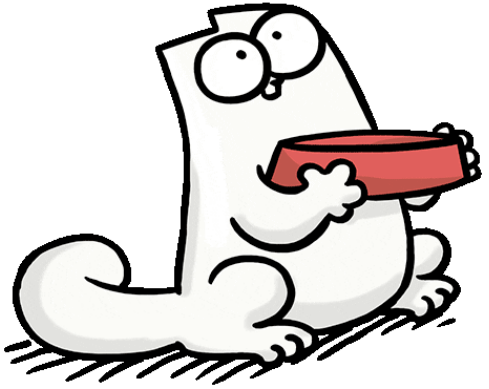
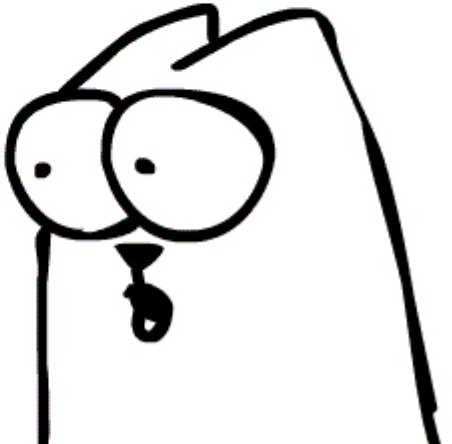
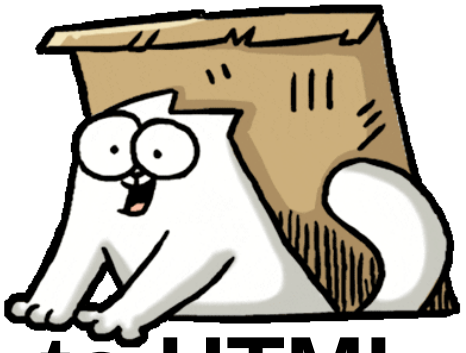
Start

Programming Basics with JavaScript



Duration: 5 Days



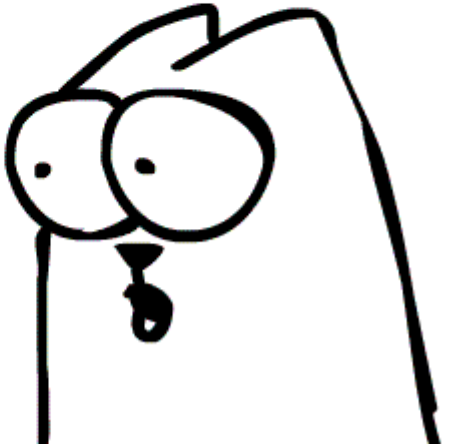


Introduction to HTML

HTML Basics

1. **Introduction to HTML**
    1. HTML Features
    2. Static Webpages
    3. Dynamic Webpages
    4. Elements
    5. Tags
    6. Lists
    7. Structure
    8. Attribute

1. **Introduction to CSS**
    1. CSS Style – Inline
    2. Internal
    3. External
    4. ID Selector
    5. Class Selector



Introduction to HTML

HTML, or Hyper Text Markup Language, serves as the backbone of the World Wide Web, providing a standardized way to structure content on web pages. Developed by Tim Berners-Lee in 1991, HTML is a key component of the trio that includes **CSS (Cascading Style Sheets)** and **JavaScript**, forming the foundation for modern web development.

Static Webpages

Static pages in HTML refer to web pages that remain unchanged unless manually modified by a developer. These pages are pre-rendered and delivered to users exactly as they were created. The content, layout, and design are fixed, providing a consistent experience for all users accessing the page. Static pages are primarily built using HTML, CSS for styling, and may include some client-side scripting using JavaScript.

Shortcomings of Static Pages:

While static pages have their advantages, they also come with certain limitations, especially when compared to dynamic and interactive web pages. Here are some shortcomings associated with static pages:

Static Pages in HTML:

Static pages in HTML refer to web pages that remain unchanged unless manually modified by a developer. These pages are pre-rendered and delivered to users exactly as they were created. The content, layout, and design are fixed, providing a consistent experience for all users accessing the page. Static pages are primarily built using HTML, CSS for styling, and may include some client-side scripting using JavaScript.

