Agenda

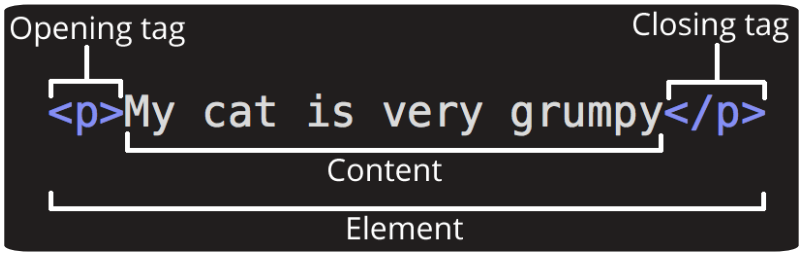
1. What is HTML?
2. HTML Basics
   a) Anatomy of an HTML element
   b) Attributes
   c) Empty Elements
   d) Anatomy of an HTML document
   e) Reserved Characters
3. HTML Elements
4. Structure of an HTML Document
5. Document Object Model (DOM)
   a) The Window Interface
   b) The Document Interface
   c) The Element Interface
6. HTML DOM API
7. HTML Debugging and Validation
8. Further Reading

What is HTML?

- Standard **markup language** for documents designed to be displayed in a **web browser**.
- Describes the **structure** of a web page semantically.
- HTML **elements** are the building blocks of HTML pages.
- HTML elements are delineated by ***tags***, written using **angle brackets**.
- HTML can embed programs which affects the **behavior** and **content** of web pages.
- Inclusion of **CSS** defines the **look** and **layout** of content.

HTML Basics

Anatomy of an HTML element



- **The opening tag:** This states where the element begins or starts to take effect.
- **The closing tag:** This states where the element ends. Failing to add a closing tag can lead to strange results.
- **The content:** This is the content of the element.
- **The element:** The opening tag, the closing tag, and the content together comprise the element.

Attributes

- Attributes contain **extra information** about the element that you don't want to appear in the actual content.
- An attribute should always have the following:
  - A space between it and the element name or the previous attribute.
  - The attribute **name** followed by an equal sign.
  - The attribute **value** wrapped by opening and closing quotation marks.
- Simple attribute values that **don't** contain ASCII whitespace (or any of the characters " ' ` = < >) can remain **unquoted**, but it is not recommended.

Attribute

```
<p class="editor-note">My cat is very grumpy</p>
```

Empty Elements

- Some elements have no content and are called **empty elements**.
- Example: **<img>** element
  - This contains two attributes, but there is **no closing** </img> **tag** and no inner content.
  - This is because an image element doesn't wrap content to affect it.
  - Its purpose is to embed an image in the HTML page in the place it appears.

```
<img
src="images/firefox-
icon.png" alt="My test
image">
```

Anatomy of an HTML document

- **<!DOCTYPE html>** — It is a required preamble.
- **<html></html>** — This element wraps all the content on the entire page and is sometimes known as the **root** element.
- **<head></head>** — This element acts as a container for all the stuff you want to **include** on the HTML page that *isn't* the content you are showing to your page's viewers.
- **<meta charset="utf-8">** — This element sets the character set your document should use to UTF-8 which includes most characters. Essentially, it can now handle **any textual content** you might put on it.
- **<title></title>** — This sets the title of your page, which is the title that appears in the **browser tab** the page is loaded in.
- **<body></body>** — This contains ***all* the content** that you want to show to web users , whether that's text, images, videos, games, playable audio tracks, or whatever else.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My test page</title>
  </head>
  <body>
    <img src="images/firefox-icon.png
test image">
  </body>
</html>
```

Reserved Characters

- The following special characters are reserved in HTML - **"**, **'**, **&**, **<**, **>**.
- If one of these characters is used, the browser will try to interpret it as HTML.
- Use the **entity name** or **entity number** when you want to output any of these reserved characters.
- **Comments** are represented in HTML as content between '**<!--**' and '**-->**'.

Result	Description
	non-breaking space
<	less than
>	greater than
&	ampersand
"	double quotation mark
'	single quotation mark (apostrophe)
¢	cent
£	pound
¥	yen
€	euro
©	copyright
®	registered trademark

< HTML Elements >

<head>

- It contains information such as:
  - the page <title>
  - links to CSS
  - links to custom favicons
  - metadata <meta>
- Web browsers use information contained in the head to render the HTML document correctly.
- Social networks have their own proprietary metadata.
  - Facebook - Open Graph Data
  - Twitter – Twitter Cards

<link>

- The **<link>** element specifies relationships between the current document and an external resource.
- Common attributes:
  - **href** - path to the stylesheet
  - **rel** - (relationship) how the item being linked to is related to the containing document.
  - **rel="preload"** - the browser should preload this resource
  - **as** - specific class of content being fetched.
- A <link> element can occur either in the <head> or <body> element, depending on whether it has a link type that is **body-ok.**

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

