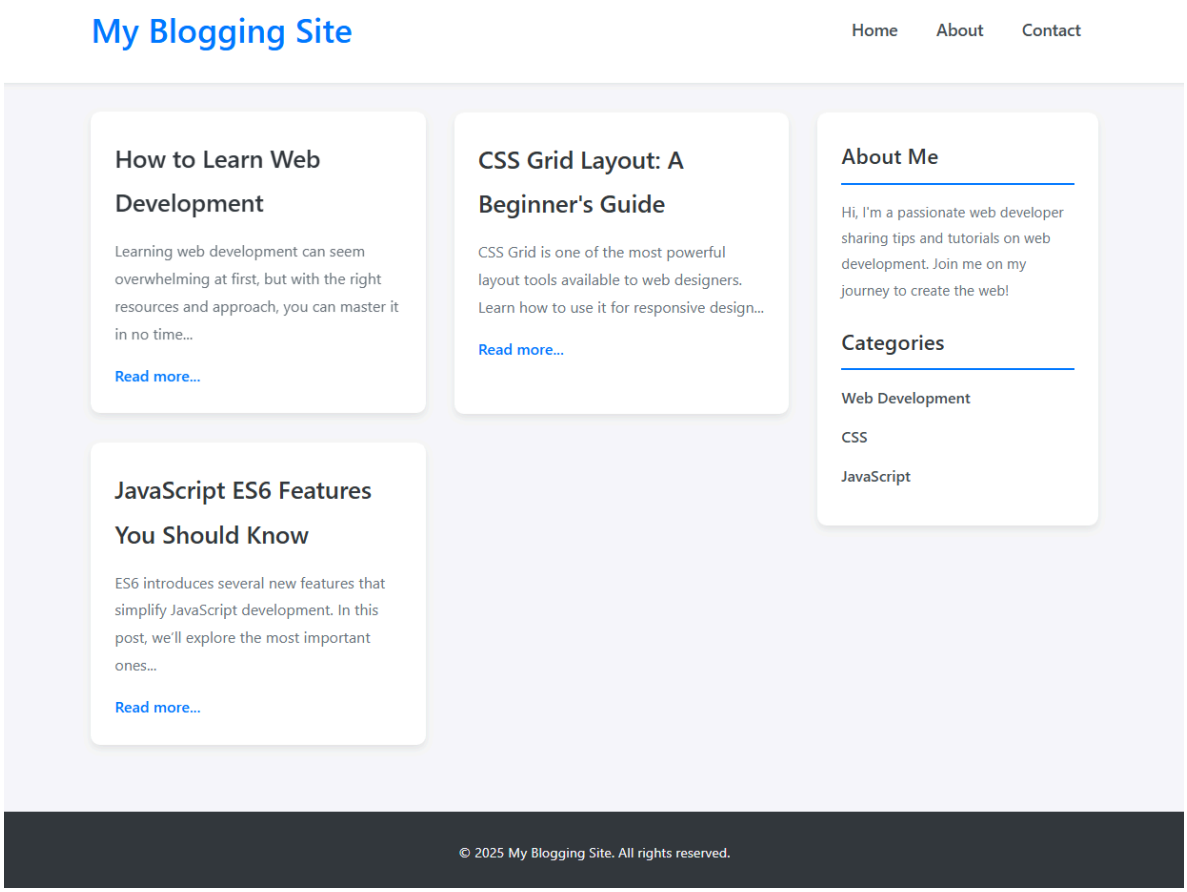

image_15.png

- tablet view

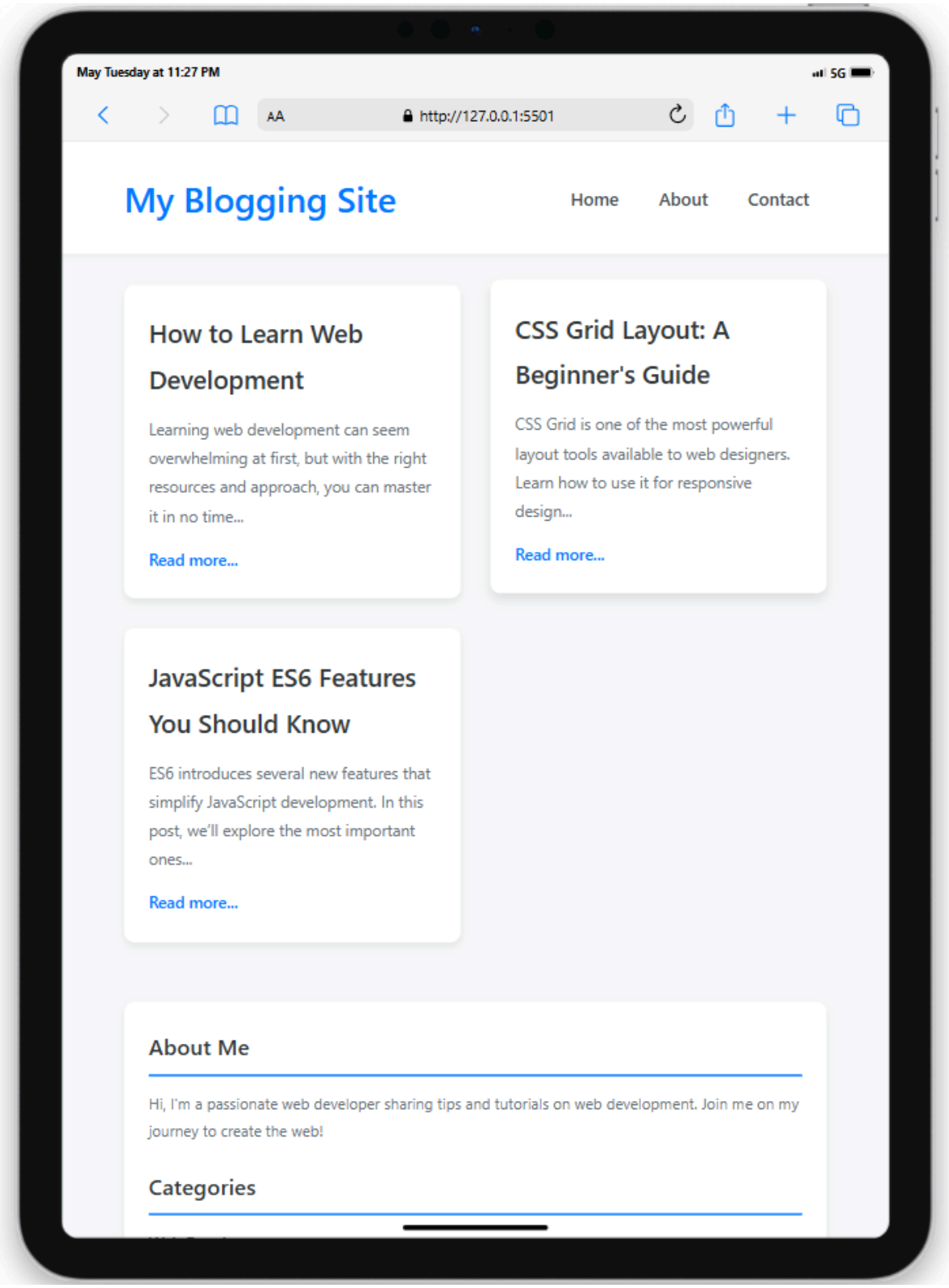

image_17.png

- mobile view

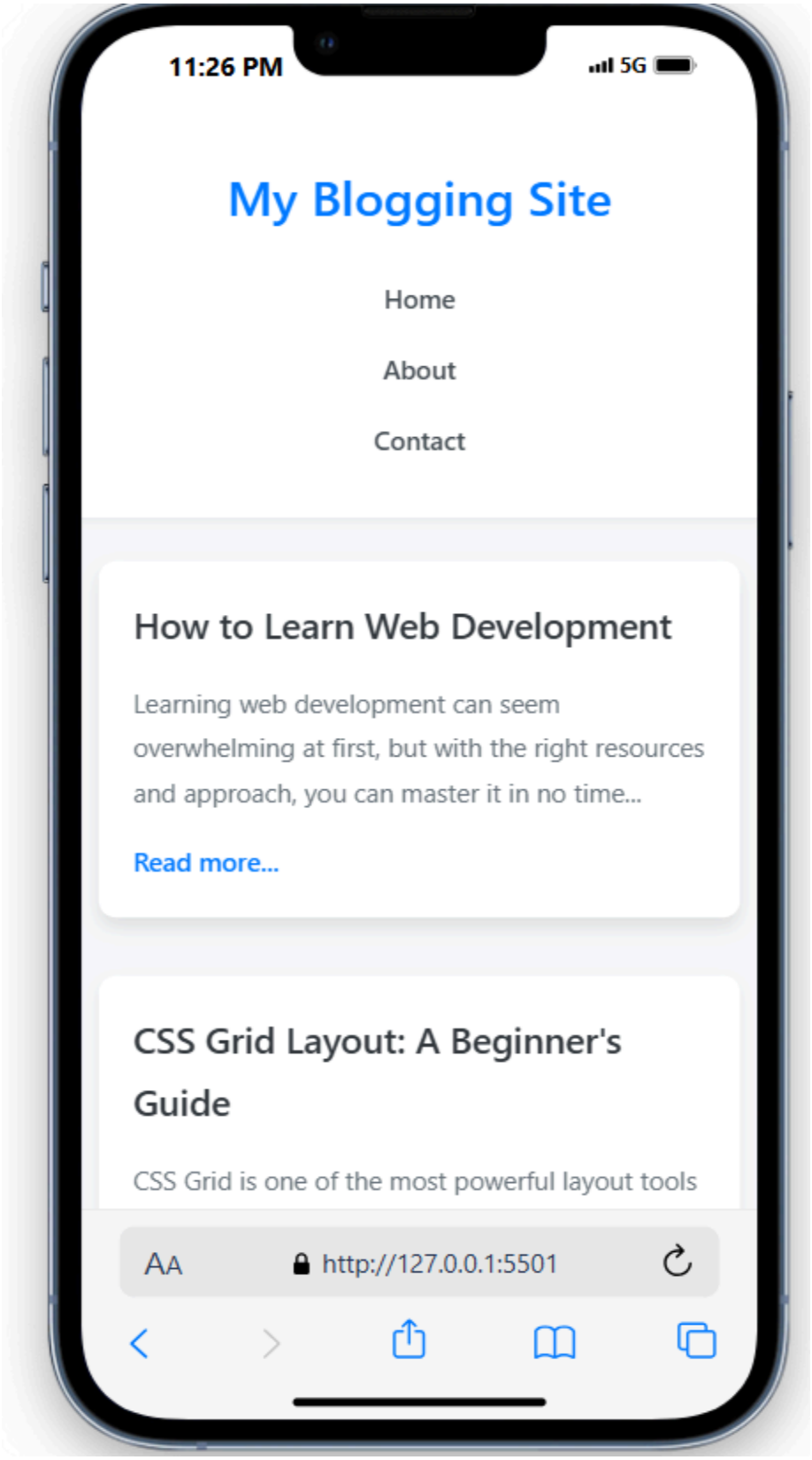
11:26 PM
5G
My Blogging Site
Home
About
Contact
How to Learn Web Development
Learning web development can seem overwhelming at first, but with the right resources and approach, you can master it in no time...
Read more...
CSS Grid Layout: A Beginner's Guide
CSS Grid is one of the most powerful layout tools
AA
http://127.0.0.1:5501

image_16.png

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Responsive Blogging Site</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

<header>
    <div class="container">
        <h1>My Blogging Site</h1>
        <nav>
            <a href="#home">Home</a>
            <a href="#about">About</a>
            <a href="#contact">Contact</a>
        </nav>
    </div>
</header>

<main>
    <div class="container">
        <section class="blog-posts">
            <article class="post">
                <h2>How to Learn Web Development</h2>
                <p>Learning web development can seem overwhelming at
first, but with the right resources and approach, you can master it in
no time...</p>
                <a href="#">Read more...</a>
            </article>
            <article class="post">
                <h2>CSS Grid Layout: A Beginner's Guide</h2>
                <p>CSS Grid is one of the most powerful layout tools
available to web designers. Learn how to use it for responsive design...
</p>
```

```
                <a href="#">Read more...</a>
            </article>
            <article class="post">
                <h2>JavaScript ES6 Features You Should Know</h2>
                <p>ES6 introduces several new features that simplify
JavaScript development. In this post, we'll explore the most important
ones...</p>
                <a href="#">Read more...</a>
            </article>
        </section>

        <aside class="sidebar">
            <h3>About Me</h3>
            <p>Hi, I'm a passionate web developer sharing tips and
tutorials on web development. Join me on my journey to create the web!
</p>
            <h3>Categories</h3>
            <ul>
                <li><a href="#">Web Development</a></li>
                <li><a href="#">CSS</a></li>
                <li><a href="#">JavaScript</a></li>
            </ul>
        </aside>
    </div>
</main>

<footer>
    <div class="container">
        <p>&copy; 2025 My Blogging Site. All rights reserved.</p>
    </div>
</footer>

</body>
</html>
```

```
/* General Styles */
* {
```

```
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    /* Modern sans-serif font */
    line-height: 1.8;
    /* Increased line height for readability */
    background-color: #f8f9fa;
    /* Lighter background */
    color: #343a40;
    /* Darker text for better contrast */
}