Limited Interactivity:

Static pages lack the dynamic and interactive features found in dynamic web applications. User interactions are typically limited to basic navigation, and real-time updates or personalized content are challenging to implement.

Maintenance Challenges:

Updating content on static pages requires manual intervention. Each change, whether it's a simple text update or a significant redesign, necessitates modifying the HTML files. This can be time-consuming and prone to errors, especially on large websites with numerous pages.

Scalability Issues:

As the size and complexity of a website grow, maintaining a large number of static pages becomes impractical. Any changes or improvements must be applied uniformly across all pages, making scalability a significant challenge.

Limited Data Handling:

Static pages are not well-suited for handling dynamic data or user-generated content. Implementing features like user authentication, dynamic forms, or real-time data updates is difficult without resorting to server-side scripting or more advanced technologies.

Poor User Engagement:

Without the ability to tailor content based on user interactions or preferences, static pages may struggle to engage users effectively. Personalization, feedback mechanisms, and interactive elements that enhance user experience are often absent.

Search Engine Optimization (SEO) Challenges:

While HTML provides the structure for content, static pages may face challenges in optimizing for search engines. Dynamic websites often have more opportunities for SEO optimization, such as creating search-friendly URLs or generating dynamic meta tags.

Server Load and Bandwidth Usage:

Every user request for a static page results in the server delivering the entire page, including content that remains consistent across multiple requests. This can lead to increased server load and higher bandwidth usage compared to dynamic websites that generate content on-the-fly.

Limited Content Updates:

Regularly updating content or providing fresh information is a manual process for static pages. This limitation can affect the timeliness of information and may impact the website's relevance to users.

While static pages have their place, particularly for simpler websites or when performance and security are critical, many modern websites and applications leverage dynamic content and interactivity to provide a more engaging user experience. As a result, developers often opt for a combination of static and dynamic elements or entirely dynamic approaches, depending on the specific requirements of the project.

Dynamic Web Pages:

Dynamic web pages are characterized by content that can change dynamically based on user interactions, preferences, or real-time data. Unlike static web pages, dynamic pages are generated on-the-fly by web servers, often utilizing server-side scripting languages, databases, and client-side technologies. Dynamic web pages allow for personalized and interactive user experiences, with content that can be modified without requiring manual changes to the HTML code.

Comparison between Static and Dynamic Web Pages:

Content Generation:

Static Pages: Content is fixed and predefined, created during the development phase. Any changes require manual modification of HTML files.
Dynamic Pages: Content is generated dynamically, often on the server side, allowing for real-time updates and personalized user experiences.

User Interaction:

Static Pages: Limited user interaction. Content remains the same for all users unless manually updated.
Dynamic Pages: Enhanced user interactivity with features such as forms, user authentication, and real-time updates.

Personalization:

Static Pages: Uniform content delivery to all users.
Dynamic Pages: Can deliver personalized content based on user preferences, location, and behaviour.

Data Handling:

Static Pages: Handling dynamic data or user-generated content is challenging.
Dynamic Pages: Capable of handling dynamic data, integrating with databases, and supporting user inputs.

Maintenance:

Static Pages: Manual updates are required for any content changes. Maintenance can be time-consuming, especially for large websites.
Dynamic Pages: Content updates can be automated, and changes often require modification of server-side scripts or database entries.

Scalability:

Static Pages: Scalability becomes challenging as the number of pages increases, as each page needs individual maintenance.
Dynamic Pages: More scalable, as changes can be managed centrally through server-side scripts and databases.

SEO Optimization:

Static Pages: May face challenges in optimizing for search engines, especially for dynamic or frequently changing content.
Dynamic Pages: Offer more opportunities for SEO optimization, including dynamic meta tags and search-friendly URLs.

Resource Usage:

Static Pages: Require less server processing since content is pre-rendered.
Dynamic Pages: May have higher server processing demands, especially for complex applications, due to real-time content generation.

Examples:

Static Pages: Brochure websites, simple informational sites.
Dynamic Pages: Social media platforms, e-commerce websites, web applications.