```
<link rel="icon" href="favicon.ico">
```

```
<link rel="preload" href="myFont.woff2" as="font"
      type="font/woff2" crossorigin="anonymous">
```

<script>

- The <script> element should go into the head.
- It should include a **src** attribute containing the path to the JavaScript you want to load.
- **defer** instructs the browser to load the JavaScript after the page has finished parsing the HTML.
- This is useful as it makes sure that the HTML is all loaded before the JavaScript runs

```html
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <script type="text/javascript">
        alert('Hello world');
    </script>
</head>
<body>

</body>
</html>
```

```html
<script src="my-js-file.js" defer></script>
```

Marking Up Text

```
<h1>The Crushing Bore</h1>

<p>By Chris Mills</p>

<h2>Chapter 1: The dark night</h2>

<p>It was a dark night. Somewhere, an owl hooted. The rain lashed down on
the ...</p>
```

- Each paragraph must be wrapped in a <p> element.
- Each heading must be wrapped in a heading element -<h1>, <h2>, <h3>, <h4>, <h5>, and <h6>.
- Every list starts off with a <ul> or <ol> element.
- Each list item is wrapped in a <li> element.

```
<ol>
  <li>Drive to the end of the road</li>
  <li>Turn right</li>
  <li>Go straight across the first two roundabouts</li>
  <li>Turn left at the third roundabout</li>
  <li>The school is on your right, 300 meters up the road</li>
</ol>
```

&

- The ** ** element is the generic container for flow content.
- It has **no effect** on the content or **layout** until styled in some way.
- Instead, it's used to **group content** so it can be easily styled.
- The **** element is a generic inline container for phrasing content.
- is very much like a element, but is a block-level element whereas a is an inline element.

```html
<div class="warning">
    <img src="/media/examples/leopard.jpg"
         alt="An intimidating leopard.">
    <p>Beware of the leopard</p>
</div>
```

```html
<p>Add the <span
class="ingredient">basil</span>, <span
class="ingredient">pine nuts</span> and <span
class="ingredient">garlic</span> to a blender
and blend into a paste.</p>

<p>Gradually add the <span
class="ingredient">olive oil</span> while
running the blender slowly.</p>
```

Block-level Element

- A Block-level element occupies the **entire horizontal space** of its container, and vertical space equal to the **height of its contents.**
- Browsers typically display the block-level element with a **newline** both before and after the element.
- A block-level element always starts on a new line and takes up the **full width available**.

The following paragraph is a

block-level element;

its background has been colored to display both the beginning and end of the block-level element's influence.

```
<div>The following paragraph is a <p class="highlight">block-level
element;</p>
its background has been colored to display both the beginning and end of
the block-level element's influence.</div>
```

Inline Element

- Inline elements are those which only occupy the space **bounded** by the tags defining the element, instead of breaking the flow of the content.
- An inline element **does not** start on a new line and only takes up as much width as necessary.
- Generally, inline elements may contain only **data** and **other inline elements**.

The following span is an inline element; its background has been colored to display both the beginning and end of the inline element's influence.

```
<div>The following span is an <span class="highlight">inline
element</span>;
its background has been colored to display both the beginning and end of
the inline element's influence.</div>
```

<a>

- The **<a>** element (or *anchor* element), with its href attribute, creates a hyperlink to web pages, files, email addresses, locations in the same page, or anything else a URL can address.
- Content within each <a> **should** indicate the link's destination.

Linking to an absolute URL

HTML

```
<a href="https://www.mozilla.com">
  Mozilla
</a>
```

Linking to an element on the same page

```
<!-- <a> element links to the section below -->
<p><a href="#Section_further_down">
  Jump to the heading below
</a></p>

<!-- Heading to link to -->
<h2 id="Section_further_down">Section further down</h2>
```


- The **<img>** element embeds an image into the document.
- The src attribute is **required** and contains the path to the image you want to embed.
- The alt attribute holds a text description of the image, which isn't mandatory but is **incredibly useful** for accessibility.
- <img> is a replaced element - **content** is not affected by the current document's styles.