.container {
    width: 85%;
    /* Slightly narrower container for better focus */
    max-width: 1200px;
    /* Max width for large screens */
    margin: 0 auto;
}

/* Header */
header {
    background-color: #ffffff;
    /* White header */
    color: #343a40;
    padding: 25px 0;
    /* Increased padding */
    border-bottom: 1px solid #e9ecef;
    /* Subtle border */
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
    /* Subtle shadow */
}

header .container {
```

```
    display: flex;
    justify-content: space-between;
    align-items: center;
}

header h1 {
    font-size: 2.2em;
    /* Slightly reduced for a cleaner look */
    color: #007bff;
    /* Accent color for the title */
    font-weight: 600;
}

header nav a {
    color: #495057;
    /* Softer link color */
    text-decoration: none;
    margin: 0 18px;
    /* Increased margin */
    font-size: 1.1em;
    /* Slightly adjusted font size */
    font-weight: 500;
    transition: color 0.3s ease;
    /* Smooth transition for hover */
}

header nav a:hover,
header nav a.active {
    /* Added active class styling */
    color: #007bff;
    /* Accent color on hover/active */
    text-decoration: none;
    /* Removed underline */
}

/* Main Content */
main {
    margin-top: 30px;
```

```
    /* Increased margin */
    padding-bottom: 30px;
    /* Increased margin */
}

main>.container {
    /* Targeting the main container for flex layout */
    display: flex;
    gap: 30px;
    /* Gap between blog posts and sidebar */
}

.blog-posts {
    flex: 3;
    /* Takes up more space */
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
    /* Responsive grid */
    gap: 30px;
    /* Increased gap */
}

.post {
    background-color: #ffffff;
    padding: 25px;
    /* Increased padding */
    border-radius: 10px;
    /* More rounded corners */
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.08);
    /* Softer, more modern shadow */
    transition: transform 0.3s ease, box-shadow 0.3s ease;
    /* Smooth transition for hover */
}

.post:hover {
    transform: translateY(-5px);
    /* Lift effect on hover */
    box-shadow: 0 6px 12px rgba(0, 0, 0, 0.1);
```

```
    /* Enhanced shadow on hover */
}

.post h2 {
    margin-bottom: 15px;
    /* Increased margin */
    font-size: 1.6em;
    /* Adjusted font size */
    color: #343a40;
    font-weight: 600;
}

.post p {
    margin-bottom: 15px;
    /* Increased margin */
    font-size: 1em;
    color: #6c757d;
    /* Softer paragraph text color */
}

.post a {
    color: #007bff;
    /* Accent color for links */
    text-decoration: none;
    font-weight: 500;
    transition: color 0.3s ease;
}

.post a:hover {
    color: #0056b3;
    /* Darker accent on hover */
    text-decoration: underline;
}

.sidebar {
    flex: 1;
    /* Takes up less space */
    background-color: #ffffff;
```

```
    padding: 25px;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.08);
    height: fit-content;
    /* Sidebar height adjusts to content */
}

.sidebar h3 {
    margin-bottom: 15px;
    font-size: 1.4em;
    color: #343a40;
    font-weight: 600;
    border-bottom: 2px solid #007bff;
    /* Accent border for sidebar titles */
    padding-bottom: 8px;
}

.sidebar p {
    /* Added styling for sidebar paragraph */
    font-size: 0.95em;
    color: #6c757d;
    margin-bottom: 20px;
}

.sidebar ul {
    list-style: none;
}

.sidebar ul li {
    margin: 12px 0;
    /* Increased margin */
}

.sidebar ul li a {
    text-decoration: none;
    color: #495057;
    font-weight: 500;
    transition: color 0.3s ease;
```

```
}

.sidebar ul li a:hover {
    color: #007bff;
    text-decoration: none;
}

/* Footer */
footer {
    background-color: #343a40;
    /* Darker footer */
    color: #f8f9fa;
    /* Lighter text for footer */
    padding: 30px 0;
    /* Increased padding */
    text-align: center;
    margin-top: 40px;
    /* Added margin top for separation */
}

footer p {
    font-size: 0.9em;
}

/* Media Queries */

/* For tablets (max-width: 992px) - Adjusted breakpoint */
@media (max-width: 992px) {
    main>.container {
        flex-direction: column;
        /* Stack blog posts and sidebar */
    }

    .sidebar {
        margin-top: 30px;
        /* Add space when stacked */
    }
}
```

```css
/* For smaller tablets and large mobiles (max-width: 768px) */
@media (max-width: 768px) {
    header .container {
        flex-direction: column;
        text-align: center;
    }

    header h1 {
        margin-bottom: 15px;
    }

    header nav {
        margin-top: 0;
        /* Reset margin */
        display: flex;
        flex-wrap: wrap;
        /* Allow nav items to wrap */
        justify-content: center;
    }

    header nav a {
        margin: 8px 10px;
        /* Adjust margin for wrapped items */
    }

    .blog-posts {
        grid-template-columns: 1fr;
        /* Single column for posts */
    }
}

/* For mobile screens (max-width: 480px) */
@media (max-width: 480px) {
    body {
        font-size: 15px;
        /* Slightly smaller base font for small screens */
    }
```

```
.container {
    width: 95%;
    /* Wider container for small screens */
}

header h1 {
    font-size: 1.9em;
    /* Adjusted for mobile */
}

header nav a {
    font-size: 1em;
    margin: 8px 0;
    /* Stack nav links vertically */
    display: block;
    /* Make links block for easier tapping */
}

header nav {
    flex-direction: column;
    /* Ensure vertical stacking if not already */
}

.post {
    padding: 20px;
}

.post h2 {
    font-size: 1.4em;
}

.sidebar {
    padding: 20px;
}

.sidebar h3 {
    font-size: 1.3em;
```

```
    }

    footer {
        padding: 20px 0;
        font-size: 0.85em;
    }
}
```

Center a Div

Perfect Centering

In the following example we will solve a common style problem: perfect centering.

SOLUTION: Set both the `justify-content` and `align-items` properties to `center`, and the flex item will be perfectly centered:

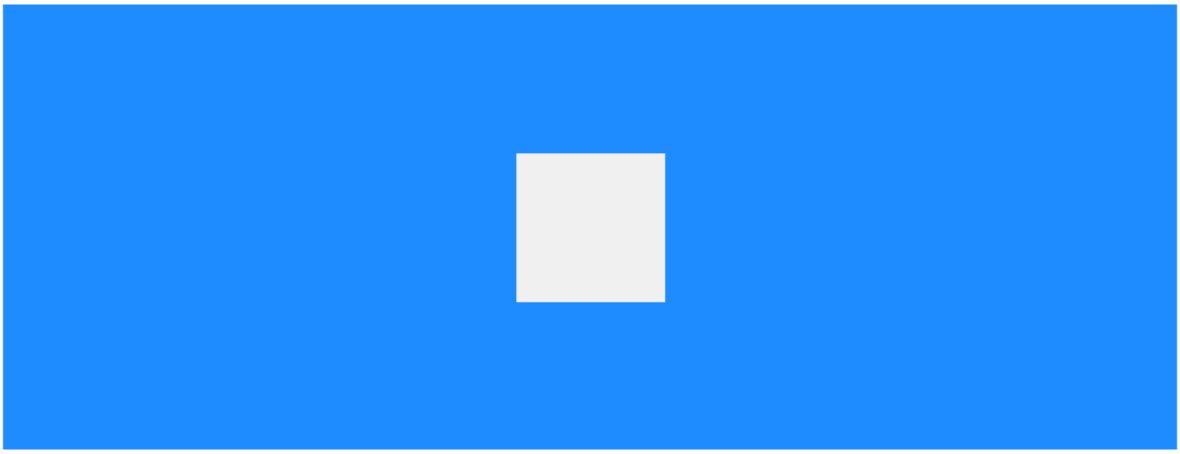

image_34.png

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 300px;
  background-color: DodgerBlue;
}

.flex-container > div {
  background-color: #f1f1f1;
  color: white;
  width: 100px;
```

```
  height: 100px;
}
</style>
</head>
<body>

<h1>Perfect Centering</h1>

<p>A flex container with both the justify-content and the align-items properties set to <em>center</em> will align the item(s) in the center (in both axis).</p>

<div class="flex-container">
  <div></div>
</div>

</body>
</html>
```

CSS Flexbox

The Flexbox Layout (Flexible Box) module (a W3C Candidate Recommendation (https://www.w3.org/TR/css-flexbox/) as of October 2017) aims at providing a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic (thus the word “flex”).

- Flexbox is short for the Flexible Box Layout module.
- Flexbox is a layout method for arranging items in rows or columns.
- Flexbox makes it easier to design a flexible responsive layout structure, without using float or positioning.

CSS Flexbox Components

A flexbox always consists of:

1. a **Flex Container** - the parent (container) element
2. **Flex Items** - the items inside the container

Flexbox Properties:

display: flex;

image_18.png

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
```

```
  display: flex;
  background-color: DodgerBlue;
}

.flex-container > div {
  background-color: #f1f1f1;
  margin: 10px;
  padding: 20px;
  font-size: 30px;
}
</style>
</head>
<body>

<h1>Create a Flex Container</h1>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

<p>A Flexible Layout must have a parent element with the
<em>display</em> property set to <em>flex</em>.</p>

<p>Direct child elements(s) of the flexible container automatically
becomes flexible items.</p>

</body>
</html>
```

- The `.flex-container` becomes a flex container with three direct child `div` elements (flex items).
- These `div` elements will be displayed in a row (default behavior of Flexbox).

flex-direction

The `flex-direction` property specifies the display-direction of flex items in the flex container.

The flex-direction property can have one of the following values:

1. `row`
2. `column`
3. `row-reverse`
4. `column-reverse`

- The **`row`** value is the default value, and it displays the `flex items horizontally` (from left to right):

```
.flex-container {
  display: flex;
  flex-direction: row;
}
```

image_19.png

- The **`column`** value displays the `flex items vertically` (from top to bottom):

```
.flex-container {
  display: flex;
  flex-direction: column;
}
```

image_20.png

- The **row-reverse** value displays the flex items horizontally (but from right to left):

```
.flex-container {
  display: flex;
  flex-direction: row-reverse;
}
```

image_21.png

- The **column-reverse** value displays the flex items vertically (but from bottom to top):

justify-content

The justify-content property is used to align the flex items along the **main axis** (default is horizontal). The justify-content property can have one of the following values:

1. flex-start: Aligns items to the start of the container (default).
2. flex-end: Aligns items to the end of the container.

3. `center`: Aligns items to the center.
4. `space-between`: Distributes items with equal space between them.
5. `space-around`: Distributes items with equal space around them.
6. `space-evenly`: Distributes items with equal space between and around them.

- The `space-between` value displays the flex items with space between them:

image_22.png

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flexbox - justify-content</title>
  <style>
    .flex-container {
      display: flex;
      justify-content: space-between; /* Distribute space between items
*/
      background-color: DodgerBlue;
    }

    .flex-container > div {
      background-color: #f1f1f1;
      margin: 10px;
      padding: 20px;
      font-size: 30px;
    }
  </style>
</head>
```

```
<body>

<h1>Flexbox - justify-content</h1>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

</body>
</html>
```

- The `space-evenly` value displays the flex items with equal space around them:

image_23.png

- The `space-around` value displays the flex items with space around them:

image_24.png

- The `flex-end` value positions the flex items at the end of the container:

image_25.png

- The `flex-start` value positions the flex items at the beginning of the container (this is default):

image_26.png

- The `center` value positions the flex items in the center of the container:

image_27.png

align-items

The `align-items` property is used to align the flex items along the **cross axis** (perpendicular to the main axis, default is vertical).