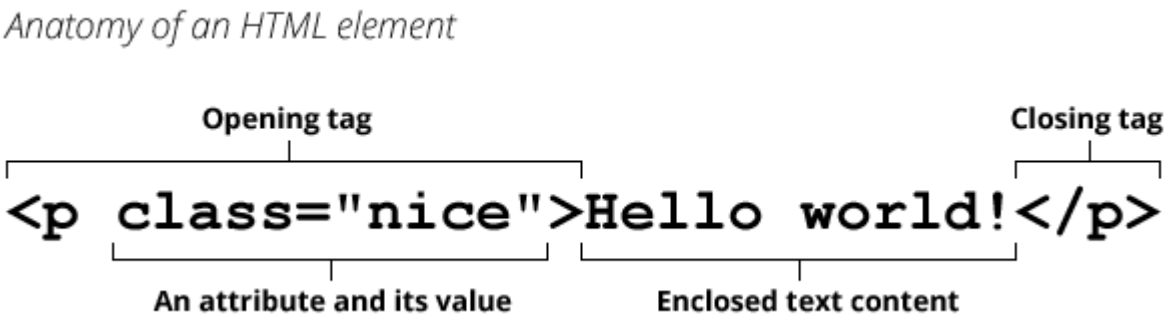
Loading Speed:

Static Pages: Generally faster loading times, as content is ready for immediate delivery.
Dynamic Pages: Loading times can vary based on server processing, database queries, and other dynamic elements.

In summary, while static web pages offer simplicity and ease of deployment, dynamic web pages provide a more interactive and personalized user experience. The choice between static and dynamic approaches depends on the specific requirements of a project, considering factors such as content volatility, user interactivity, and scalability needs. Often, modern web applications use a combination of both static and dynamic elements to achieve a balance between performance and flexibility.

HTML: Elements

An element is a part of a webpage. In XML and HTML, an element may contain a data chunk of text or an image, or perhaps nothing. A typical element includes an opening ta attributes, enclosed text content, and a closing tag.



Elements and tags are not the same things. Tags begin or end an element in source code, whereas elements are part of the DOM, the document model for displaying the page in the browser.

HTML: Tags

In HTML, a tag is used for creating an element.

The name of an HTML element is the name that appears at the beginning of the eleme and at the end of the element's end tag (if the element has an end tag). For example, th start tag and end tag is the name of the HTML paragraph element. Note that an element name in an end tag is preceded by a slash character: , and that for void elements, the end tag is neither required nor allowed.

Void element

A void element is an element in HTML that cannot have any child nodes (i.e., nested elements or text nodes). Void elements only have a start tag; end tags must not be specified for void elements.

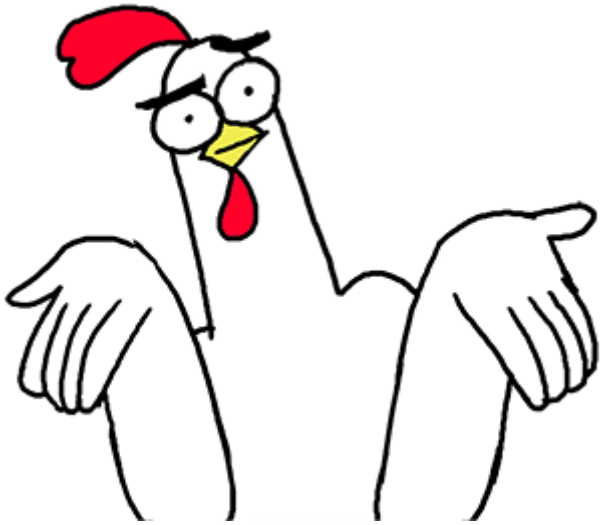
Although there is no way to mark up a void element as having any children, child nodes
added programmatically to the element in the DOM using JavaScript. But that is not a g
practice, as the outcome will not be reliable.

The void elements in HTML are as follows:

<area>
<base>

<col>
<embed>
<hr>
<img>
<input>
<link> etc...

HTML: Lists



HTML Lists:

HTML provides three types of lists to organize and structure content:

1. ordered lists (<ol>)
2. unordered lists (<ul>)
3. definition lists (<dl>)

Lists help in presenting information in a structured and readable manner. Here's an explanation of each type of list:

Ordered List (<ol>):

An ordered list is used to create a list where the order of items matters. Each list item with a number or another type of marker. The <ol> tag is used to define an ordered list, tags are used to define each item within the list.

