Table of Contents

[1 Introduction to HTML 3](#_Toc164195657)

[1.1 What is HTML? 3](#_Toc164195658)

[1.2 HTML Heading Elements 3](#_Toc164195659)

[1.3 HTML Paragraph Elements 3](#_Toc164195660)

[1.4 Void Elements 3](#_Toc164195661)

[2 Intermediate HTML 4](#_Toc164195662)

[2.1 The List Elements 4](#_Toc164195663)

[2.2 Nesting and Indentation 5](#_Toc164195664)

[2.3 Anchor Elements 5](#_Toc164195665)

[2.4 Image Elements 5](#_Toc164195666)

[3 Multi Page Website 5](#_Toc164195667)

[3.1 Computer File Paths 5](#_Toc164195668)

[3.2 What are Webpages? 6](#_Toc164195669)

[3.3 The HTML Boilerplate 6](#_Toc164195670)

[4 Introduction to CSS 7](#_Toc164195671)

[4.1 What is CSS 7](#_Toc164195672)

[4.2 How to use CSS 7](#_Toc164195673)

[4.3 CSS Selectors 8](#_Toc164195674)

[5 CSS Properties 9](#_Toc164195675)

[5.1 CSS Colours 9](#_Toc164195676)

[5.2 Font Properties 9](#_Toc164195677)

[5.3 Inspecting CSS 9](#_Toc164195678)

[5.4 The CSS Box Model 9](#_Toc164195679)

[5.4.1 Content Area 9](#_Toc164195680)

[5.4.2 Padding 9](#_Toc164195681)

[5.4.3 Border 10](#_Toc164195682)

[5.4.4 Margin 10](#_Toc164195683)

[5.4.5 Example 11](#_Toc164195684)

[6 Intermediate CSS 11](#_Toc164195685)

[6.1 The Cascade – Specificity and Inheritance 11](#_Toc164195686)

[6.2 Combining CSS Selectors 13](#_Toc164195687)

[6.3 CSS Positioning 15](#_Toc164195688)

[7 Advanced CSS 16](#_Toc164195689)

[7.1 CSS Display 16](#_Toc164195690)

[7.2 CSS Float 18](#_Toc164195691)

[7.3 How to Create Responsive Websties 19](#_Toc164195692)

[7.4 Media Queries 20](#_Toc164195693)

[8 Flexbox 22](#_Toc164195694)

[8.1 Display: Flex 22](#_Toc164195695)

[8.2 Flex Direction 22](#_Toc164195696)

[8.3 Flex Layout 22](#_Toc164195697)

[8.4 Flex Sizing 22](#_Toc164195698)

[9 Advanced CSS 22](#_Toc164195699)

[9.1 Display: Grid 22](#_Toc164195700)

[9.2 Grid Sizing 22](#_Toc164195701)

[9.3 Grid Placement 22](#_Toc164195702)

[10 Bootstrap 22](#_Toc164195703)

[10.1 What is Bootstrap 22](#_Toc164195704)

[10.2 Bootstrap In-depth 23](#_Toc164195705)

[11 Web Design 23](#_Toc164195706)

[12 Introduction to Javascript 23](#_Toc164195707)

[13 Intermediate Javascript 23](#_Toc164195708)

[14 The Document Object Model 24](#_Toc164195709)

[15 Advanced JS and DOM 24](#_Toc164195710)

[16 jQuery 24](#_Toc164195711)

[17 The Unix Command Line 24](#_Toc164195712)

[18 Backend Web Development 24](#_Toc164195713)

[19 Node.js 24](#_Toc164195714)

[20 Express.js with Node.js 25](#_Toc164195715)

[21 EJS 25](#_Toc164195716)

[22 Git, Github, Version Control 25](#_Toc164195717)

[23 APIS 25](#_Toc164195718)

[24 Databases 25](#_Toc164195719)

# 1 Introduction to HTML

## 1.1 What is HTML?

HTML defines the content and structure of the website. HTML stands for:

* HT: Hyper Text
  + The pieces of text that can link to other documents on the website. It defines the meaning and structure of web content. Other technologies besides HTML are generally used to describe a web page's appearance/presentation (CSS) or functionality/behaviour (JavaScript). "Hypertext" refers to links that connect web pages to one another, either within a single website or between websites [Wikipedia].
* ML: Markup Language
  + A markup language is a text-encoding system consisting of a set of symbols inserted in a text document to control its structure, formatting, or the relationship between its parts. Markup is often used to control the display of the document or to enrich its content to facilitate automated processing [Wikipedia].

## 1.2 HTML Heading Elements

HTML heading elements are used to define headings in a document, with <h1> being the highest-level heading and <h6> being the lowest. Each heading level represents a different section and sub-section hierarchy in the document structure.