Example : Aligning Flex Items Vertically

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flexbox - align-items</title>
  <style>
    .flex-container {
      display: flex;
      align-items: center; /* Align items vertically */
      height: 200px;
      background-color: DodgerBlue;
    }

    .flex-container > div {
```

```
      background-color: #f1f1f1;
      margin: 10px;
      padding: 20px;
      font-size: 30px;
    }
  </style>
</head>
<body>

<h1>Flexbox - align-items</h1>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

</body>
</html>
```

align-items Values:

- `flex-start`: Aligns items to the start of the cross axis.

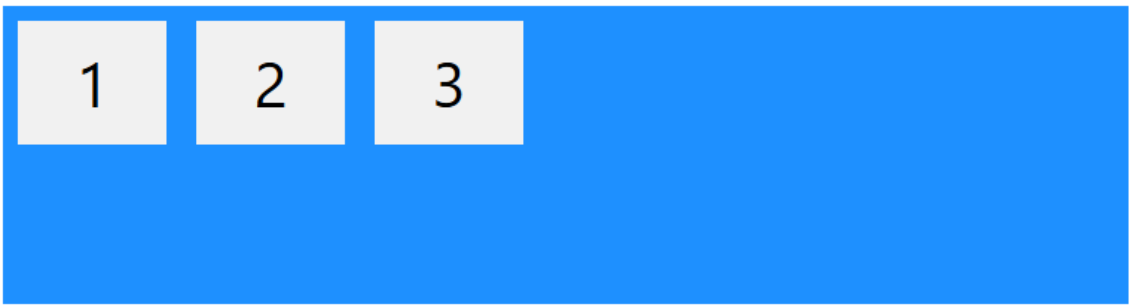

image_28.png

- `flex-end`: Aligns items to the end of the cross axis.

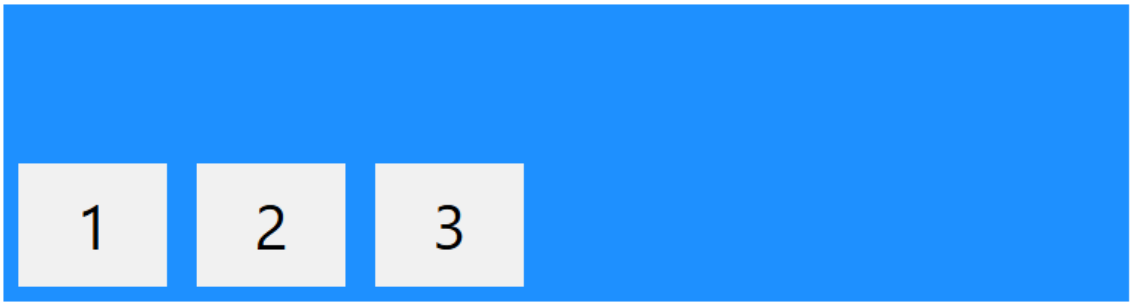

image_29.png

- `center`: Aligns items to the center of the cross axis.

image_32.png

- `baseline`: Aligns items to their baseline (useful for text).

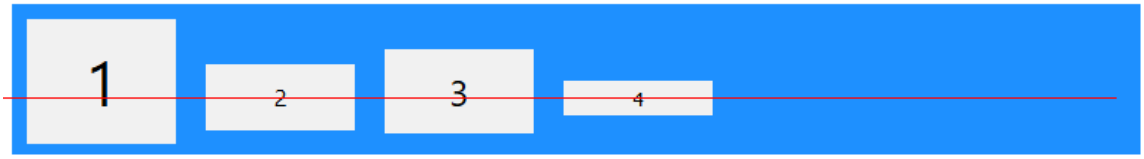

image_31.png

- `stretch`: Stretches items to fill the container (default).

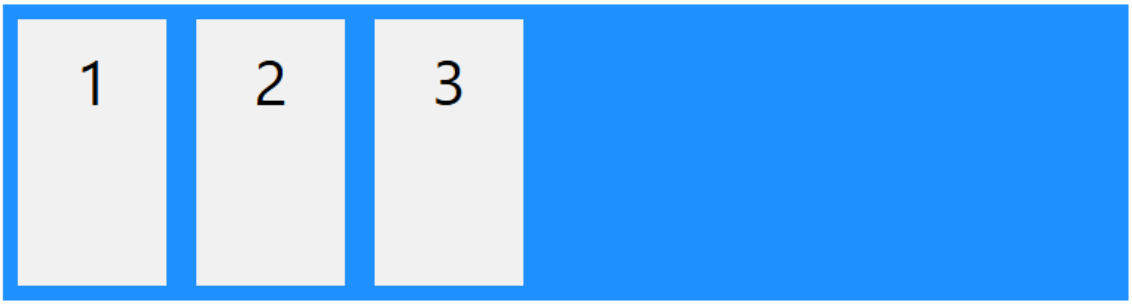

image_30.png

flex-wrap

By default, flex items will try to fit into one line. The `flex-wrap` property controls whether items should wrap to the next line if there's not enough space.

Example : Wrapping Flex Items

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flexbox - flex-wrap</title>
  <style>
    .flex-container {
      display: flex;
      flex-wrap: wrap; /* Allow wrapping */
      background-color: DodgerBlue;
    }

    .flex-container > div {
      background-color: #f1f1f1;
      margin: 10px;
      padding: 20px;
      font-size: 30px;
      flex: 1 1 100px; /* Allow the items to shrink or grow */
    }
  </style>
</head>
<body>

<h1>Flexbox - flex-wrap</h1>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
```

```
  <div>5</div>
</div>

</body>
</html>
```

flex-wrap Values:

- `nowrap` (default): Items will not wrap, they will overflow.
- `wrap`: Items will wrap onto the next line.
- `wrap-reverse`: Items will wrap onto the next line in reverse order.

flex-grow, flex-shrink, flex-basis

These properties control how the flex items grow, shrink, or have a fixed size.

Example : Flexible Items with Grow, Shrink, and Basis

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flexbox - flex-grow, flex-shrink, flex-basis</title>
  <style>
    .flex-container {
      display: flex;
      background-color: DodgerBlue;
    }

    .flex-container > div {
      background-color: #f1f1f1;
      margin: 10px;
      padding: 20px;
      font-size: 30px;
      flex-grow: 1; /* Allow items to grow */
    }
```

```
    .flex-container > div:first-child {
      flex-grow: 2; /* First item will grow twice as much */
    }
  </style>
</head>
<body>

<h1>Flexbox - flex-grow, flex-shrink, flex-basis</h1>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

</body>
</html>
```

flex Shorthand:

- `flex-grow`: How much the item should grow relative to others.
- `flex-shrink`: How much the item should shrink relative to others.
- `flex-basis`: The initial size of the item before growing or shrinking.
- The shorthand `flex: grow shrink basis` allows you to set all three at once.

7. align-self

This property allows you to override the alignment set by `align-items` for individual items.

image_35.png

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flexbox - align-self</title>
  <style>
    .flex-container {
      display: flex;
      height: 200px;
      background-color: DodgerBlue;
    }

    .flex-container > div {
      background-color: #f1f1f1;
      margin: 10px;
      padding: 20px;
      font-size: 30px;
      align-self: flex-start; /* Override alignment */
    }

    .flex-container > div:nth-child(2) {
      align-self: center; /* Center this item */
    }
  </style>
</head>
```

```
<body>

<h1>Flexbox - align-self</h1>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

</body>
</html>
```

Responsive Flexbox

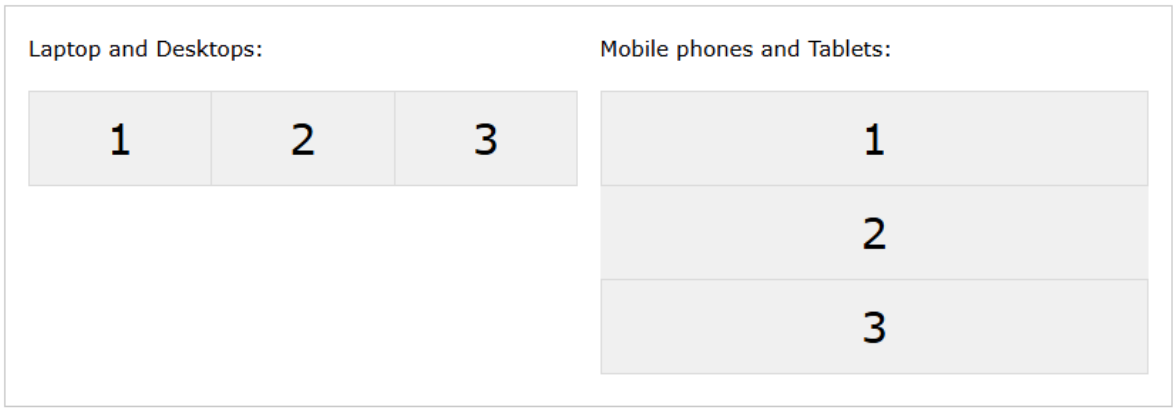

image_36.png

For example, if you want to create a two-column layout for most screen sizes, and a one-column layout for small screen sizes (such as phones and tablets), you can change the flex-direction from row to column at a specific breakpoint (800px in the example below):

```
<!DOCTYPE html>
<html>
<head>
<style>
* {
  box-sizing: border-box;
}

.flex-container {
  display: flex;
  flex-direction: row;
  font-size: 30px;
  text-align: center;
}

.flex-item-left {
  background-color: #f1f1f1;
  padding: 10px;
  flex: 50%;
}
```

```
.flex-item-right {
  background-color: dodgerblue;
  padding: 10px;
  flex: 50%;
}

/* Responsive layout - makes a one column-layout instead of two-column
layout */
@media (max-width: 800px) {
  .flex-container {
    flex-direction: column;
  }
}
</style>
</head>
<body>

<h1>Responsive Flexbox</h1>

<p>The "flex-direction: row;" stacks the flex items horizontally (from
left to right).</p>
<p>The "flex-direction: column;" stacks the flex items vertically (from
top to bottom).</p>
<p><b>Resize the browser window to see that the direction changes when
the
screen size is 800px wide or smaller.</b></p>

<div class="flex-container">
  <div class="flex-item-left">1</div>
  <div class="flex-item-right">2</div>
</div>

</body>
</html>
```

Another way is to change the percentage of the flex property of the flex items to create different layouts for different screen sizes. Note that we also have to include flex-wrap: wrap; on the flex container for this example to work:

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
}

.flex-item-left {
  flex: 50%;
}

.flex-item-right {
  flex: 50%;
}

/* Responsive layout - makes a one column layout instead of a two-column
layout */
@media (max-width: 800px) {
  .flex-item-right, .flex-item-left {
    flex: 100%;
  }
}
```

Responsive Image Gallery using Flexbox

Use flexbox to create a responsive image gallery that varies between four, two or full-width images, depending on screen size:

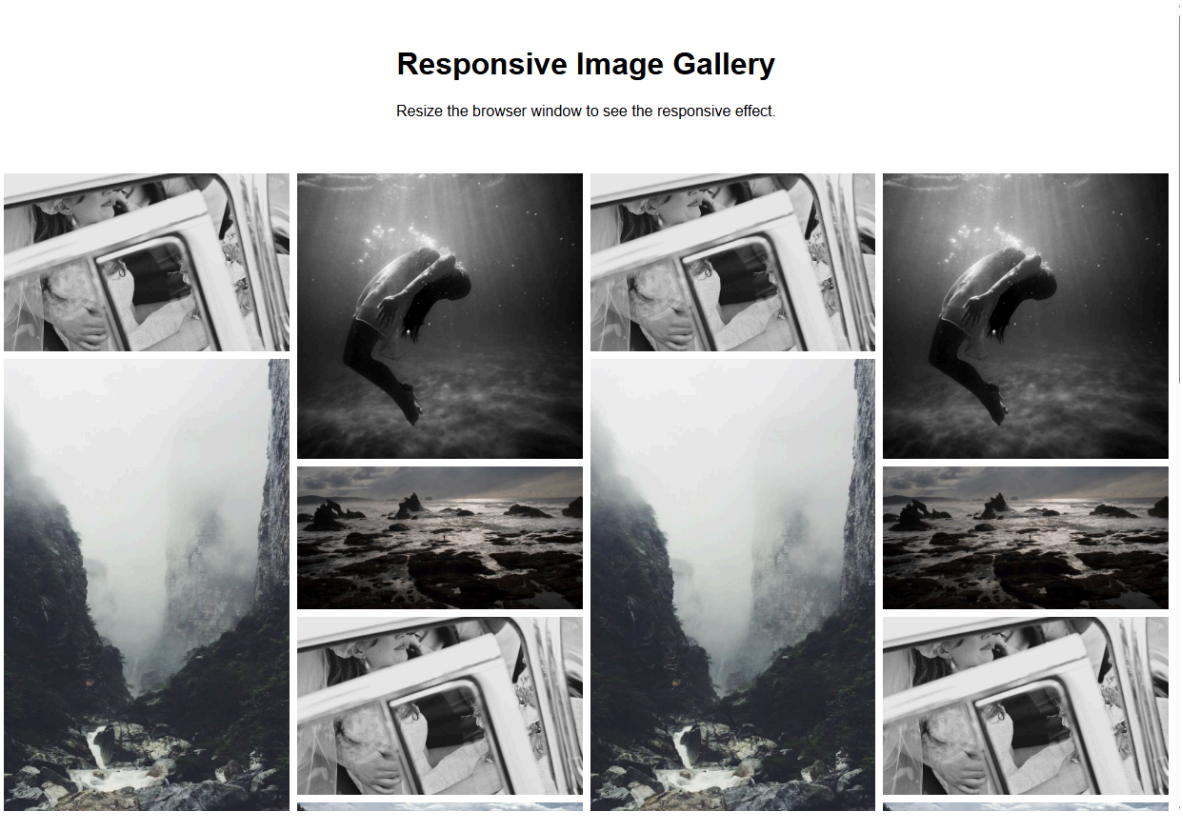

image_37.png

```
<!DOCTYPE html>
<html>
<head>
    <title>Responsive Image Gallery</title>
<style>
* {
    box-sizing: border-box;
}

body {
```

```
margin: 0;
font-family: Arial, Helvetica, sans-serif;
/* Added Helvetica as a fallback */
}

.header {
text-align: center;
padding: 32px;
background-color: #f2f2f2;
/* Added a light background color */
}

.header h1 {
/* Added styling for h1 in header */
font-size: 2.5em;
color: #333;
}

.header p {
/* Added styling for p in header */
font-size: 1.1em;
color: #555;
}

.row {
display: flex;
flex-wrap: wrap;
padding: 0 4px;
}

/* Create four equal columns that sits next to each other */
.column {
flex: 25%;
max-width: 25%;
padding: 0 4px;
}

.column img {
```

```
margin-top: 8px;
vertical-align: middle;
width: 100%;
/* Replaced inline style with CSS rule */
border-radius: 4px;
/* Added slight rounding to images */
box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
/* Added a subtle shadow to images */
transition: transform 0.2s ease-in-out;
/* Added hover effect */
}