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

Unordered List (<ul>):

An unordered list is used when the order of items doesn't matter. Each list item is ma
bullet point or another type of marker. The <ul> tag is used to define an unordered list,
tags are used to define each item within the list.

```
<ul>
   <li>Red</li>
   <li>Green</li>
   <li>Blue</li>
</ul>
```

Definition List (<dl>):

A definition list is used to define terms and their corresponding definitions. It consists <dt> (definition term) and <dd> (definition description) tags.

Result:

HTML

Hyper Text Markup Language

CSS

Cascading Style Sheets

```
<dl>
   <dt>HTML</dt>
   <dd>HyperText Markup Language</dd>
   <dt>CSS</dt>
   <dd>Cascading Style Sheets</dd>
</dl>
```

Nested Lists:

Lists can also be nested within one another to create more complex structures. For exa
can nest an unordered list within an ordered list or vice versa.

Example of Nested Lists:

```
<ol>
   <li>First item</li>
   <li>Second item
      <ul>
         <li>Nested item 1</li>
         <li>Nested item 2</li>
      </ul>
   </li>
   <li>Third item</li>
</ol>
```

Result:

1. First item
2. Second item
   - Nested item 1
   - Nested item 2
3. Third item

These HTML list elements are fundamental for structuring and presenting information on web pages, offering flexibility in organizing content based on different criteria.

HTML: Structure

HTML (Hyper Text Markup Language) has a basic structure that serves as the foundation for creating web pages. The structure consists of several key elements that organize content and provide essential information to web browsers. Here's an explanation of the basic HTML structure along with examples:

Basic HTML Structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Your Page Title</title>
</head>
<body>
    <!-- Your content goes here -->
</body>
</html>
```

<!DOCTYPE html>: Declaration specifying the HTML version being used (HT
case).

<html lang="en">: The root element of the HTML document. The lang attribu
the language of the document, such as "en" for English.

<head>: Contains meta-information about the HTML document, such as character set, viewport settings, and the page title.

<meta charset="UTF-8">: Specifies the character set used in the document (UTF-8 for universal character encoding).

<meta name="viewport" content="width=device-width, initial-scale=1.0">: Defines the viewport settings for responsive design on various devices.

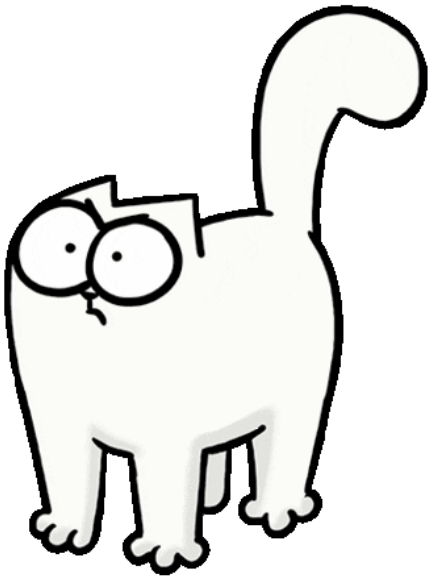
<title>Your Page Title</title>: Sets the title of the web page, which appears in the browser tab.

<body>: Contains the main content of the HTML document.

<!-- Your content goes here -->: This is a comment that allows you to add comments to your HTML code. Comments are not displayed in the browser.

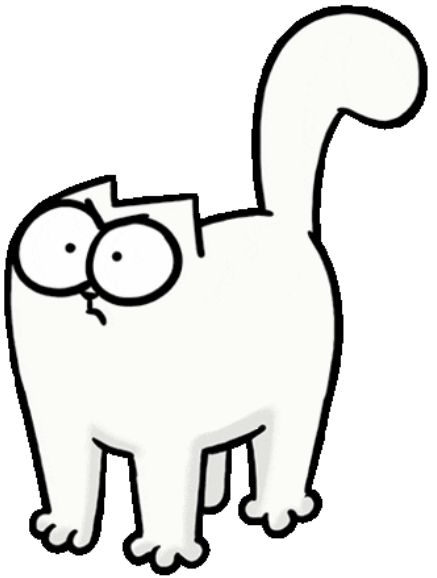
Simple HTML Page

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My First HTML Page</title>
</head>
<body>
    <h1>Hello, World!</h1>
    <p>This is a simple HTML page.</p>
</body>
</html>
```