<h1>This is a heading level 1</h1>

<h2>This is a heading level 2</h2>

<h3>This is a heading level 3</h3>

<h4>This is a heading level 4</h4>

<h5>This is a heading level 5</h5>

<h6>This is a heading level 6</h6>

## 1.3 HTML Paragraph Elements

The HTML paragraph element <p> is used to define a paragraph. Text inside a paragraph element is displayed with a space above and below it, which is the default margin applied by browsers.

<p>This is a paragraph of text. It will be separated from other text by margins.</p>

## 1.4 Void Elements

Void elements in HTML are elements that don't have a closing tag because they can't have any content within them. They are also known as self-closing elements. Examples include the <img> element for images, the <br> element for line breaks, and the <input> element for form inputs.

<img src="image.jpg" alt="A description of the image">

<br>

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

In the examples above:

* The <img> element has attributes src to specify the image source and alt to provide alternative text.
* The <br> element creates a line break.
* The <input> element is a textbox where users can enter their username, with a placeholder text displayed when the textbox is empty.

These elements provide structure and functionality to a webpage, allowing developers to organize content and enable user interaction in a meaningful way.

# 2 Intermediate HTML

## 2.1 The List Elements

HTML provides three types of lists: ordered lists, unordered lists, and description lists.

* Ordered Lists (<ol>): The items in an ordered list are marked with numbers.

<ol>

<li>First item</li>

<li>Second item</li>

<li>Third item</li>

</ol>

* Unordered Lists (<ul>): The items in an unordered list are marked with bullets.

<ul>

<li>Apple</li>

<li>Banana</li>

<li>Cherry</li>

</ul>

* Description Lists (<dl>): Description lists consist of terms (<dt>) and descriptions (<dd>).

<dl>

<dt>HTML</dt>

<dd>Hypertext Markup Language</dd>

<dt>CSS</dt>

<dd>Cascading Style Sheets</dd>

</dl>

## 2.2 Nesting and Indentation

Nesting refers to placing elements within other elements, following the parent-child relationship. Indentation helps in visually organizing the code to show the nesting structure.

<ul>

<li>Fruits

<ul>

<li>Apple</li>

<li>Banana</li>

</ul>

</li>

<li>Vegetables

<ul>

<li>Carrot</li>

<li>Spinach</li>

</ul>

</li>

</ul>

## 2.3 Anchor Elements

Anchor elements (<a>) are used to define hyperlinks that link to other webpages or resources.

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

* href attribute specifies the URL.
* target attribute with value \_blank opens the link in a new tab/window.

## 2.4 Image Elements

Image elements (<img>) are used to embed images into webpages.

<img src="image.jpg" alt="A beautiful scenery" width="500" height="600">

* src attribute specifies the image source.
* alt attribute provides alternative text for screen readers or if the image fails to load.
* width and height attributes specify the dimensions of the image.

These elements and practices are fundamental to HTML.

# 3 Multi Page Website

## 3.1 Computer File Paths

Multipage websites consist of multiple web pages linked together, usually through navigation menus. Each page has its unique URL and displays different content, but all pages share a common design template. This structure allows for organized, easy-to-navigate websites with various content sections.

Example:

* Home Page: example.com
* About Us Page: example.com/about
* Contact Page: example.com/contact

File paths indicate the location of files and directories on a computer. They can be absolute or relative.

Absolute Path: Specifies the full URL or file path.

* Example: C:\Users\John\Documents\file.txt or https://example.com/images/photo.jpg

Relative Path: Specifies the path relative to the current location.

* Example: If your HTML file is in a directory named website, and there's an image in a subdirectory named images, the relative path to the image would be images/photo.jpg.

## 3.2 What are Webpages?

Web pages are documents encoded in HTML that are accessible via a web browser. They can display text, images, videos, and other media and can contain links to other web pages. Each web page has a unique URL (Uniform Resource Locator).

## 3.3 The HTML Boilerplate

HTML boilerplate is a basic HTML template that provides a starting point for building a web page. It includes essential HTML elements and attributes that are common to most web pages.

A screen shot of a computer code

Description automatically generated

* <!DOCTYPE html>: Declares the document to be HTML5.
* <html>: The root element that contains all other HTML elements.
* <head>: Contains meta-information, links to stylesheets, and other resources.
* <body>: Contains the content of the document.
* <title>: Sets the title of the web page.
* <link>: Links to an external stylesheet.
* <script>: Links to an external JavaScript file or contains inline JavaScript code.

These fundamental concepts are crucial for understanding web development and building effective, well-structured websites.

# 4 Introduction to CSS

## 4.1 What is CSS

Foeuiwbhfwfupe

## 4.2 How to use CSS

CSS can be implemented in three ways: Inline, Internal, and External.

* Inline CSS is applied directly to HTML elements using the style attribute.

<p style="color: red; font-size: 16px;">This is a paragraph with inline CSS.</p>

* Internal CSS is placed within the <head> section of an HTML document using the <style> tag.