```
<img src="favicon144.png"
     alt="MDN logo">
```

<form>

- The **<form>** element represents a document section containing interactive controls for submitting information.
- **action** should contain the URL that processes the form submission.
- **method** contains The HTTP method to submit the form with.

HTML Demo: <form>

HTML CSS

```html
<form action="" method="get" class="form-example">
  <div class="form-example">
    <label for="name">Enter your name: </label>
    <input type="text" name="name" id="name" required>
  </div>
  <div class="form-example">
    <label for="email">Enter your email: </label>
    <input type="email" name="email" id="email" required>
  </div>
  <div class="form-example">
    <input type="submit" value="Subscribe!">
  </div>
</form>
```

<iframe>

- The **<iframe>** element represents a nested browsing context - embedding another HTML page into the current one.
- Each embedded browsing context has its own session history and document.
- The browsing context that embeds the others is called the ***parent*** browsing context.
- The **topmost** browsing context is usually the browser window, represented by the Window object.
- Since each browsing context is a **complete document** environment, every <iframe> in a page requires **increased memory** and other **computing resources**.

HTML

```
<iframe src="https://example.org"
            title="iframe Example 1" width="400" heig
</iframe>
```

Result

<table>

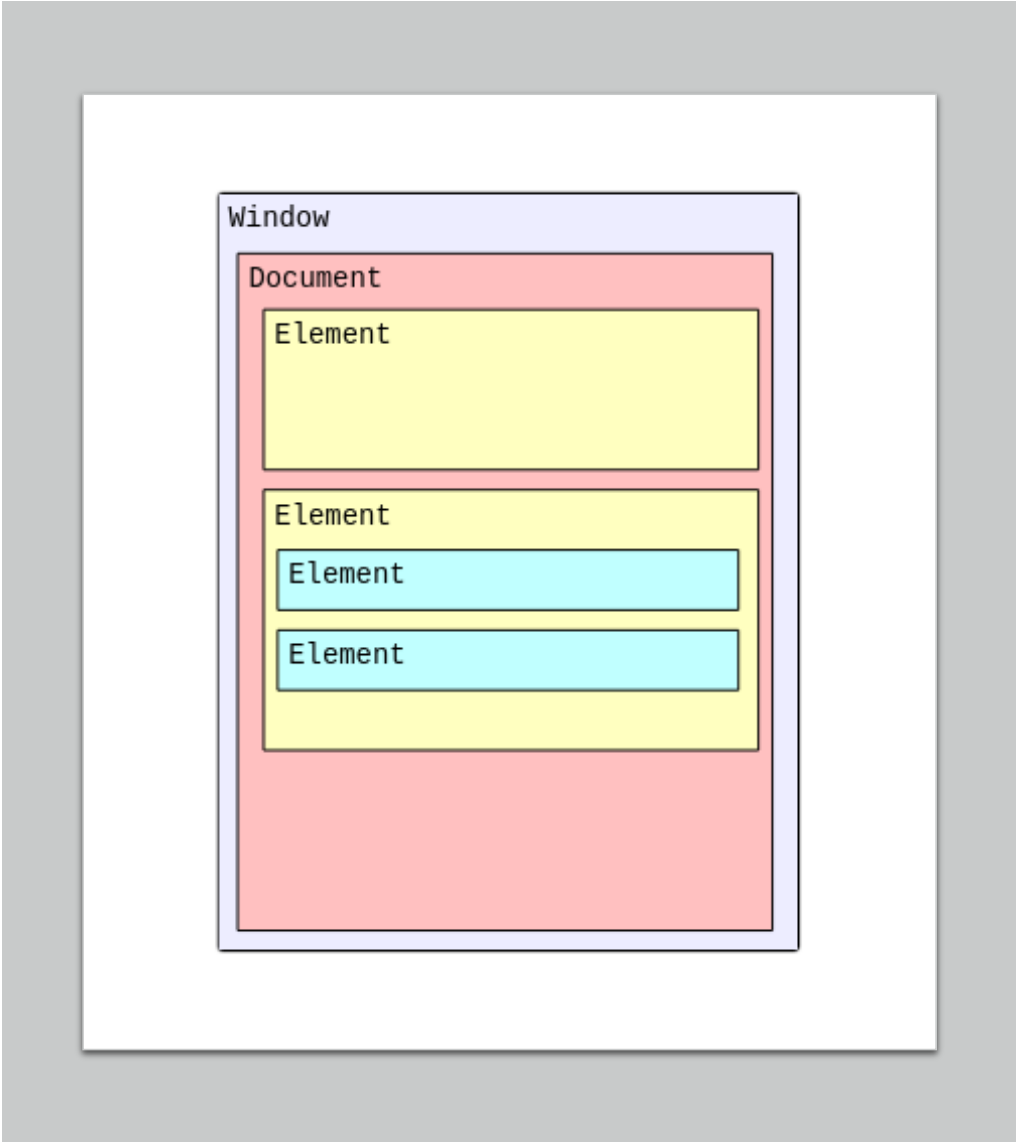
- The content of every table is enclosed by **<table></table>**.
- The smallest container inside a table is a table cell, which is created by a **<td>** element.
- Each row needs to be wrapped in an **<tr>** element, with each cell contained in a **<td>**.
- You can use the **<th>** element to denote a header.
- Define styling information for an entire column of data all in one place using the **<col>** and **<colgroup>** elements.