HTML page with Lists

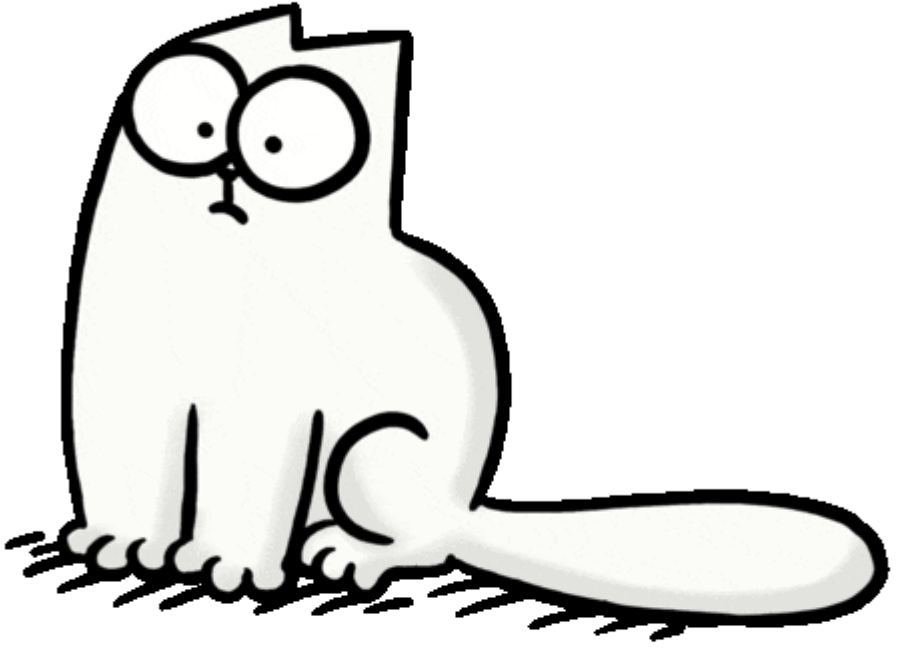
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>List Example</title>
</head>
<body>
    <h1>My Favorite Fruits</h1>
    <ul>
        <li>Apple</li>
        <li>Banana</li>
        <li>Orange</li>
    </ul>
</body>
</html>
```



HTML page with Links

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Link Example</title>
</head>
<body>
    <h1>Useful Links</h1>
    <ul>
        <li><a href="https://www.brainstrom.in">Example Website</a></li>
        <li><a href="https://www.brainstrom.in">Brainstrom Technologies</a></li>
    </ul>
</body>
</html>
```

HTML: Attributes



HTML Attributes:

In HTML, attributes provide additional information about HTML elements. They are always included in the opening tag of an element and are used to modify the behavior or appearance of the element. Attributes are made up of a name and a value, separated by an equal sign (=), and are enclosed in double or single quotes.

Examples of HTML Attributes:

1. src Attribute for Images:

The **src** attribute is commonly used with the <img> tag to specify the source (URL or file path) of an image.

<img src="example.jpg" alt="An example image">

Here, src tells the browser where to find the image file, and alt provides alternative text for accessibility.

2. href Attribute for Links:

The **href** attribute is used with the <a> tag to define the hyperlink reference, indicating the destination URL.

Visit Brainstrom Technologies Website

The **href** attribute guides the browser to the linked web page when the user clicks on the link.

3. width and **height** Attributes for Images:

These attributes, **width** and **height**, are often applied to the <img> tag to set the dimensions of an image.

<img src="example.jpg" alt="An example image" width="300" height="200">

They define the width and height of the image in pixels, influencing its display size.

4. **class** and **id** Attributes for Styling and Scripting:

The class and id attributes are used to identify HTML elements for styling with CSS or targeting with JavaScript.

This is an important paragraph.

The class and id attributes help apply specific styles or interact with elements using scripts.