A screen shot of a computer code

Description automatically generated

* External CSS is written in a separate file with a .css extension and linked to the HTML document using the <link> element.

A screen shot of a computer

Description automatically generated

## 4.3 CSS Selectors

CSS selectors are patterns used to select the HTML elements you want to style. In CSS, selectors are used to target the HTML elements on a web page so that style properties can be applied to them. There are several types of selectors in CSS, each allowing you to select elements based on different criteria such as their name, id, class, attributes, and more. Here's an overview of the most common types of CSS selectors:

1. Type Selector (also known as Element Selector): Selects all elements of a specific type. For example, p selects all <p> elements.
2. Class Selector: Selects all elements with a specific class attribute. It's prefixed with a period (.). For example, .menu selects all elements with class="menu".
3. ID Selector: Selects a single element with a specific id attribute. It's prefixed with a hash (#). For example, #header selects the element with id="header".
4. Attribute Selector: Selects elements based on the presence or value of a given attribute. For example, [type="text"] selects all elements with a type attribute value of "text".
5. Descendant Selector: Selects an element that is a descendant of another element, including children, grandchildren, etc. It's indicated by a space. For example, div p selects all <p> elements that are inside a <div> element.
6. Child Selector (>): Selects an element that is a direct child of another element. For example, ul > li selects all <li> elements that are direct children of <ul> elements.
7. Adjacent Sibling Selector (+): Selects an element that is directly after another specific element on the same level of the hierarchy. For example, h1 + p selects the first <p> element directly following any <h1> element.
8. General Sibling Selector (~): Selects all elements that are siblings of a specified element. For example, h1 ~ p selects all <p> elements that are siblings of an <h1> element.
9. Pseudo-class Selector: Selects elements based on their state or relation to other elements. For example, :hover applies styles to an element when the mouse hovers over it, and :nth-child(n) selects the nth child of its parent.
10. Pseudo-element Selector: Selects part of an element, such as ::first-line or ::before. These allow you to style specific parts of an element rather than the whole element.

CSS selectors can be combined to target elements more specifically, enabling precise styling of web page components. By mastering CSS selectors, you can effectively control the presentation of HTML elements and create visually appealing web pages.

# 5 CSS Properties

## 5.1 CSS Colours

CSS properties are used to style and layout HTML elements. Each property has a name and a value, separated by a colon, and is placed inside a CSS declaration block.

## 5.2 Font Properties

1px

1pt

1em = 100% of parent

1rem = 100% of root (recommended)

## 5.3 Inspecting CSS

Lwreuingwprg

## 5.4 The CSS Box Model

The CSS Box Model is a fundamental concept in web design as it dictates how elements are displayed and interact with each other on the page. Each HTML element can be viewed as a box that may have padding, borders, and margins.

### 5.4.1 Content Area

The content area is the core of the box model, containing the actual content of the element such as text, images, or other media. The dimensions of the content area are specified by the width and height properties.

A screenshot of a computer program

Description automatically generated

### 5.4.2 Padding

Padding is the space between the content area and the border. It's like an inner margin around the content. Padding is specified using the padding property, which can have one to four values to set the padding for all four sides of the box.

A screenshot of a computer program

Description automatically generated

### 5.4.3 Border

The border surrounds the padding (if any) and content area. It's defined using the border property, which can set the border style, width, and color.

A black background with white text

Description automatically generated

### 5.4.4 Margin

The margin is the outermost layer that clears space between the element's border and its surrounding elements. It's like an outer margin around the border. Margins are specified using the margin property, similar to padding.

A screenshot of a computer program

Description automatically generated

### 5.4.5 Example

A screen shot of a computer program

Description automatically generated

In this example, the .box element has a content area of 200x100 pixels, a padding of 10 pixels on all sides, a solid black border of 2 pixels, and a margin of 20 pixels on all sides. This structure adheres to the CSS Box Model, demonstrating how the padding, border, and margin properties affect the overall layout and positioning of the element on the page.

Div element

# 6 Intermediate CSS

## 6.1 The Cascade – Specificity and Inheritance

Positions specificity Type Importance 4-1

The CSS Cascade, along with Specificity and Inheritance, are fundamental concepts in CSS (Cascading Style Sheets) that determine how styles are applied to HTML elements. These concepts work together to resolve conflicts when multiple CSS rules could apply to the same element. Understanding these concepts is crucial for effectively writing and debugging CSS.

The CSS Cascade

The "cascade" in CSS describes the process used to determine which styles are applied to an element. CSS stands for Cascading Style Sheets, emphasizing the "cascading" nature of applying styles. The cascade follows a set of rules to resolve conflicts by assigning a priority to each style rule. These rules are based on:

Source Order: If two rules have the same specificity, the later rule in the CSS file will overwrite earlier ones.