```
<table>
  <colgroup>
    <col>
    <col style="background-color: yellow">
  </colgroup>
  <tr>
    <th>Data 1</th>
    <th>Data 2</th>
  </tr>
  <tr>
    <td>Calcutta</td>
    <td>Orange</td>
  </tr>
  <tr>
    <td>Robots</td>
    <td>Jazz</td>
  </tr>
</table>
```

</ HTML Elements >

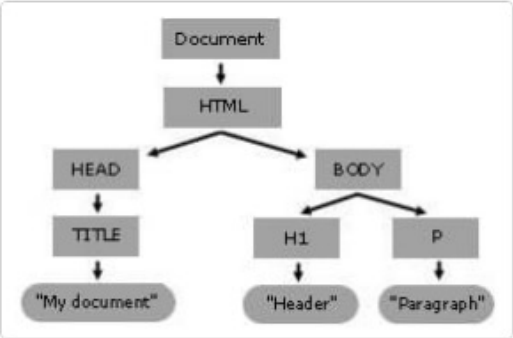
Structure of an HTML Document

- The **Document Object Model (DOM)** is an architecture that describes the **structure** of a document.
- Each document is represented by an instance of the interface Document.
- A document consists of a hierarchical tree of **nodes.** A **node** is a fundamental record representing a single **object** within the document.
- Each node is based on the Node interface which provides **properties** as well as **methods** for creating, deleting, and organizing nodes within the DOM.
- Nodes **don't** have any concept of **including** the **content** that is displayed in the document. A node can **represent** the visual content via the Element interface.
- An Element instance represents a **single element** in a document created using either HTML or XML.



Document Object Model (DOM)

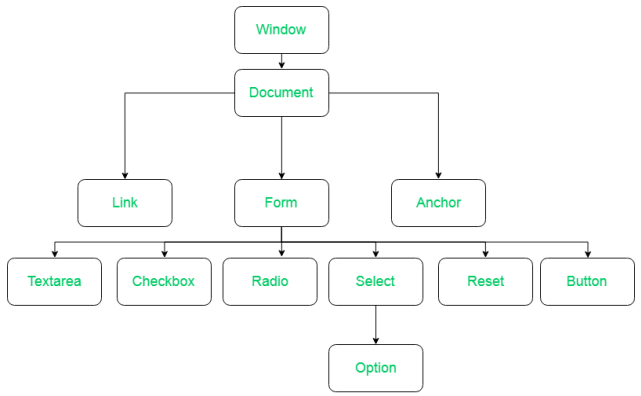
- The **DOM** connects web pages to scripts or programming languages by representing the structure of a document.
- The DOM **represents** a document with a **logical tree.**
- Each branch of the tree ends in a **node**, and each node contains **objects**.
- DOM methods allow [programmatic access]() to the tree with which you can change the document's **structure**, **style**, or **content**.
- The DOM is built using **multiple APIs** that work together.
- The **core DOM** defines the entities describing any document and the objects within it.
- This is **expanded** upon as needed by other APIs that add **new features** and **capabilities** to the DOM.