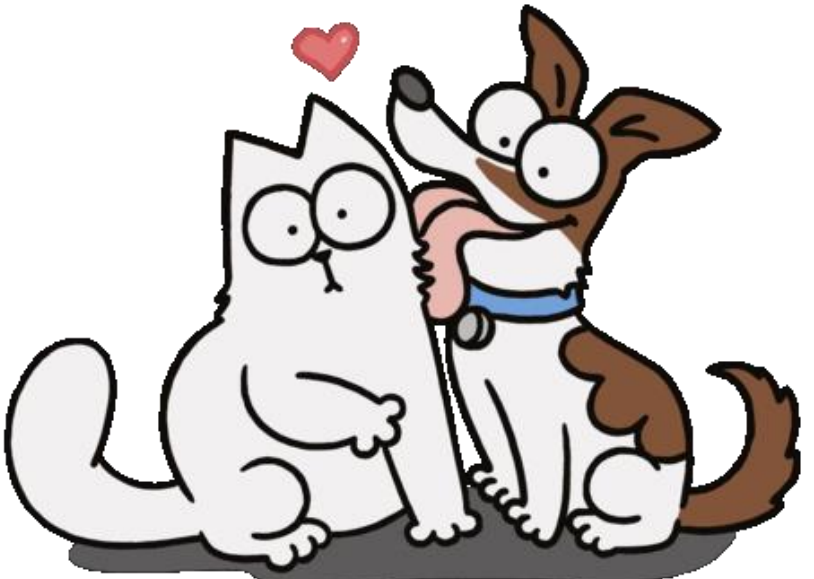
5. **type** Attribute for **Input** Elements:

The type attribute is often used with the <input> tag to specify the type of input field, such as text, password, or checkbox.

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

The **type** attribute ensures the correct behavior and appearance of the input field.

These are just a few examples of HTML attributes. Each attribute serves a unique purpose, enhancing the functionality and appearance of HTML elements. Isn't it fascinating how these little details play a big role in shaping the behavior of web pages? Feel free to explore more attributes and experiment with different combinations to see their impact on the web page!



Introduction to CSS

CSS (Cascading Style Sheets):

CSS, or Cascading Style Sheets, is a stylesheet language used to describe the presentation and formatting of a document written in HTML or XML. It enables web developers to control the layout, appearance, and styling of multiple web pages from a single external file or within the HTML document itself. CSS helps separate the structure (HTML) and presentation (CSS) of a web page, making it more modular, maintainable, and efficient.

Key Features of CSS:

1.Selective Styling:

- CSS allows the targeting of specific HTML elements, classes, or IDs to apply styles selectively. This flexibility enables precise control over the appearance of different parts of a web page.

2.Style Inheritance:

- CSS properties can be inherited from parent elements to their child elements, creating a consistent and organized styling hierarchy. This inheritance mechanism simplifies styling and reduces redundancy.

1.Modularity and Reusability:

- Styles can be defined in separate CSS files, promoting modularity. Reusable styles can be applied across multiple pages, ensuring a consistent look and feel throughout a website.

2.Responsive Design:

- CSS plays a crucial role in creating responsive web designs. It allows developers to define styles based on the device's characteristics, such as screen size, enabling web pages to adapt to various display sizes.

3.Ease of Maintenance:

- By separating style concerns from the HTML structure, CSS makes it easier to update and maintain web pages. Changes to the design can be applied globally by modifying the CSS file, without altering the HTML markup.

4.Cross-Browser Compatibility:

- CSS helps address inconsistencies in the rendering of web pages across different browsers. It provides a standardized way to define styles, reducing the need for browser-specific hacks.

Reason for CSS:

Before the introduction of CSS, styling was primarily done using HTML attributes like <font>, <color>, and <align>. However, this approach had several drawbacks:

1. Limited Control:

- HTML attributes for styling provided limited control and flexibility. They were designed for structural purposes, not for extensive formatting and presentation.

2. Redundancy and Repetition:

- With inline styling, the same styles had to be repeated for every instance of a particular element, leading to redundancy and increased file sizes.

3. Maintenance Challenges:

- Making changes to the styling of a website was laborious and error-prone. modification required updating each HTML tag individually.

4. Poor Separation of Concerns:

- Mixing HTML structure with styling information violated the principle of separation of concerns, making code less modular and harder to maintain.

CSS was introduced to address these challenges by providing a clear and efficient way to separate structure from presentation. It allows developers to create more sophisticated and consistent designs, improve maintainability, and enhance the overall user experience on the web. The adoption of CSS has significantly contributed to the evolution of web design and development practices.