Importance: Styles marked with !important have the highest priority. However, using !important is generally discouraged unless absolutely necessary because it makes debugging difficult.

Specificity: A calculation that determines which styles apply to an element when multiple rules could apply. More specific selectors override more general ones.

Specificity

Specificity is a system to calculate which CSS rule applies if multiple rules have selectors matching the same element. It's essentially a score that is calculated based on the types of selectors in a given rule. The specificity calculation considers:

* Inline styles: Added directly to an element's style attribute, carrying the highest specificity.
* IDs: Selectors that target an element based on its id attribute.
* Classes, attributes, and pseudo-classes: Selectors based on class names, attributes, or states (:hover, :focus, etc.).
* Elements and pseudo-elements: Selectors that target elements by their type (e.g., div, p) or specific parts of elements (::before, ::after).

The specificity is calculated in a "0-0-0-0" format, counting the number of each selector type. The leftmost number has the highest priority. For example, a rule with one ID selector (0-1-0-0) is more specific than a rule with 100 class selectors (0-0-100-0).

Inheritance

Inheritance in CSS is a mechanism where some CSS properties (but not all) are passed from parent elements to their children. This means that if you set a property on a parent element, the child elements can inherit that property value. For example, if you set the font-family on a <body> element, all the text inside the body can inherit this font style, unless a more specific rule applies.

Not all properties are inherited by default, but inheritance can be explicitly controlled using the inherit value, forcing a property to be inherited, or initial, resetting to its default value.

Working Together

The CSS cascade, specificity, and inheritance work together to determine how styles are applied. The cascade resolves conflicts by considering the source of the style (author, user, or user agent styles), the specificity of the selector, and whether styles are marked as !important. Specificity helps to decide which rule applies when multiple rules target the same element. Inheritance allows properties to be passed down from parent elements to their children, creating a consistent look within a document without having to declare styles on every element.

Understanding and mastering these concepts is key to effectively writing CSS that is both powerful and easy to maintain.

The "0-0-0-0" format in CSS specificity is a way to quantify and compare the specificity of different CSS selectors. Specificity determines which CSS rule is applied to an element when multiple rules could apply. The format consists of four parts or levels, each representing a different type of selector. From left to right, these levels are:

1. Inline styles (highest priority): Styles applied directly to an element using the style attribute in HTML. This level is represented by the leftmost number in the specificity calculation. Each inline style adds 1 to this level, but since you can only have one style attribute per element, this part of the specificity score is either 0 or 1.
2. IDs: Selectors that target an element based on its id attribute. The second number in the specificity calculation represents the count of ID selectors in the selector being evaluated.
3. Classes, attributes, and pseudo-classes: The third number counts the total of three different types of selectors:

* Class selectors that target elements by their class attribute.
* Attribute selectors that apply styles based on the presence or value of an attribute.
* Pseudo-classes like :hover, :focus, etc., that style elements based on their state rather than their structure.

1. Elements and pseudo-elements (lowest priority): The rightmost number counts both:

* Type selectors (or element selectors) that target elements by their tag name (e.g., div, p).
* Pseudo-elements like ::before and ::after that style specified parts of an element.

Each level is considered more specific than the one to its right, with inline styles having the highest specificity. When calculating specificity, you tally the number of selectors of each type in the rule's selector, and then you can compare these specificity scores to determine which rule's styles are applied to an element.

For example, the specificity score of an inline style is "1-0-0-0", making it more specific than any other selector. A rule with an ID selector would have a specificity of "0-1-0-0", which is more specific than a rule with any number of classes, attributes, or pseudo-classes ("0-0-X-0") or element and pseudo-element selectors ("0-0-0-X").

Understanding this "0-0-0-0" format and how to calculate specificity is crucial for effectively writing CSS that behaves as expected, particularly when dealing with complex stylesheets where multiple rules might conflict.

## 6.2 Combining CSS Selectors

Combining CSS selectors is a powerful feature that allows developers to target HTML elements with precision and specificity beyond what can be achieved with single selectors. By combining selectors, you can define styles that apply only to elements that match specific criteria, making your stylesheets more efficient and easier to manage. Here are the most common ways to combine CSS selectors:

1. Descendant Selector (space)

The descendant selector targets elements that are nested within another element, regardless of the depth of nesting. It is represented by a space between two or more selectors.

Syntax: ancestor descendant

Example: div p will select all <p> elements that are inside a <div>, including those deeply nested within other elements.

2. Child Selector (>)

The child selector targets elements that are direct children of another element, ignoring any deeper descendants.

Syntax: parent > child

Example: ul > li will select <li> elements that are directly inside a <ul>, but not <li> elements nested further down.

3. Adjacent Sibling Selector (+)

This selector targets an element that is immediately preceded by a specific element at the same level of the hierarchy.