.column img:hover {
transform: scale(1.03);
/* Slight zoom on hover */
}

/* Responsive layout - makes a two column-layout instead of four columns
*/
@media (max-width: 800px) {
.column {
flex: 50%;
max-width: 50%;
}
}

/* Responsive layout - makes the two columns stack on top of each other
instead of next to each other */
@media (max-width: 600px) {
.column {
flex: 100%;
max-width: 100%;
}
}
</style>

</head>
<body>
```

```
<!-- Header -->
    <div class="header">
        <h1>Responsive Image Gallery</h1>
        <p>Resize the browser window to see the responsive effect.</p>
    </div>

<!-- Photo Grid -->
    <div class="row">
        <div class="column">
            <img src="https://www.w3schools.com/w3images/wedding.jpg">
            <img src="https://www.w3schools.com/w3images/rocks.jpg">
            <img src="https://www.w3schools.com/w3images/falls2.jpg">
            <img src="https://www.w3schools.com/w3images/paris.jpg">
            <img src="https://www.w3schools.com/w3images/nature.jpg">
            <img src="https://www.w3schools.com/w3images/mist.jpg">
            <img src="https://www.w3schools.com/w3images/paris.jpg">
        </div>

        <div class="column">
            <img
src="https://www.w3schools.com/w3images/underwater.jpg">
            <img src="https://www.w3schools.com/w3images/ocean.jpg">
            <img src="https://www.w3schools.com/w3images/wedding.jpg">
            <img
src="https://www.w3schools.com/w3images/mountainskies.jpg">
            <img src="https://www.w3schools.com/w3images/rocks.jpg">
            <img
src="https://www.w3schools.com/w3images/underwater.jpg">
        </div>

        <div class="column">
            <img src="https://www.w3schools.com/w3images/wedding.jpg">
            <img src="https://www.w3schools.com/w3images/rocks.jpg">
            <img src="https://www.w3schools.com/w3images/falls2.jpg">
            <img src="https://www.w3schools.com/w3images/paris.jpg">
            <img src="https://www.w3schools.com/w3images/nature.jpg">
            <img src="https://www.w3schools.com/w3images/mist.jpg">
            <img src="https://www.w3schools.com/w3images/paris.jpg">
```

```
        </div>

        <div class="column">
            <img
src="https://www.w3schools.com/w3images/underwater.jpg">
            <img src="https://www.w3schools.com/w3images/ocean.jpg">
            <img src="https://www.w3schools.com/w3images/wedding.jpg">
            <img
src="https://www.w3schools.com/w3images/mountainskies.jpg">
            <img src="https://www.w3schools.com/w3images/rocks.jpg">
            <img
src="https://www.w3schools.com/w3images/underwater.jpg">
        </div>
    </div>

</body>
</html>
```

FlexBox Project

Try this challenge

This news homepage will be an excellent opportunity to practice your CSS Grid or flex skills.

Download this resource below from ⬇️

Design a News homepage (https://www.frontendmentor.io/challenges/news-homepage-H6SWTa1MFl)

- Desktop design

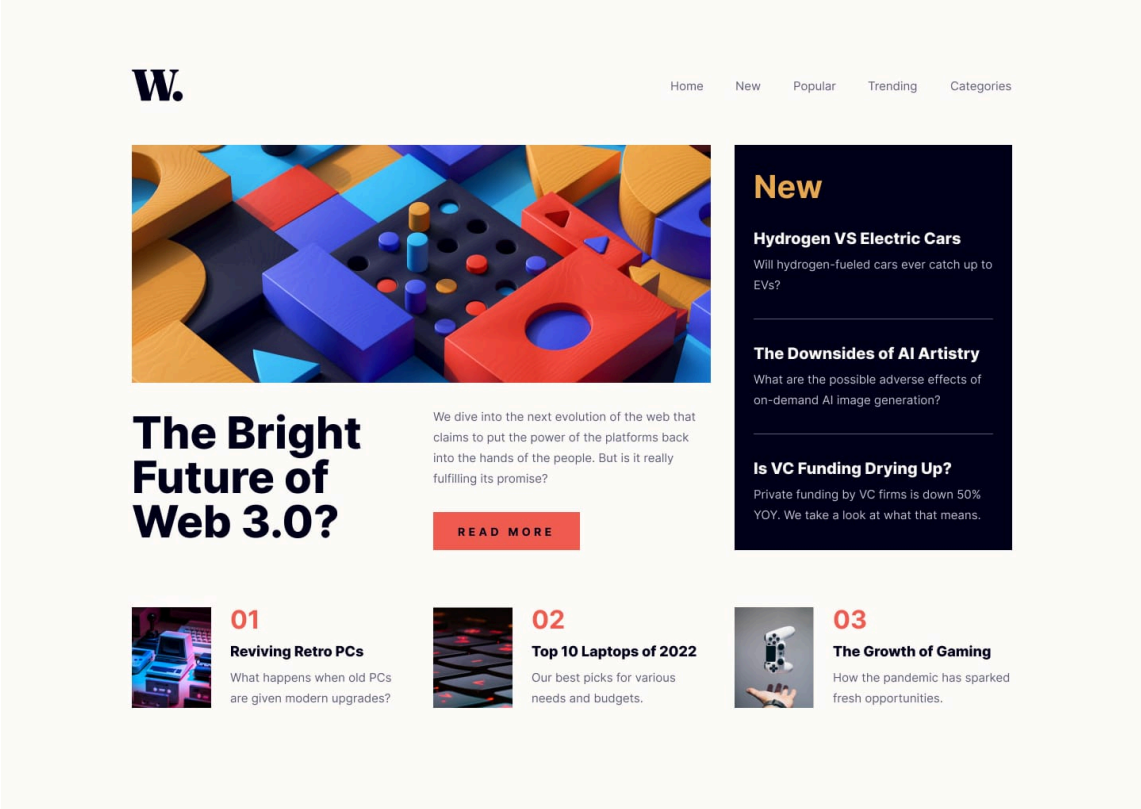

image_38.png

- Mobile view


W.
The Bright Future of Web 3.0?
We dive into the next evolution of the web that claims to put the power of the platforms back into the hands of the people. But is it really fulfilling its promise?
READ MORE
New
Hydrogen VS Electric Cars
Will hydrogen-fueled cars ever catch up to EVs?
The Downsides of AI Artistry
What are the possible adverse effects of on-demand AI image generation?
Is VC Funding Drying Up?
Private funding by VC firms is down 50% YOY. We take a look at what that means.
01
Reviving Retro PCs
What happens when old PCs are given modern upgrades?
02
Top 10 Laptops of 2022
Our best picks for various needs and budgets.
03
The Growth of Gaming
How the pandemic has sparked fresh opportunities.

image_39.png

Flexbox cheat-sheet

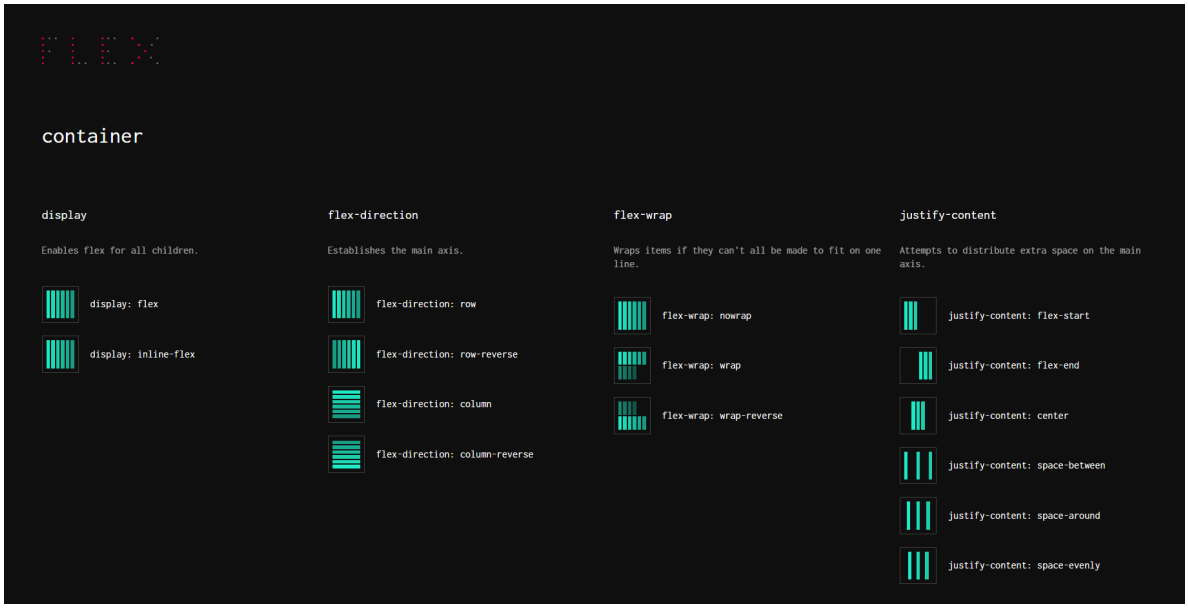

image_46.png

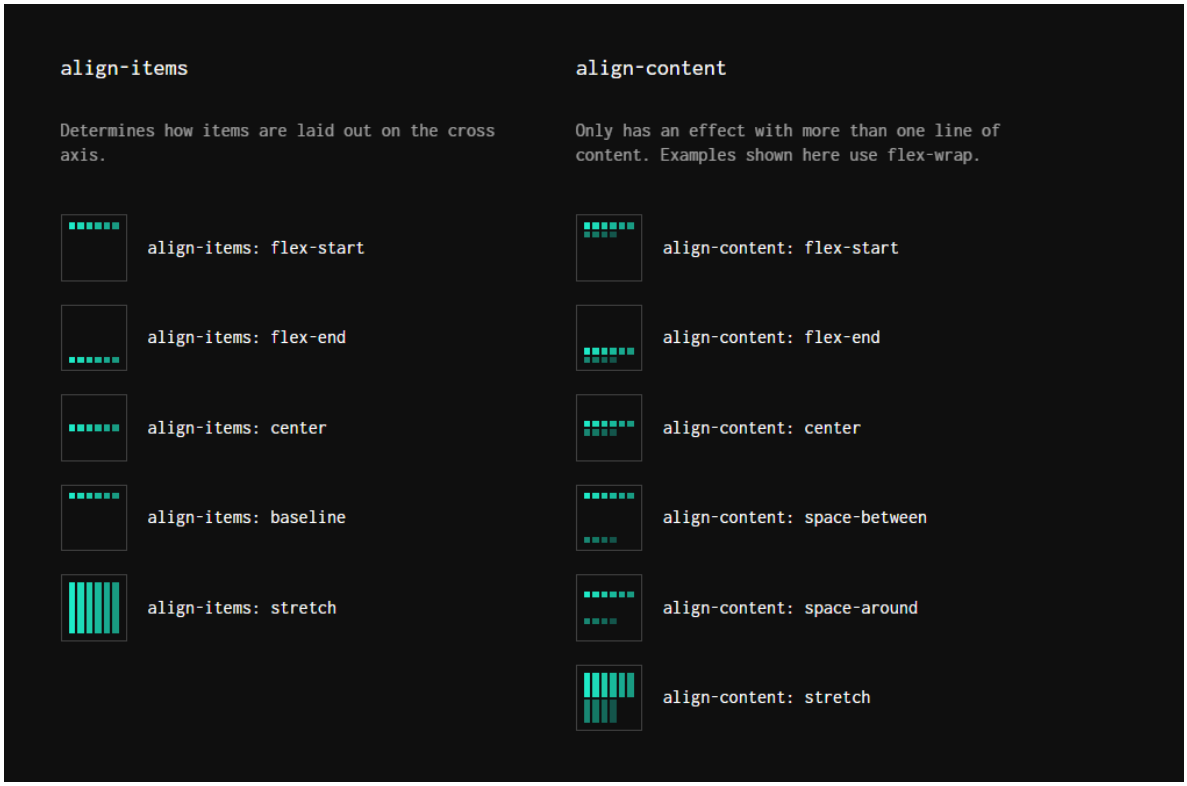

image_47.png

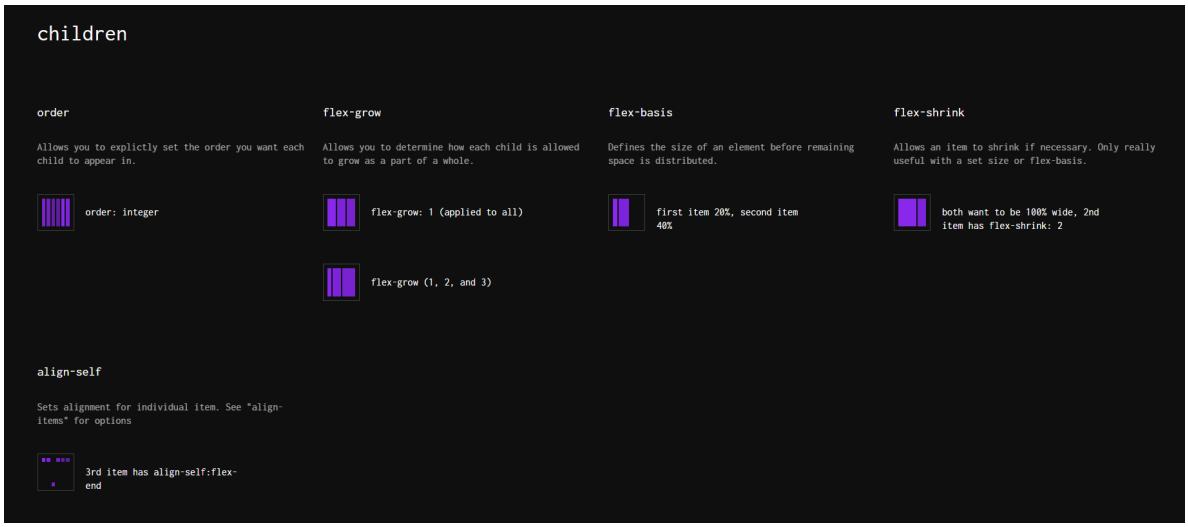
children
order
Allows you to explictly set the order you want each child to appear in.
order: integer
flex-grow
Allows you to determine how each child is allowed to grow as a part of a whole.
flex-grow: 1 (applied to all)
flex-grow (1, 2, and 3)
flex-basis
Defines the size of an element before remaining space is distributed.
first item 20%, second item 40%
flex-shrink
Allows an item to shrink if necessary. Only really useful with a set size or flex-basis.
both want to be 100% wide, 2nd item has flex-shrink: 2
align-self
Sets alignment for individual item. See "align-items" for options
3rd item has align-self:flex-end

image_48.png

CSS Grid

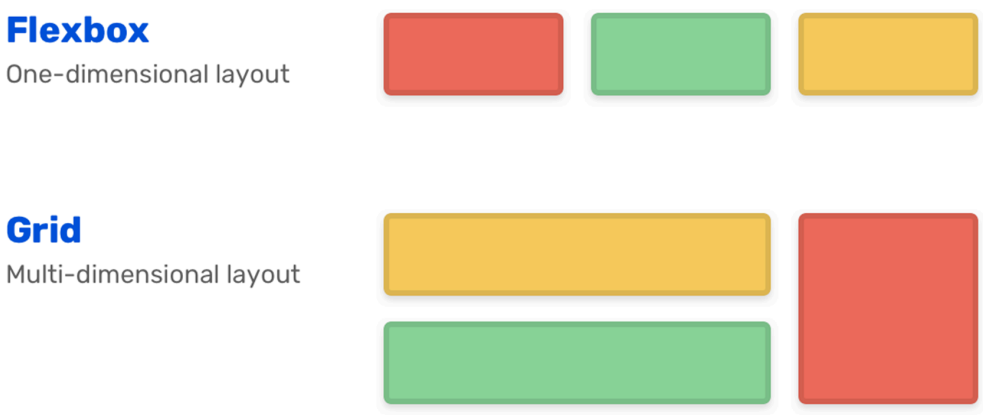

image_44.png

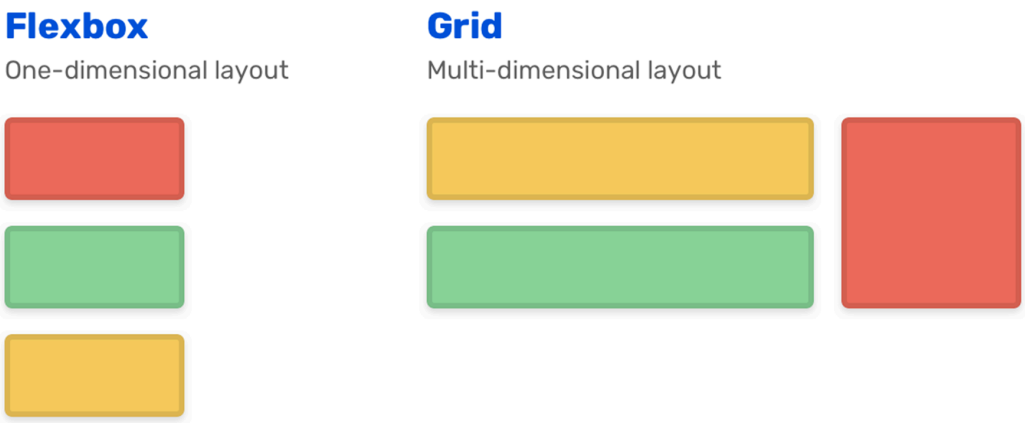

image_45.png

CSS Grid Layout

The Grid Layout Module offers a grid-based layout system, with rows and columns.

The Grid Layout Module allows developers to easily create complex web layouts.

Grid vs. Flexbox

The CSS Grid Layout should be used for two-dimensional layout, with rows AND

columns.

CSS Grid Components

A grid always consists of:

- a `Grid Container` - the parent (container) `<div>` element
- `Grid Items` - the items inside the container `<div>`

Grid Container and Grid Items

A grid layout consists of a parent element (the grid container), with one or more grid items.

All direct children of the grid container automatically become grid items. example :