CSS: Inline

CSS Inline Styles:

CSS inline styles involve applying styles directly to individual HTML elements using the style attribute. With inline styles, you can define the presentation properties (e.g., color, font, margin) directly within the HTML tag, providing a way to override external or iternal stylesheet styles on a per-element basis.

```
<tagname style="property: value; property: value;">
   <!-- Content of the element -->
</tagname>
```

In the syntax:

tagname is the HTML tag for the element you want to style.
property: value; pairs define the specific style properties and their values.

```
<p style="color: blue; font-size: 16px;">This is a blue paragraph with a font size of 16 pixels.</p>
```

In this example, the inline styles are applied directly to the <p> (paragraph) element, specifying a blue text color and a font size of 16 pixels.

Implementation:

You can implement inline styles in HTML by adding the style attribute to the HTML tag you want to style. Here's how you can do it:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Inline Styles Example</title>
</head>
<body>
   <h1 style="color: green; text-align: center;">Welcome to My Website</h1>

   <p style="color: blue; font-size: 16px;">This is a blue paragraph with a font size of 16 pixels.</p>

   <div style="background-color: #f0f0f0; padding: 10px;">
      <p style="font-weight: bold;">A styled div with a light gray background and bold text.</p>
   </div>

</body>
</html>
```

In this example:

- The <h1> element has inline styles for a green text color and center alignment.
- The first <p> element has inline styles for a blue text color and a font size of 16 pixels.
- The element has inline styles for a light gray background color and bold text.

While inline styles can be useful for quick styling adjustments on a per-element basis, they are not typically recommended for extensive styling across an entire website. External or internal stylesheets are preferred for maintaining a more organized and maintainable code structure. Inline styles can be useful in situations where you need to override specific styles for a particular element without affecting the rest of the page.

CSS: External

CSS External Concept:

The external concept in CSS refers to the practice of placing CSS code in a separate external file with a .**css** extension. This external CSS file is then linked to the HTML document using the <**link**> element in the document's <**head**> section. This external file contains all the style rules for the entire website, allowing for a more organized and modular approach to styling.

Benefits of External CSS:

1.Separation of Concerns:

- External CSS promotes the separation of concerns by keeping the structure (HTML) and presentation (CSS) of a web page in separate files. This separation makes the code more modular, readable, and easier to maintain.

2.Reusability:

- With external CSS, styles can be reused across multiple pages of a website. A single stylesheet can define the styling for an entire website, ensuring a consistent look and feel.

3. Easy Maintenance:

- Making changes to the styling of a website becomes more efficient with extern
  Modifications can be made in a single file, and those changes automatically ap
  pages linked to that stylesheet.

4. Improved Page Loading:

- External CSS files are cached by the browser after the first request. Subsequent page views benefit from faster loading times, as the browser does not need to reload the entire stylesheet.

5. Accessibility:

- External CSS allows developers to optimize styles for accessibility features. Styles can be adjusted or overridden for different devices, ensuring a better user experience for a diverse audience.

6. Consistency Across the Site:

- External CSS ensures a consistent design and layout throughout the entire web
  updates or changes made to the stylesheet reflect uniformly across all linked p

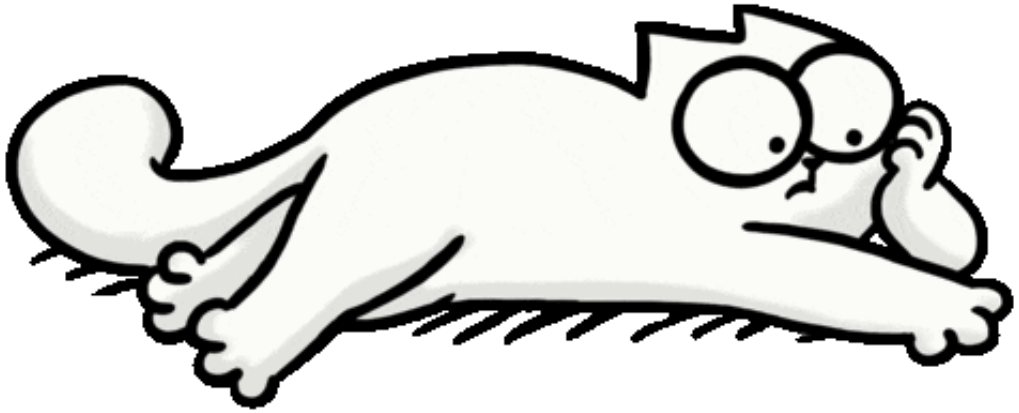
Example Implementation: Here's an example of implementing external CSS:

Create an External CSS File (styles.css):