Syntax: previous + next

Example: h2 + p will select the first <p> element that directly follows an <h2>, but only if they are siblings (share the same parent).

4. General Sibling Selector (~)

The general sibling selector targets all elements that are siblings of a specified element, not just the immediately following sibling as with the adjacent sibling selector.

Syntax: sibling ~ sibling

Example: h2 ~ p will select all <p> elements that follow an <h2> element, as long as they are siblings.

5. Grouping Selector (,)

Grouping selectors allow you to apply the same style rules to multiple, different selectors by separating each selector with a comma. This is useful for reducing code redundancy.

Syntax: selector1, selector2, ...

Example: div, p, h1 will apply the same style rules to <div>, <p>, and <h1> elements.

6. Combining Pseudo-classes and Pseudo-elements with Selectors

Pseudo-classes and pseudo-elements can be combined with other selectors to apply styles based on state or to specific parts of an element.

Pseudo-classes: a:hover targets links when they are hovered over.

Pseudo-elements: p::first-line targets the first line of every <p> element.

Benefits of Combining Selectors

* Specificity: Enables more specific targeting of elements, which can be crucial for applying styles correctly in complex layouts.
* Efficiency: Reduces the need for adding classes or IDs to every element you want to style, keeping your HTML cleaner.
* Flexibility: Allows for dynamic styling based on the structure of the document (e.g., styling all paragraphs that follow a certain heading differently).

Understanding and utilizing combined selectors effectively can greatly enhance your CSS styling capabilities, allowing for sophisticated styling with minimal code.

## 6.3 CSS Positioning

CSS positioning is a fundamental aspect of web design that allows developers to control the layout of elements on a web page. It determines how elements are placed and interact with each other in the document flow. CSS provides several positioning schemes through the position property, each with its own behavior and use cases. Here are the primary positioning values in CSS:

1. Static Positioning

Value: static

Description: This is the default positioning for all elements. Elements are positioned according to the normal document flow, meaning they appear in the order they are written in the HTML, without any offset.

Use cases: Used when you want elements to follow the standard page layout without any special positioning.

2. Relative Positioning

Value: relative

Description: Positions the element relative to its normal position in the document flow. You can use the offset properties (top, right, bottom, left) to move the element from its original location, but it still occupies space in the layout as if it were in its static position.

Use cases: Useful for minor adjustments from an element's original position or for stacking elements with z-index.

3. Absolute Positioning

Value: absolute

Description: Removes the element from the normal document flow, and no space is created for the element in the page layout. It is positioned relative to its nearest positioned ancestor (i.e., an ancestor element with a position other than static). If no such ancestor exists, it's placed relative to the initial containing block (usually the viewport or the html element).

Use cases: Perfect for creating dropdown menus, modals, or other elements that should float over other elements.

4. Fixed Positioning

Value: fixed

Description: Similar to absolute positioning, but the element is positioned relative to the viewport. It stays in the same place even when the page is scrolled.

Use cases: Ideal for creating a navigation bar or a button that remains visible on the screen as the user scrolls.

5. Sticky Positioning

Value: sticky

Description: A hybrid of relative and fixed positioning. The element is treated as relative until it crosses a specified point, then it becomes fixed. For instance, a sticky header will scroll away with the content until the top of the viewport is reached, at which point it becomes fixed.

Use cases: Useful for headers, navigation bars, or any element that you want to remain accessible as the user scrolls through a page.

Offset Properties

Properties: top, right, bottom, left

Description: These properties determine the final location of an element with relative, absolute, fixed, or sticky positioning. They specify the distance from the element's original position to its new position based on the edges of its nearest positioned ancestor.

Z-Index

Property: z-index

Description: Controls the stacking order of positioned elements. Elements with a higher z-index will appear on top of those with a lower index. It only works on elements with a position value other than static.

Understanding CSS positioning is crucial for designing complex layouts and interactions on web pages. It allows developers to precisely control the placement of elements, ensuring that the design is both functional and visually appealing.

# 7 Advanced CSS

## 7.1 CSS Display

The CSS display property is a fundamental CSS property that specifies if/how an element is displayed on the web page. It controls the layout behavior of an element and its children, affecting both the box model and document flow. The display property can take various values, each defining how an element participates in the layout process. Here are some of the key values and their implications:

1. block

Description: Elements with display: block; start on a new line and stretch out as wide as their container allows, creating a "block" that fills the horizontal space. Block-level elements can have margins and padding on all sides.

Common Use: Used for elements that should stack vertically, like paragraphs, divs, and headers.

2. inline

Description: Elements with display: inline; do not start on a new line; they only take up as much width as necessary. Margins and paddings are respected horizontally but not vertically. Inline elements cannot have a width and height set.

Common Use: Suitable for content that should flow within a paragraph or alongside other elements without breaking the flow, like spans or links.