```
<html>
<head>
  <title>My Document</title>
</head>
<body>
  <h1>Header</h1>
  <p>Paragraph</p>
</body>
</html>
```

The Window Interface

- The **Window** interface represents a window **containing** a DOM document.
- The **document property** points to the **DOM document** loaded in that window.
- The Window interface contains a variety of **functions**, **namespaces**, **objects**, and **constructors**.
- In a tabbed browser, **each tab** is represented by its own **Window object.**
- Although, even in a tabbed browser, some properties and methods still apply to the **overall window** that contains the tab.
- A global variable, **window**, representing the window in which the script is **running**, is exposed externally.



The Document Interface

- The **Document** interface represents any web page **loaded** in the browser and serves as an **entry point** into the web page's content, which is the DOM tree.
  - **document.createElement()** method creates the HTML element specified by *tagName.*
  - **document.getElementById()** returns an Element object representing the element whose id property matches the specified string.
  - **document.querySelector()** returns the first Element within the document that matches the specified selector, or group of selectors.
  - document.getElementsByClassName()
  - document.getElementsByName()
  - document.getElementsByTagName()

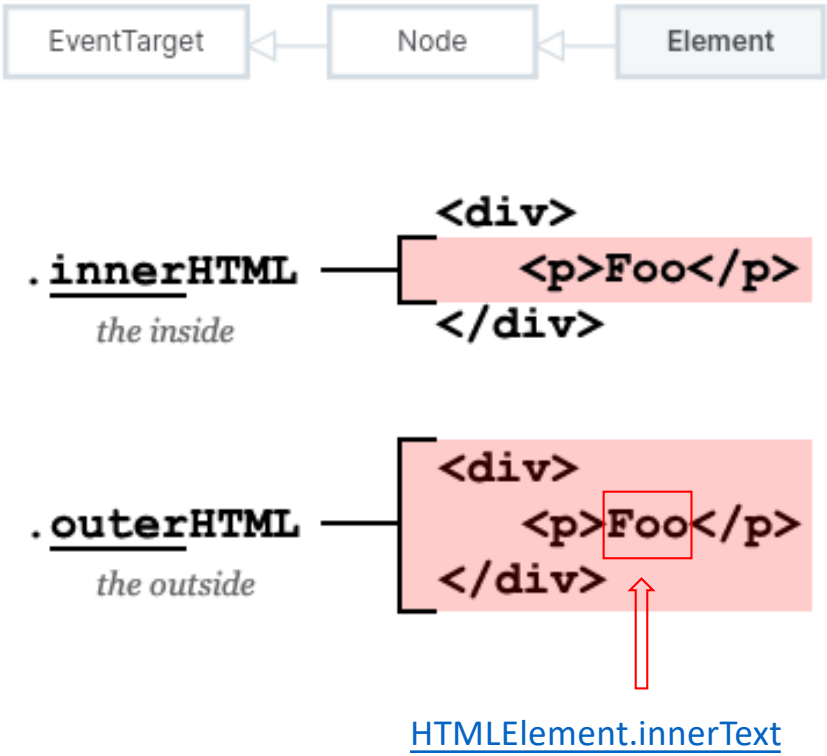