```
<div class="container">
      <div>1</div>
      <div>2</div>
      <div>3</div>
      <div>4</div>
      <div>5</div>
      <div>6</div>
      <div>7</div>
      <div>8</div>
  </div>
```

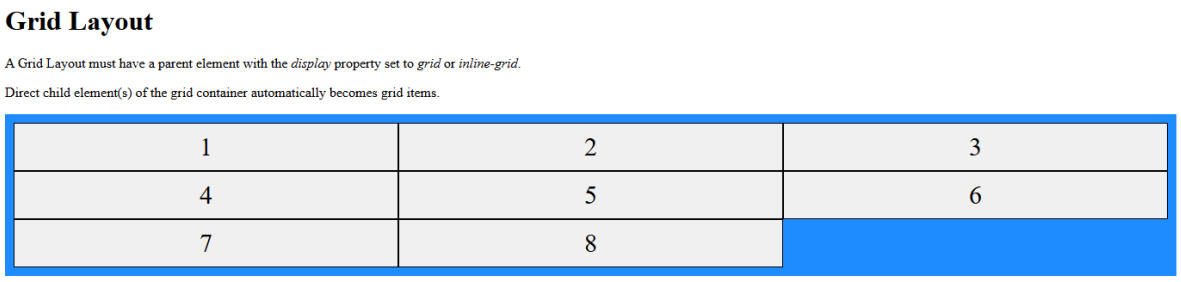

image_53.png

```
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  display: grid;
  grid-template-columns: auto auto auto;
  background-color: dodgerblue;
  padding: 10px;
}
.container > div {
  background-color: #f1f1f1;
  border: 1px solid black;
  padding: 10px;
  font-size: 30px;
  text-align: center;
}
</style>
</head>
<body>

<h1>Grid Layout</h1>

<p>A Grid Layout must have a parent element with the <em>display</em>
property set to <em>grid</em> or <em>inline-grid</em>.</p>

<p>Direct child element(s) of the grid container automatically becomes
grid items.</p>

<div class="container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
```

```
  <div>8</div>
</div>

</body>
</html>
```

Grid Item

A grid container contains one or more grid items.

By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.

The grid-column-start and grid-column-end Properties

The `grid-column-start` property specifies where to start a grid item.

```
.item1 {
  grid-column-start: 1;
  grid-column-end: 3;
}
```

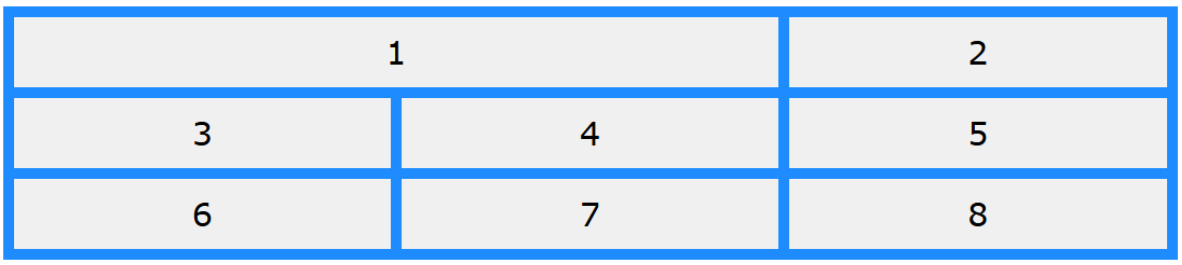

image_70.png

The grid-column Property

The `grid-column` property is a shorthand property for the `grid-column-start` and the `grid-column-end` properties.

To place an item, you can refer to line numbers, or use the keyword "span" to define how many columns the item will span.

- Place the first grid item at column-line 1, and let it span 2 columns:

```
.item1 {
  grid-column: 1 / span 2;
```

```
}
```

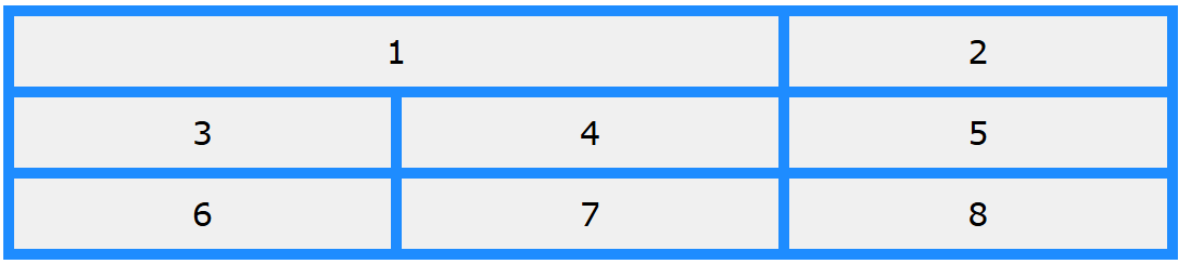

image_71.png

- Make "item1" start on column 1 and end before column 4

```
.item1 {
    grid-column: 1 / 4;
}
```

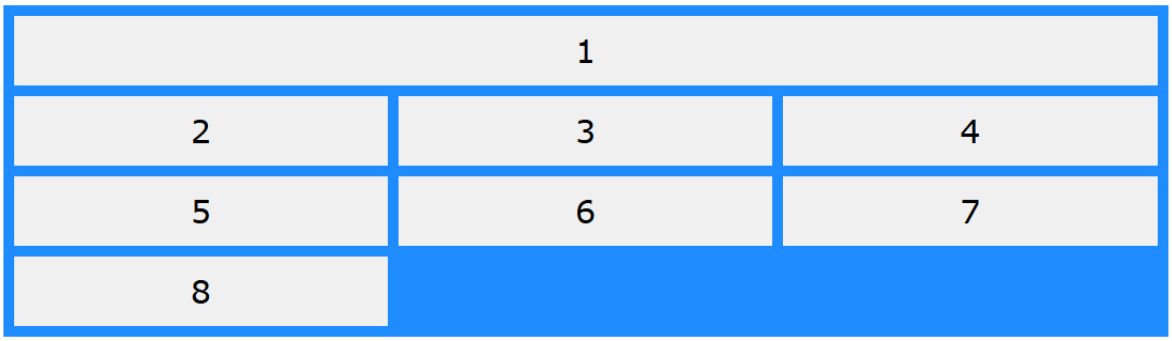

image_72.png

- Make "item2" start on column 2 and span 2 columns:

```
.item2 {
    grid-column: 2 / span 2;
}
```

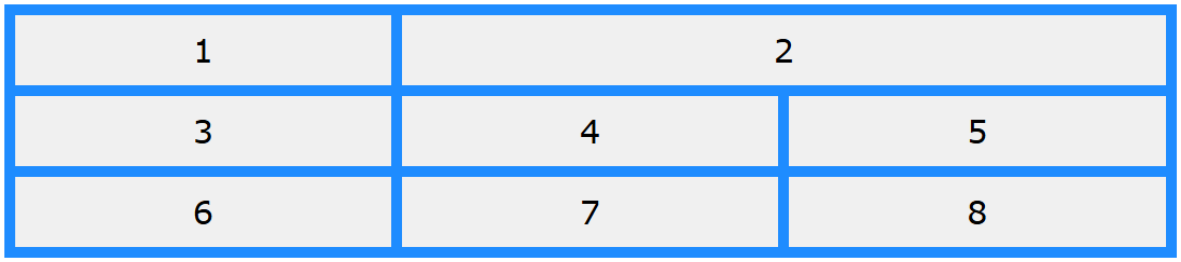

image_73.png

The grid-row Property

The `grid-row` property is a shorthand property for the `grid-row-start` and the `grid-row-end` properties.

- Place the first `grid` item at row-line 1, and let it span 2 rows:

```
.item1 {
  grid-row: 1 / span 2;
}
```

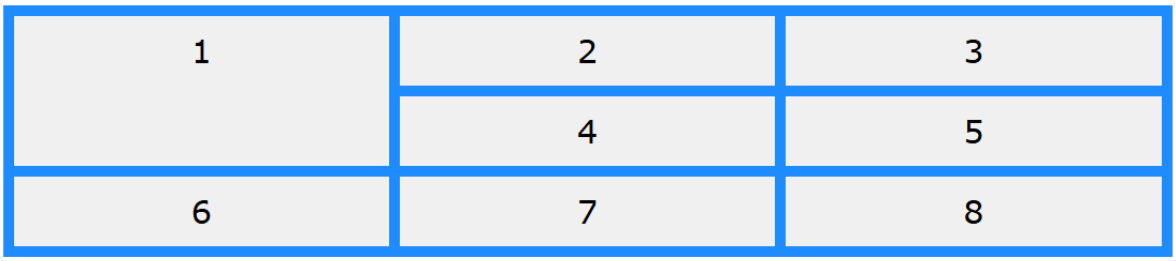

image_74.png

- Make "item1" start on row-line 1 and end before row-line 4:

```
.item1 {
  grid-row: 1 / 4;
}
```

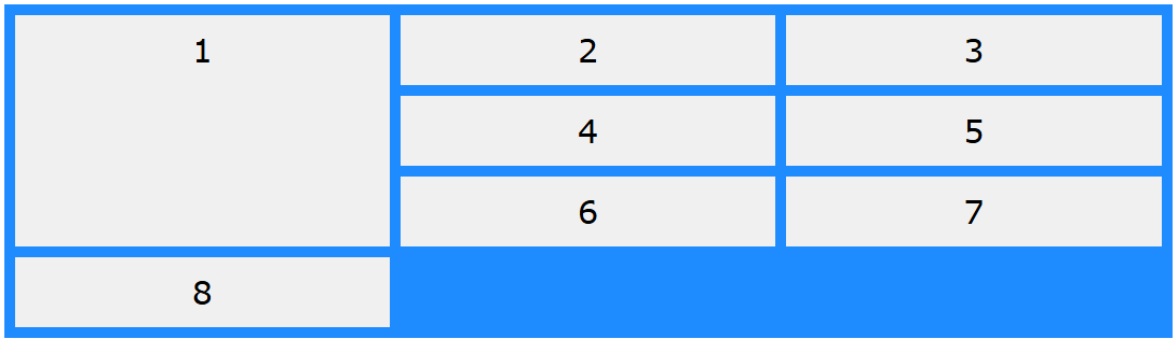

image_75.png

The grid-area Property

The `grid-area` property is a shorthand property for the `grid-row-start`, `grid-column-start`, `grid-row-end` and the `grid-column-end` properties.

The syntax is `grid-row-start/grid-column-start/grid-row-end/grid-column-end`.

- Make "item4" start on row-line 1 and column-line 2, and end on row-line 3 and column line 2:

```
.item4 {
  grid-area: 1 / 2 / 3 / 2;
}
```

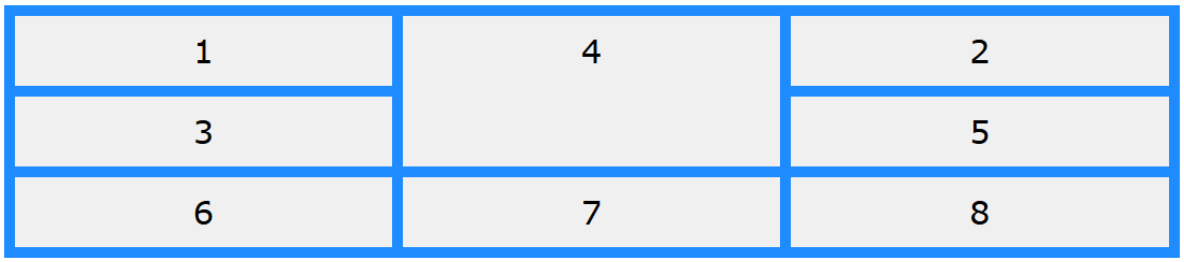

image_76.png

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
```

```
  display: grid;
  grid-template-columns: auto auto auto;
  gap: 10px;
  background-color: dodgerblue;
  padding: 10px;
}

.grid-container > div {
  background-color: #f1f1f1;
  text-align: center;
  padding: 10px;
  font-size: 30px;
}

.item4 {
  grid-area: 1 / 2 / 3 / 2;
}
</style>
</head>
<body>

<h1>The grid-area Property</h1>

<p>You can use the <em>grid-area</em> property to specify where to place an item.</p>

<p>Make "item4" start on row 1 and column 2, and end on row 3 and column 2:</p>

<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
  <div class="item7">7</div>
  <div class="item8">8</div>
```

```
</div>

</body>
</html>
```

Naming Grid Items with grid-area

The `grid-area` property can also be used to assign names to grid items.

The named grid items can then be referred to by the `grid-template-areas` property of the grid container.

```
.item1 { grid-area: header; }
.item2 { grid-area: menu; }
.item3 { grid-area: main; }
.item4 { grid-area: right; }
.item5 { grid-area: footer; }

.grid-container {
  grid-template-areas:
    'header header header header header header'
    'menu main main main main right'
    'menu footer footer footer footer footer';
}
```

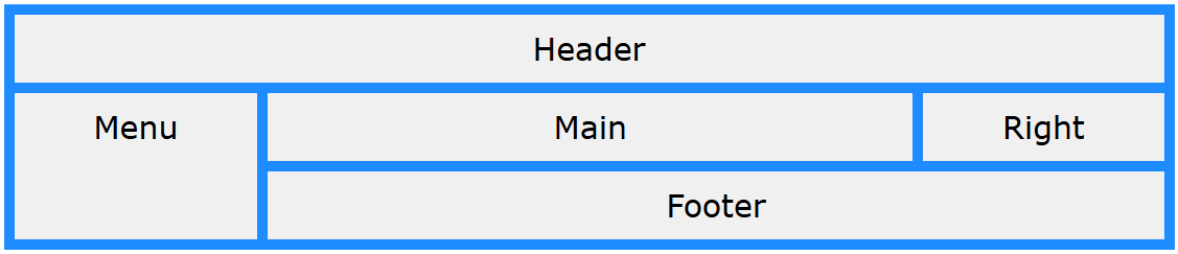

image_77.png

```
<!DOCTYPE html>
<html>
<head>
<style>
```

```
.grid-container {
  display: grid;
  grid-template-areas:
    'header header header header header header'
    'menu main main main main right'
    'menu footer footer footer footer footer';
  gap: 10px;
  background-color: dodgerblue;
  padding: 10px;
}

.grid-container > div {
  background-color: #f1f1f1;
  color:#000;
  padding: 10px;
  font-size: 30px;
  text-align: center;
}

.item1 { grid-area: header; }
.item2 { grid-area: menu; }
.item3 { grid-area: main; }
.item4 { grid-area: right; }
.item5 { grid-area: footer; }
</style>
</head>
<body>

<h1>The grid-area Property</h1>

<p>You can use the <em>grid-area</em> property to name grid items.</p>

<p>This grid layout contains six columns and three rows:</p>

<div class="grid-container">
  <div class="item1">Header</div>
  <div class="item2">Menu</div>
  <div class="item3">Main</div>
```

```
  <div class="item4">Right</div>
  <div class="item5">Footer</div>
</div>

</body>
</html>
```

Grid Columns/Rows

Grid Columns

The vertical lines of grid items are called columns.

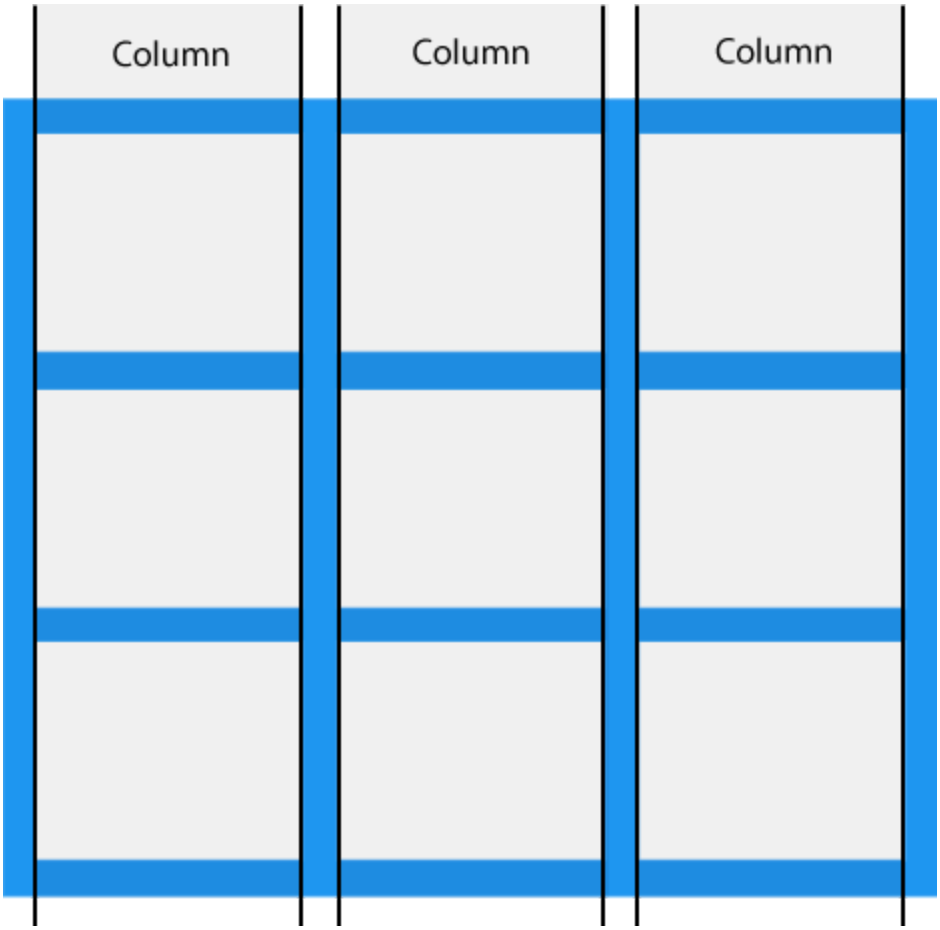

image_54.png

Grid Rows

The horizontal lines of grid items are called rows. ![image_55.png](image_55.png

Grid Gaps

The spaces between each column/row are called gaps. ![image_56.png](image_56.png

You can adjust the gap size by using one of the following properties:

- `column-gap`
- `row-gap`
- `gap`
- The `column-gap` property specifies the gap between the columns in a grid.

```
.container {
  display: grid;
  column-gap: 50px;
}
```

- The `row-gap` property specifies the gap between the rows in a grid.

```
.container {
  display: grid;
  row-gap: 50px;
}
```

- The `gap` property is a shorthand property for row-gap and column-gap:

```
.container {
    display: grid;
    gap: 50px 100px;
}
```

The grid-column-start and grid-column-end Properties

The `grid-column-start` property specifies where to start a grid item.

The `grid-column-end` property specifies where to end a grid item.

```
.item1 {
  grid-column-start: 1;
```

```
  grid-column-end: 3;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  gap: 10px;
  background-color: dodgerblue;
  padding: 10px;
}

.grid-container > div {
  background-color: #f1f1f1;
  text-align: center;
  padding: 10px;
  font-size: 30px;
}

.item1 {
  grid-column-start: 1;
  grid-column-end: 3;
}
</style>
</head>
<body>

<h1>The grid-column-start and grid-column-end Properties</h1>

<p>The <em>grid-column-start</em> property specifies where to start a grid item. The <em>grid-column-end</em> property specifies where to end a grid item.</p>

<p>Place the first grid item at column-line 1, and let it end on column-
```

```
line 3:</p>

<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
  <div class="item7">7</div>
  <div class="item8">8</div>
</div>
</body>
</html>
```

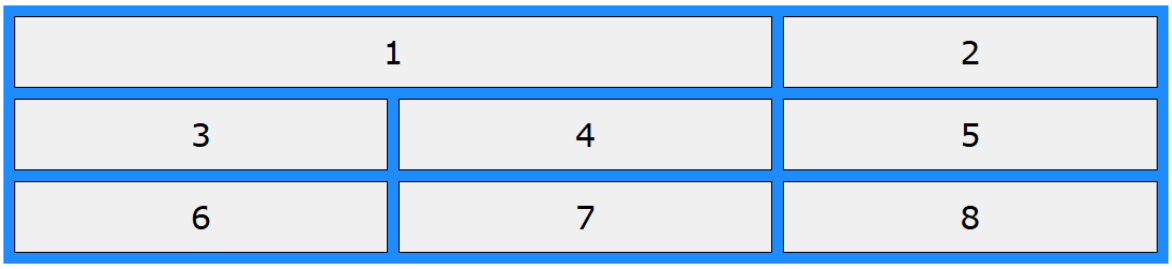

image_57.png

The grid-column Property

The `grid-column` property is a shorthand property for the `grid-column-start` and the `grid-column-end` properties.

```
.item1 {
  grid-column: 1 / span 2;
}
```

The grid-row-start and grid-row-end Property

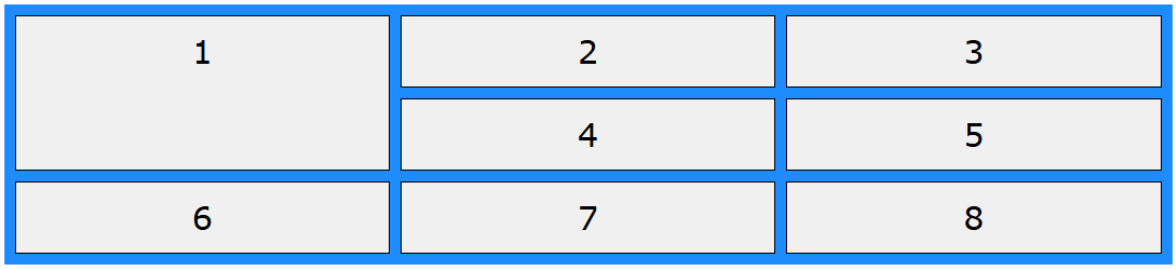

image_58.png

```
.item1 {
  grid-row-start: 1;
  grid-row-end: 3;
}
```

The grid-row Property

The `grid-row` property is a shorthand property for the `grid-row-start` and the `grid-row-end` properties.