3. inline-block

Description: Combines behaviors of both inline and block. Elements are placed inline but behave as block-level elements regarding padding, margin, width, and height.

Common Use: Useful for creating layouts where elements sit side-by-side but also need to have their dimensions specified, like a navigation menu.

4. none

Description: The element is completely removed from the document flow and does not take up any space. It is as if the element does not exist in the document.

Common Use: Often used dynamically with JavaScript to show or hide elements without deleting them from the DOM.

5. flex

Description: Displays an element as a flex container and enables a flexible box layout. Direct children of the element become flex items, which can be aligned and distributed in various ways within the container.

Common Use: Ideal for creating complex layouts that need to be responsive and dynamically adjusted, such as modern web applications' interfaces.

6. grid

Description: Displays an element as a grid container and establishes a new grid formatting context for its contents. Children of the container become grid items placed into rows and columns according to the grid definition.

Common Use: Perfect for designing layouts that require more control over positioning and aligning elements in two dimensions, such as complex web page layouts with multiple rows and columns.

7. table, table-row, table-cell, etc.

Description: These values make the element behave like a table or a part of a table (row, cell, header, etc.). It's a way to use CSS for tabular layouts without using <table> HTML elements.

Common Use: Useful when you need tabular layouts but want to avoid the semantic implications of using <table>, <tr>, <td>, etc., in your HTML.

8. inline-flex and inline-grid

Description: These values combine the layout models of flex or grid with inline behavior, allowing the container to be laid out as an inline element while still controlling its contents as flex or grid containers.

Common Use: Beneficial when you need a flex or grid container to be laid out inline with other elements, such as inline lists or complex layout components within text blocks.

The display property is essential for defining how elements interact with each other and their parent containers. By understanding and using different display values, developers can create a wide range of layouts and component behaviors tailored to the needs of modern web design.

## 7.2 CSS Float

The CSS float property is a powerful tool used in web design to implement layout control, especially for wrapping text around images or creating multi-column layouts. When an element is floated, it is taken out of the normal flow of the document (though not entirely removed like with position: absolute;) and shifted to the left or right side of its container. Other elements in the container then flow around the floated element.

Key Concepts of Float

Values: The float property can have one of three values: left, right, or none. left and right move the element to the left or right edge of its containing block, respectively, allowing other content to wrap around it. none is the default value, meaning the element will not float and will remain in the normal flow.

Clearing Floats: Floating elements can affect the layout of elements that follow them in the document flow. The clear property is used on subsequent elements to specify which sides of the element floating elements are not allowed to float. Values for clear include left, right, both, and none, controlling the behavior based on the direction of the floated elements.

Containing Floats: A common challenge with floats is that a container might not automatically adjust its height to account for the floated elements inside it, as these are somewhat removed from the document flow. Techniques like adding a clearfix hack or setting the container's overflow property to a value other than visible (commonly overflow: hidden;) are used to ensure the container encompasses its floated elements.

Uses of Float

Text Wrapping: One of the most common uses of float is to wrap text around images. An image can be floated to the left or right, and text content will flow around it, creating a magazine-style layout.

Creating Columns: Before the advent of Flexbox and Grid, float was widely used to create columnar layouts. By floating elements such as divs to the left or right, you could align them side by side to form a grid-like structure.

Navigation Bars: Floats can be used to align navigation links horizontally across the top of a webpage, making for a simple horizontal navigation bar.

Considerations

Layout Challenges: While float is useful, it can sometimes lead to complex layout challenges, especially in complex designs or when trying to achieve vertical alignment. Modern CSS layout models like Flexbox and Grid offer more robust and flexible solutions for many of these layout needs without some of the side effects of floats.

Floats and Modern CSS: With the rise of Flexbox and Grid, the need to use float for layout purposes has decreased. However, float still has its place for specific tasks (like wrapping text around images) and remains a useful tool in a web developer's arsenal.

Understanding how to use float effectively, along with modern layout tools, allows developers to create intricate, responsive designs that work across different devices and screen sizes.

## 7.3 How to Create Responsive Websties

Creating responsive websites is essential in today's diverse device landscape, where users access web content on a wide range of screens, from mobile phones to large desktop monitors. A responsive website dynamically adjusts its layout, content, and functionalities to provide an optimal viewing experience across different devices. Here's a guide on how to create responsive websites:

1. Use a Fluid Grid Layout

Concept: Instead of using fixed-width layouts, use fluid grids that utilize percentages for widths. This approach allows the layout to adjust to the screen size.

Implementation: Design your grid layout based on a maximum layout width. Then, define element sizes as percentages of their containing elements to create a flexible and scalable layout.

2. Implement Flexible Images and Media

Concept: Ensure that images and media content are also responsive, scaling up or down depending on the screen size.

Implementation: Use CSS properties like max-width: 100%; and height: auto; for images. This allows them to scale down proportionally within their containing elements without exceeding the width of the container.