```
/* styles.css */
body {
    font-family: 'Arial', sans-serif;
    background-color: #f0f0f0;
}

h1 {
    color: #336699;
}

p {
    font-size: 16px;
    line-height: 1.5;
}
```



Link the External CSS File in HTML:

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

    <h1>Welcome to My Website</h1>

    <p>This is a paragraph with consistent styling across the entire site.</p>

</body>
</html>
```

In this example, the styles.css file contains the styling rules, and the <link> element in the HTML file connects to this external stylesheet. The styling defined in styles.css applies to the HTML elements on the page.

External CSS provides a clean, modular, and maintainable way to apply styles to a website, making it an essential part of modern web development practices.

CSS: ID Selector

ID Selector in CSS:

In CSS, the ID selector is used to target a specific HTML element with a unique identifier. The unique identifier is defined by the id attribute in the HTML, and it must be unique within the entire document. The ID selector is denoted by a hash symbol (#) followed by the ID value.

Syntax:

```
#yourID {
   /* Styles for the element with the specified ID */
}
```

Example: Consider the following HTML element with an ID:

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

  <div id="uniqueDiv">
     <p>This is a paragraph inside a div with a unique ID.</p>
  </div>

</body>
</html>
```

Now, you can apply styles using the ID selector in a CSS file (styles.css):

```
/* styles.css */
#uniqueDiv {
    background-color: #ffeecc;
    padding: 20px;
    border: 1px solid #e0e0e0;
}
```

In this example:

The HTML element has the id attribute set to "uniqueDiv".
The CSS ID selector #uniqueDiv targets this specific element.
The styles in the CSS file define a background color, padding, and border for the element with the "uniqueDiv" ID.

Benefits of ID Selector:

- ID selectors are particularly useful when you want to apply unique styles to a specific element on a page.

- They provide a straightforward and efficient way to style a particular element without affecting other elements on the page.

- Since IDs must be unique, the ID selector allows for precise targeting of a single element.

However, it's important to use ID selectors judiciously and reserve them for truly unique elements. Overusing IDs for styling purposes can lead to less maintainable and more tightly coupled code. For general styling that can be applied to multiple elements, classes or other selectors may be more appropriate.

CSS: Class Selector

Class Selector in CSS:

In CSS, the class selector is used to target HTML elements based on their assigned class attribute. Unlike IDs, classes are not required to be unique, allowing multiple elements to share the same class. The class selector is denoted by a period (.) followed by the class name.

Syntax:

```
.className {
   /* Styles for elements with the specified class */
}
```

Example: Consider the following HTML elements with a shared class:

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

  <p class="highlight">This is a paragraph with a highlight class.</p>
  <p class="highlight">Another paragraph with the same highlight class.</p>
  <p>This paragraph does not have the highlight class.</p>

</body>
</html>
```

Now, you can apply styles using the class selector in a CSS file (styles.css):

```
/* styles.css */
.highlight {
    color: #ff0000; /* Red text color */
    font-weight: bold;
}
```

In this example:

The HTML <p> elements have the class attribute set to "highlight".
The CSS class selector .highlight targets all elements with the "highlight" class.
The styles in the CSS file define a red text color and bold font weight for elements with the "highlight" class.

Benefits of Class Selector:

Class selectors are versatile and allow you to style multiple elements with the same class.
They are useful for applying consistent styles to groups of elements, making the styling more modular and reusable.
Unlike IDs, classes do not need to be unique, allowing the same class to be applied to multiple elements.

Usage Tips:

- Use class selectors for styling elements that share similar characteristics or styles.
- Classes are suitable for applying styles to multiple elements across a page or website.
- Combine class selectors with other selectors to create more specific targeting when needed.

Classes are a fundamental part of CSS and play a key role in creating maintainable and scalable stylesheets. They contribute to a modular and reusable approach to styling web pages.


"That's all Folks!"