Grid Container

A grid container contains one or more grid items arranged in columns and rows. An element becomes a `grid` container when its `display` property is set to `grid` or `inline-grid`.

Grid Tracks

The `rows` and `columns` of a `grid` is defined with the `grid-template-rows` and the `grid-template-columns` properties (or the shorthand `grid` or `grid-template` properties).

The grid-template-columns Property

The `grid-template-columns` property defines the number of columns in your grid layout, and it can define the width of each column.

The value is a space-separated-list, where each value defines the width of the respective column.

- Make a grid with 4 columns of equal width:

image_59.png

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto;
  gap: 10px;
  background-color: dodgerblue;
  padding: 10px;
}
```

```
.grid-container > div {
  background-color: #f1f1f1;
  color: #000;
  padding: 10px;
  font-size: 30px;
  text-align: center;
}
</style>
</head>
<body>

<h1>The grid-template-columns Property</h1>

<p>You can use the <em>grid-template-columns</em> property to specify
the number of columns in your grid layout.</p>

<div class="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
</div>

</body>
</html>
```

- Set a fixed size for column 1, 2, and 4, and keep column 3 as auto size:

```
.grid-container {
    display: grid;
    grid-template-columns: 80px 200px auto 40px;
}
```

1	2	3	4
5	6	7	8

image_60.png

Cell Sizing with the fr Unit

The `fr unit` stands for "fraction". This unit automatically divides the available space into `fractions`.

Example: `1fr` will take 1 fraction of the available space, while 2fr will take 2 fractions of the available space.

- Here, each column will take up 25% of the container width, splitting it equally:

1	2	3	4
5	6	7	8

image_61.png

```
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
}
```

- Here, the second column will be twice as big as the others:

1	2	3	4
5	6	7	8

image_62.png

```
.grid-container {
    display: grid;
    grid-template-columns: 1fr 2fr 1fr 1fr;
}
```

The grid-template-rows Property

The grid-template-rows property defines the height of each row.

The value is a space-separated-list, where each value defines the height of the respective row

```
.grid-container {
  display: grid;
  grid-template-rows: 80px 200px;
}
```

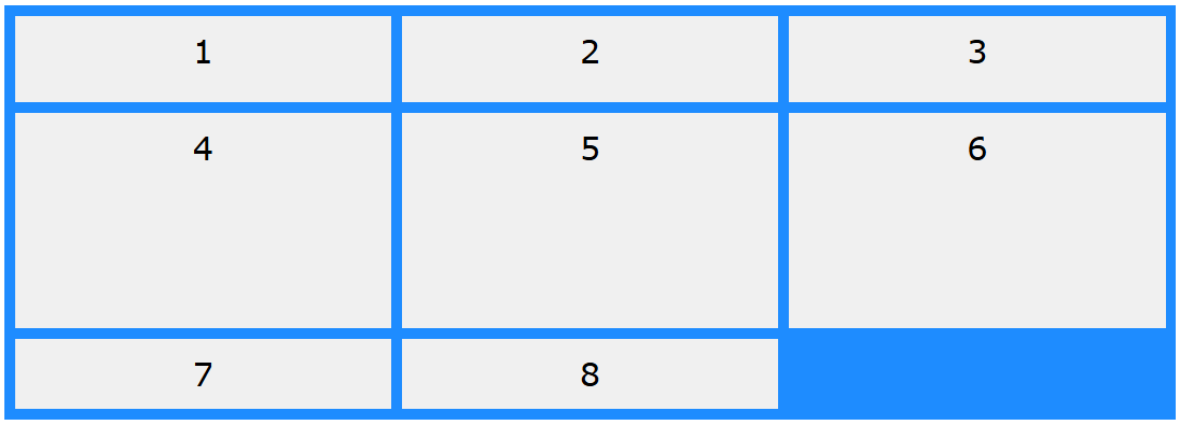

image_63.png

The justify-content Property

The `justify-content` property is used to align the grid items when they do not use all available space on the main-axis (horizontally).

The `justify-content` property can have one of the following values:

1. `space-evenly`

2. `space-around`
3. `space-between`
4. `center`
5. `start`
6. `end`

- The `space-evenly` value displays the grid items with equal space around them:

```
.grid-container {
    display: grid;
    justify-content: space-evenly;
}
```

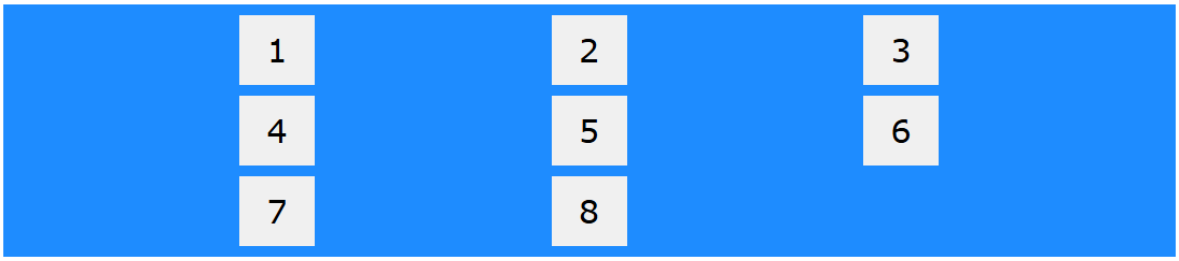

image_64.png

- The `space-around` value displays the grid items with space around them:

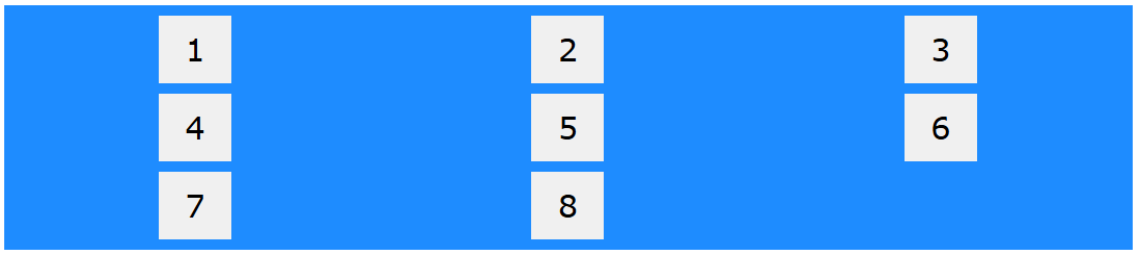

image_65.png

- The `space-between` value displays the grid items with space between them:

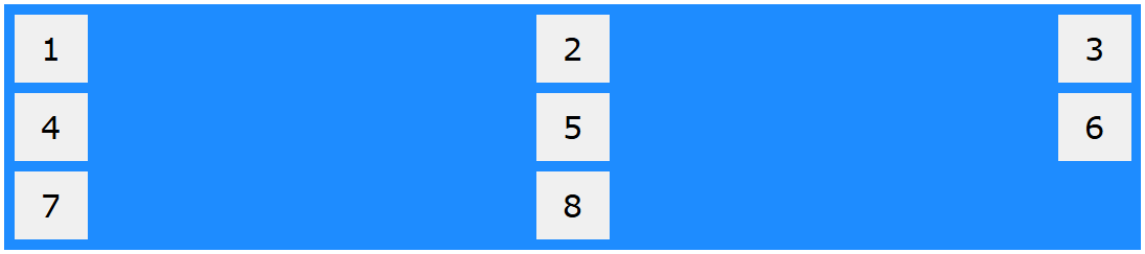

image_66.png

- The `center` value positions the grid items in the center of the container:

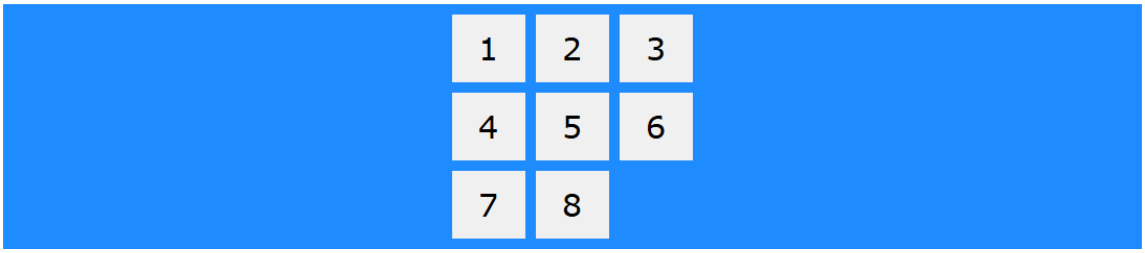

image_67.png

- The start value positions the grid items at the start of the container:

image_68.png

- The end value positions the grid items at the end of the container:

image_69.png

The align-content Property

The `align-content` property is used to align the grid items when they do not use all available space on the cross-axis (vertically).

The `align-content` property can have one of the following values:

1. `space-evenly`
2. `space-around`
3. `space-between`
4. `center`
5. `start`
6. `end`

Grid cheat-sheet

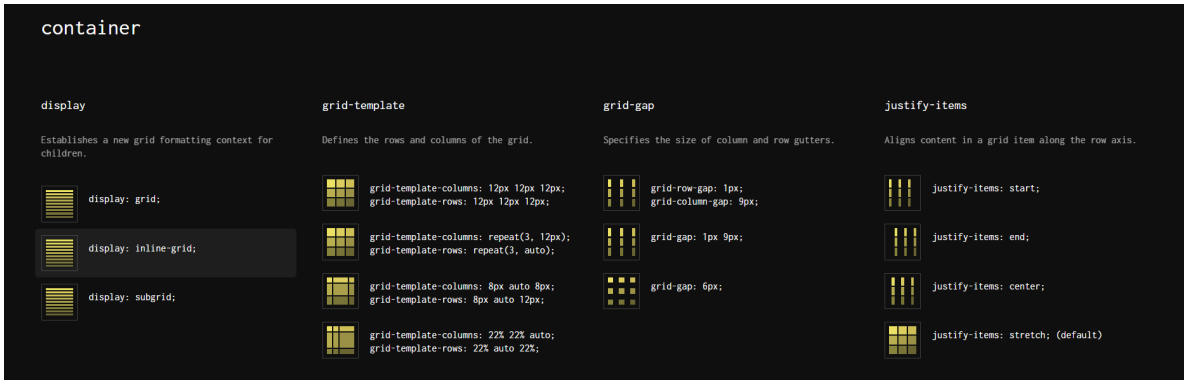

image_49.png

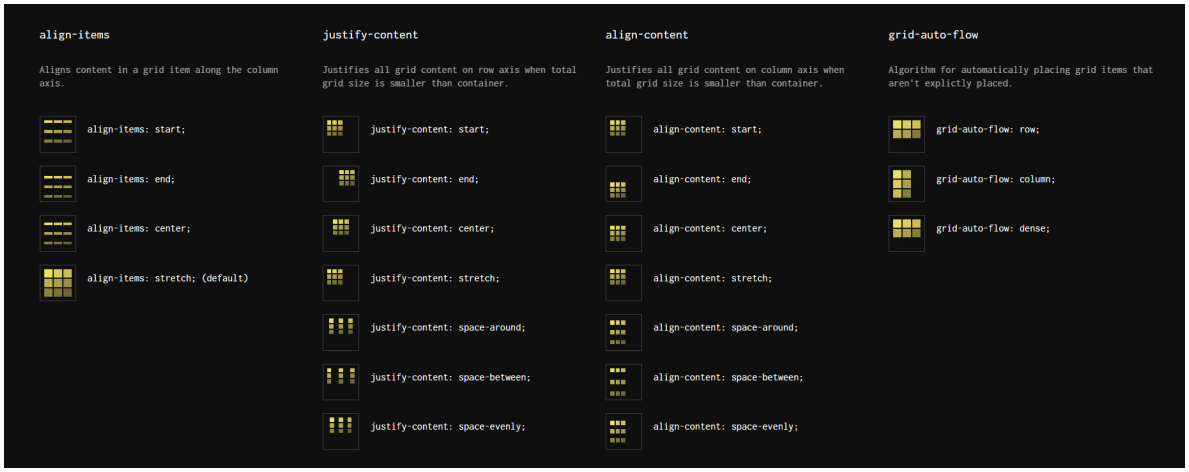

image_50.png

image_51.png

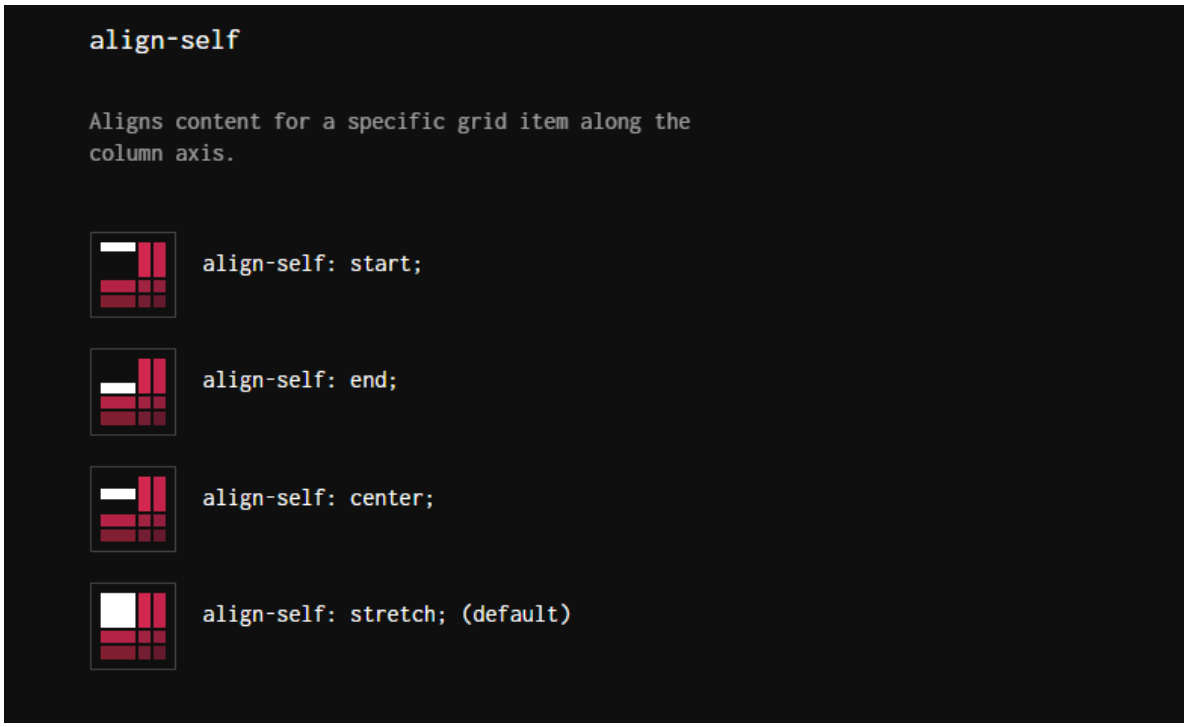
align-self
Aligns content for a specific grid item along the column axis.
align-self: start;
align-self: end;
align-self: center;
align-self: stretch; (default)

image_52.png

Grid Project

Try this challenge

Time tracking dashboard,

A perfect opportunity to practice your CSS Grid skills. For anyone wanting to take it up a notch, we provide a JSON data file to practice working with data.

Download this resource below from ⬇️

Design a Time tracking dashboard (https://www.frontendmentor.io/challenges/time-tracking-dashboard-UIQ7167Jw)

- Desktop design

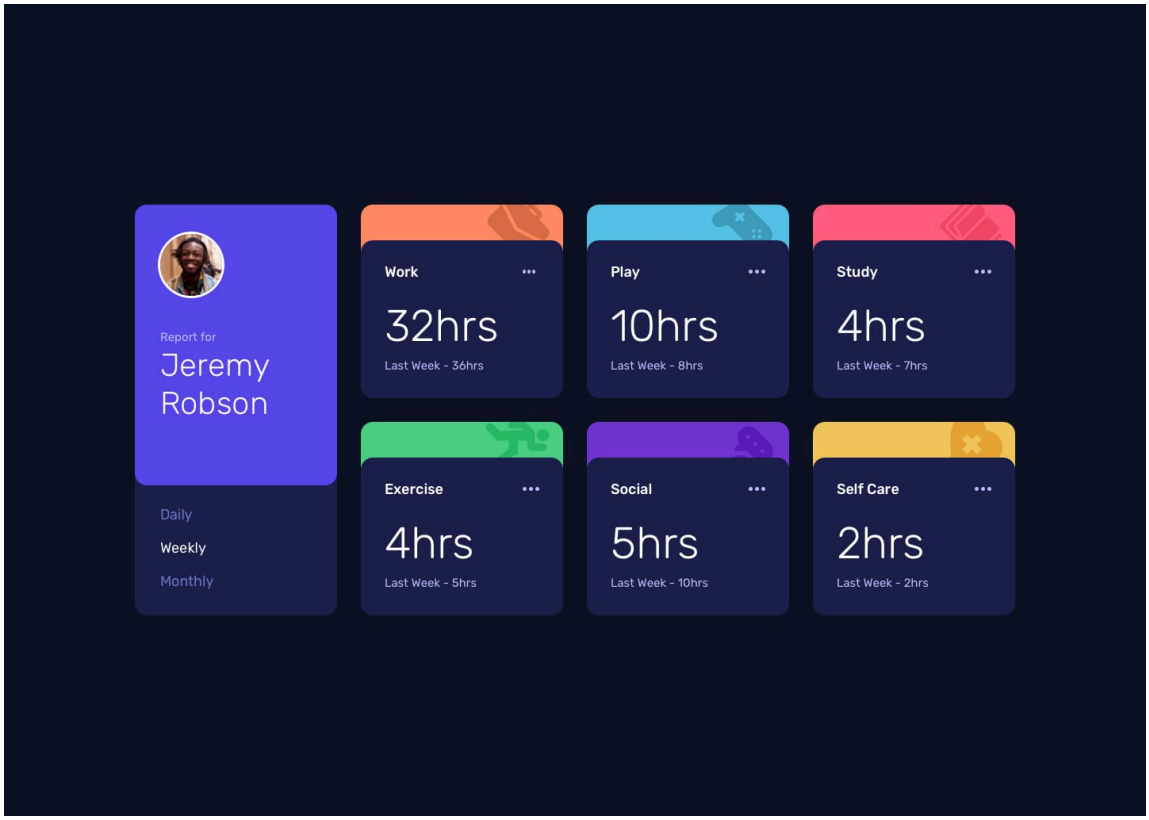

image_40.png

- Mobile view


Report for
Jeremy Robson
Daily
Weekly
Monthly
Work
32hrs
Last Week – 36hrs
Play
10hrs
Last Week – 8hrs
Study
4hrs
Last Week – 7hrs
Exercise
4hrs
Last Week – 5hrs
Social
5hrs
Last Week – 10hrs
Self Care
2hrs
Last Week – 2hrs

image_41.png

Bento grid

This challenge is perfect for testing your CSS Grid and responsive skills with this bento grid layout.

Download this resource below from ⬇️

Design Bento grid (https://www.frontendmentor.io/challenges/bento-grid-RMydElrlOj)

- Desktop design

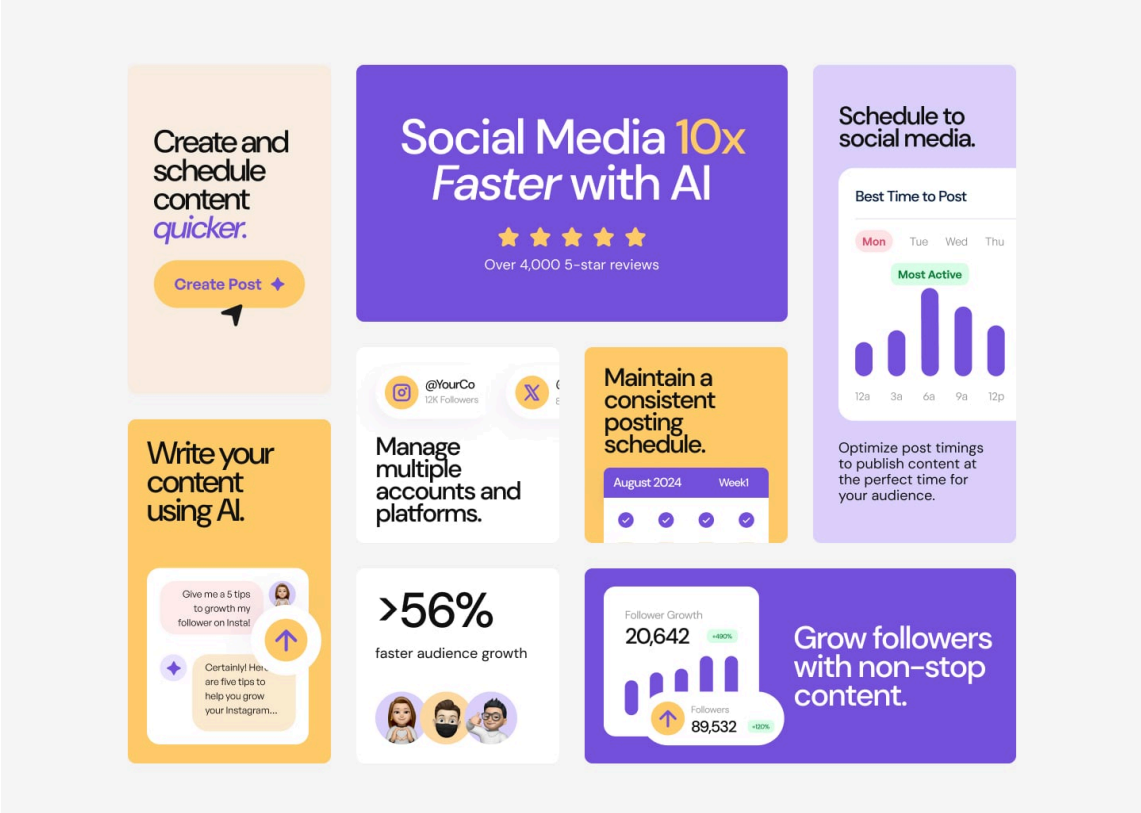

image_42.png

- Mobile view

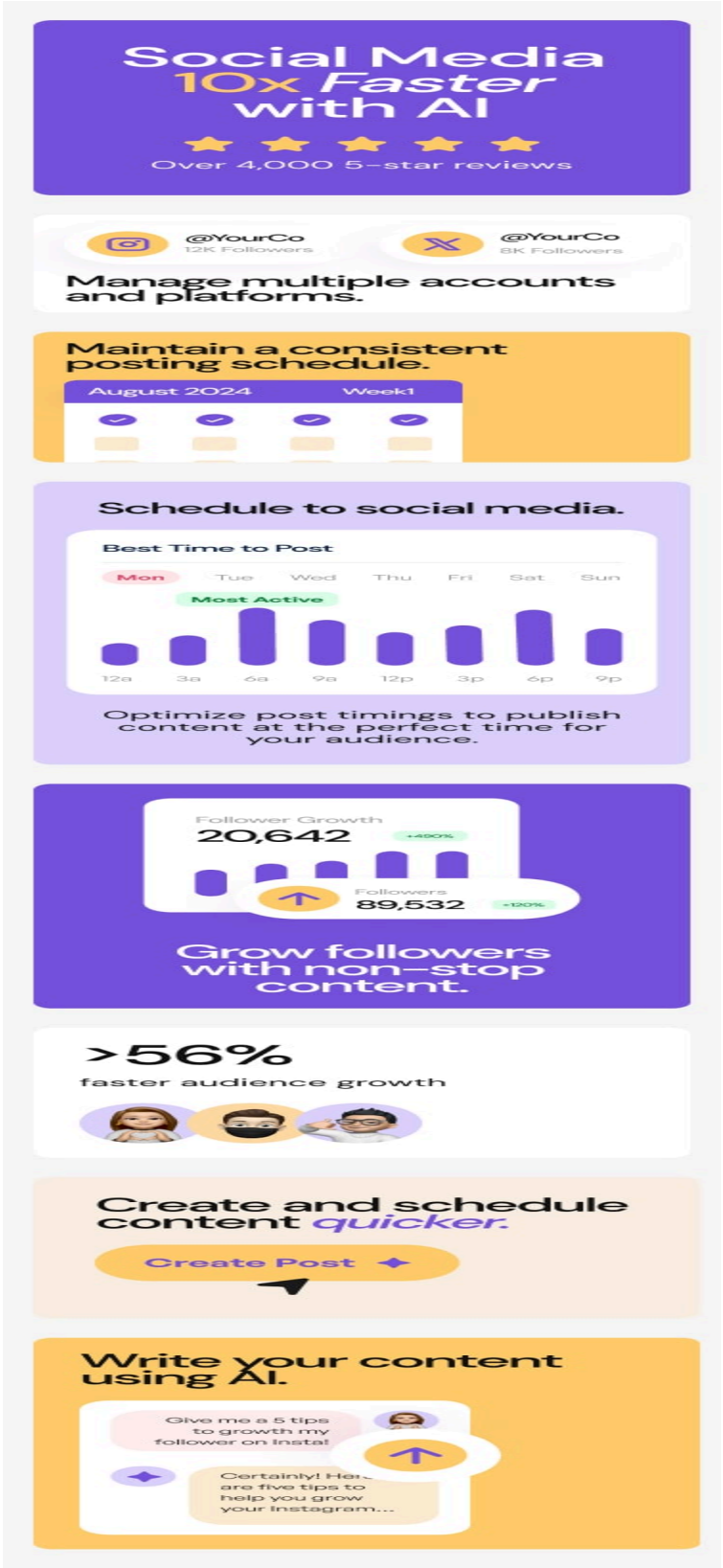
Social Media
10x Faster
with AI
Over 4,000 5-star reviews
@YourCo
12K Followers
@YourCo
8K Followers
Manage multiple accounts and platforms.
Maintain a consistent posting schedule.
August 2024
Week1
Schedule to social media.
Best Time to Post
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Most Active
12a
3a
6a
9a
12p
3p
6p
9p
Optimize post timings to publish content at the perfect time for your audience.
Follower Growth
20,642
+480%
Followers
89,532
+120%
Grow followers with non-stop content.
>56%
faster audience growth
Create and schedule content quicker.
Create Post
Write your content using AI.
Give me a 5 tips to growth my follower on Insta!
Certainly! Here are five tips to help you grow your Instagram...

image_43.png