3. Use Media Queries

Concept: Media queries allow you to apply CSS styles based on the device's characteristics, such as its width, height, orientation, and resolution.

Implementation: Define breakpoints in your CSS where certain styles apply. For example, you might have one set of styles for screens wider than 600px and another for screens narrower than that, allowing for different layouts on mobile devices and desktops.

4. Utilize Flexible Typography

Concept: Text size should adjust to enhance readability on different devices.

Implementation: Use relative units like ems, rems, or viewport units (vw, vh) for font sizes. This way, text can scale based on the size of its container or the viewport, maintaining readability and user experience.

5. Optimize Navigation

Concept: Navigation menus should adapt to different screen sizes, sometimes changing form (e.g., from a horizontal bar to a dropdown menu).

Implementation: Use CSS and JavaScript to create collapsible menus for smaller screens. Consider tools like hamburger menus for mobile devices to save space.

6. Testing and Tools

Concept: Regular testing across different devices and browsers is crucial to ensure the website is truly responsive.

Implementation: Use browser developer tools to simulate various devices. Employ responsive design testing tools and conduct real-device testing to catch and fix issues.

7. Performance Optimization

Concept: Responsive designs must not only look good across devices but also load efficiently.

Implementation: Optimize images, minify CSS and JavaScript, and use lazy loading for images and media. Consider the impact of fonts and third-party scripts on loading times.

8. Progressive Enhancement

Concept: Start with a basic level of user experience that all browsers can support, then add enhancements for more capable devices.

Implementation: Structure your HTML to make sense without CSS. Add basic styles for older browsers, then enhance with advanced CSS and JavaScript features for modern browsers.

Creating responsive websites involves a combination of fluid layouts, flexible content, and media queries, alongside a mindful approach to navigation and performance optimization. As web technology evolves, so do the strategies for building responsive sites, but the core goal remains the same: to provide an optimal and seamless user experience across all devices.

## 7.4 Media Queries

Media queries are a cornerstone feature of responsive web design, enabling developers to apply CSS styles conditionally based on various factors of the user's device, such as its screen size, resolution, orientation, and more. They are part of CSS3 and allow for dynamic adjustments to the layout and appearance of a website to ensure it looks and functions well across all devices.

Syntax and Structure

A media query consists of a media type and can contain one or more expressions, which evaluate to either true or false. The result determines whether the specified CSS rules are applied. The basic syntax is as follows:

A black background with white text

Description automatically generated

Media Type: Indicates the type of device the document is being displayed on. Common media types include all, screen (for computer screens, tablets, smart phones, etc.), print (for pages being printed), and speech (for screenreaders that "read" the page out loud). The most commonly used type for responsive design is screen.

Expressions: Evaluate specific features of the user's device, such as width, height, orientation, and resolution. For example, (min-width: 600px) is an expression that returns true if the viewport width is 600 pixels or wider.

Common Uses of Media Queries

Responsive Breakpoints

Media queries are often used to create breakpoints at which a website's layout changes to accommodate different screen sizes. For instance, you might have a one-column layout for mobile devices and a multi-column layout for desktop screens.

A screen shot of a computer code

Description automatically generated

Adapting to Orientation

You can change styles based on the orientation of the device (portrait or landscape), which is particularly useful for optimizing the layout on tablets and smartphones.

A black screen with white text

Description automatically generated

High-Resolution Displays

Media queries enable you to tailor your site's graphics for devices with high-resolution displays, ensuring that images, fonts, and other elements remain crisp and clear.

A screen shot of a computer program

Description automatically generated

Best Practices

Mobile-First Approach: Start with styling for the smallest screens, and then use media queries to progressively enhance the design for larger screens. This approach is efficient and ensures compatibility with a wide range of devices.

Content Over Device: Focus on the content and how it should be presented rather than targeting specific devices. This approach is more sustainable as new devices with different screen sizes are constantly being released.

Use Relative Units: When specifying sizes within media queries, consider using relative units (e.g., ems, rems, percentages) instead of pixels for better scalability and readability across devices.

Media queries empower web developers to build flexible, responsive designs that adapt to the user's environment, providing an optimal browsing experience regardless of device or screen size.

# 8 Flexbox

## 8.1 Display: Flex

Foiwuebfwpuef

## 8.2 Flex Direction

foiwuebfwpuef

## 8.3 Flex Layout

foiwuebfwpuef

## 8.4 Flex Sizing

foiwuebfwpuef

# 9 Advanced CSS

## 9.1 Display: Grid

Foiwuebfwpuef

## 9.2 Grid Sizing

foiwuebfwpuef

## 9.3 Grid Placement

Foiwuebfwpuef

# 10 Bootstrap

## 10.1 What is Bootstrap

Bootstrap (CSS) is a popular open-source front-end framework primarily used for developing responsive and mobile-first web pages. It is a collection of CSS and JavaScript tools that provide a convenient way to design websites and web applications. Bootstrap offers a range of features including a grid system, pre-designed components (like buttons, forms, and navigation bars), and JavaScript plugins.

CSS Frameworks in general are pre-prepared libraries that are meant to be used as a base for designing websites. They offer a common structure for web design, making it faster and easier to create consistent and professionally looking websites. They usually come with a set of predefined styles and components that help in standardizing the layout and look of a web page.

Advantages of Bootstrap and CSS Frameworks:

1. Efficiency: They speed up the development process by providing pre-built components.
2. Responsiveness: They are designed to create websites that work well on different screen sizes and devices.
3. Consistency: They provide a uniform look across different web pages and components.
4. Customizable: Many frameworks, including Bootstrap, are customizable, allowing developers to tweak the design as per their requirements.
5. Community Support: Popular frameworks have large communities, providing extensive documentation, forums, and third-party tools.

Disadvantages:

1. Size and Performance: Frameworks can add extra weight to your website, potentially affecting load times and performance.
2. Learning Curve: Understanding and effectively using a framework requires time and effort.
3. Overhead: Sometimes, you might use only a small portion of the framework, but you need to include the whole library.
4. Design Limitation: Frameworks may limit creativity, leading to websites that look similar.
5. Dependency: Relying on a framework can create dependency, where updates or changes to the framework could impact your website.

## 10.2 Bootstrap In-depth

Wergwh

# 11 Web Design

‘you only have 3s to make a good impression’

Understanding colour theory

Typography and font choosing

UI design

UX design

# 12 Introduction to Javascript

Data types

Prompt keyword

Typeof keyword

Var keyword

Naming conventions e.g myName

String concatenation, length, slice, change the capitilsation of stings, .toUpperCase

Arithmetic, modulo

Functions, parameters and arguments x++

# 13 Intermediate Javascript

If else

Comparators and equality ===, ==, =, &&, ||

Arrays INCLUDE()

While loops for loops

# 14 The Document Object Model

Inline external internal

Selecting html elements w javascript

Separation of concerns

Text manipulation and text content property

Manipulating attributes

# 15 Advanced JS and DOM

Anonymous function

Higher order functions and passing functions as arguments,

Switch statements

Objects

Keydown

Callbacks and responding to events

Adding animations to website, timeout

# 16 jQuery

Minification, Selecting elements, manipulating text and attributes, adding even listeneds, adding removing elemds, animations

Before() after() prepend() append()

# 17 The Unix Command Line

Command line stuff, git bash, techniques and directory navigation

Touch, open, start and rm (windows eqiv)

# 18 Backend Web Development

What is backend – server, application, database

Tools: can use mainly any language

# 19 Node.js

Asynchronous event driven javascript runtime

REPL

Native modules

Node package manager

# 20 Express.js with Node.js

Servers

netstat -ano | findstr "LISTENING"

http – get (request), post (sending), put (replace), patch (patch), delete

nodemon

postman

middlewares

preprocessing, authentication, error, logging

# 21 EJS

Separation of concerns

# 22 Git, Github, Version Control

Git and GitHub

* Git: A version control system that allows multiple developers to work on a single project without interfering with each other. It tracks changes, allowing users to revert back to previous versions if necessary.
* GitHub: A platform for hosting and managing Git repositories. It provides a graphical interface and additional features like issue tracking, wikis, and pull requests.

Version Control

* Version Control: A system that records changes to a file or set of files over time so that you can recall specific versions later. It's essential for managing changes and coordinating tasks among multiple people.

Key Operations

* Pushing: Sending local repository changes to a remote repository. This is how changes are shared with others.
* Merging: Integrating changes from one branch into another. Merges can be done via pull requests on GitHub, which are then reviewed before integration.
* Commits: Snapshots of your repository files at a specific point in time. Committing saves your changes and allows Git to keep a history of your work.
* Pulls: Fetching and downloading content from a remote repository and updating your local repository to match that content.
* Cloning: Making a copy of a repository from a remote location to your local machine. This is often done at the start of working on a project to get a copy of the existing codebase.
* Branching: Creating separate branches allows developers to work on independent lines of development. Branches are useful for experimenting and making changes without affecting the main or 'master' branch.
* Forking: Copying a repository to your GitHub account. This allows you to freely experiment with changes without affecting the original project.
* .gitignore: A file that tells Git which files or directories to ignore in a project. Typically used to exclude build outputs, temporary files, and sensitive information from being added to the repository.

# 23 APIS

# 24 Databases

Data persistence

Sql – structured query language

# 25 SQL

Create, read, update, destroy

<https://www.w3schools.com/sql/sql_datatypes.asp>

create table and insert data

updating single valuesd and adding columns

delete

sql relationships foreign keys and inner joins