```
createElement(tagName)
createElement(tagName, options)
```

```
getElementById(id)
```

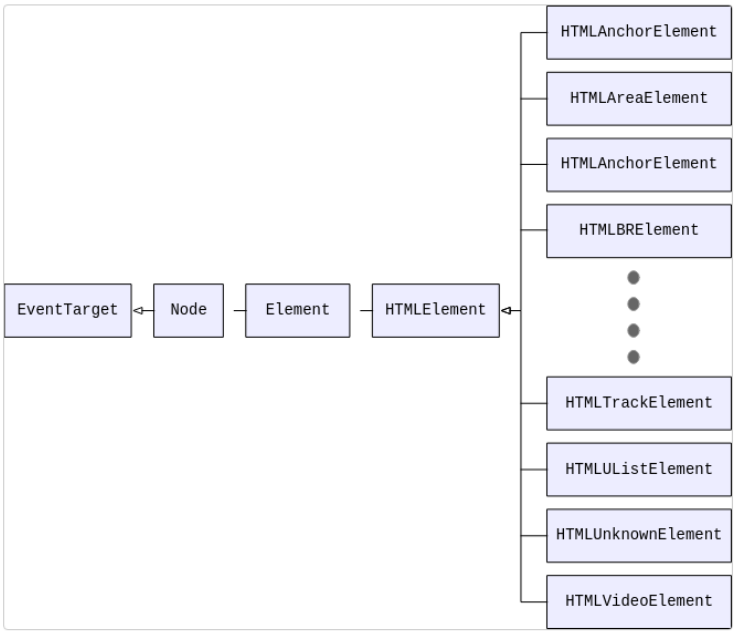
```
var el = document.querySelector(".myclass");
```

The Element Interface

- **Element** is the most general **base class** from which all element objects in a Document inherit.
- It only has **methods** and **properties** common to all kinds of elements.
  - Element.innerHTML: A string representing the markup of the element's content.
  - Element.outerHTML: A string representing the markup of the element including its content. When used as a setter, replaces the element with nodes parsed from the given string.
  - HTMLElement.innerText: This represents the rendered text content of a node and its descendants.



HTML DOM API



- The HTML DOM API adds support for **representing** HTML documents to the **core DOM**.
- It is made up of the **interfaces** that define the **functionality** of each of the elements in HTML.
- The **HTML specification** significantly enhances the **DOM specification** to **add** information specific to using the DOM in the **context** of a **web browser**, as well as to using it to **represent** HTML documents specifically.
- The **Element** interface has been further adapted by introducing the HTMLElement interface, which all more specific HTML element classes inherit from.

HTML Debugging and Validation

- HTML itself doesn't suffer from **syntax errors** because browsers parse it **permissively**, meaning that the page still displays even if there are syntax errors.
- Browsers have **built-in rules** to state how to interpret incorrectly written markup.
- We can use the browser developer tools to look at the rendered markup.
- Markup Validation Service - This webpage takes an HTML document as an input, goes through it, and gives you a report to tell you what is wrong with your HTML.

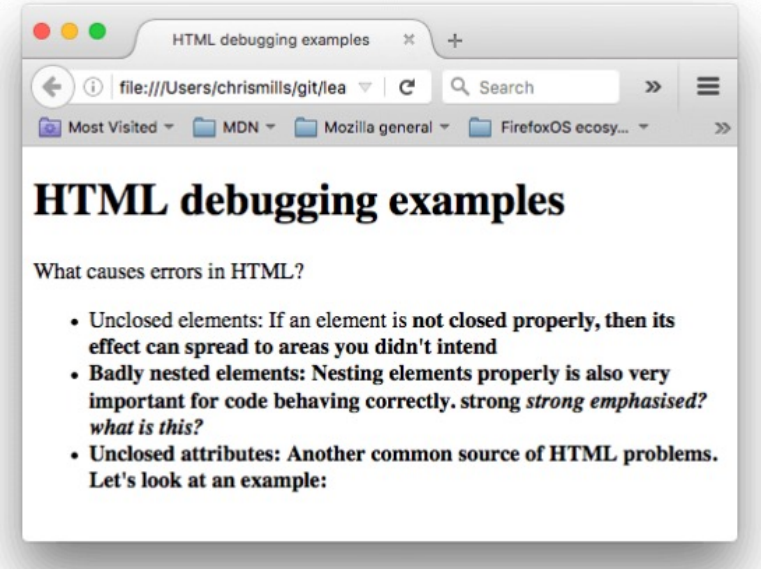
```
<h1>HTML debugging examples</h1>

<p>What causes errors in HTML?

<ul>
  <li>Unclosed elements: If an element is <strong>not closed properly,
      then its effect can spread to areas you didn't intend

  <li>Badly nested elements: Nesting elements properly is also very
important
      for code behaving correctly. <strong>strong <em>strong
emphasized?</strong>
      what is this?</em>

  <li>Unclosed attributes: Another common source of HTML problems.
Let's
      look at an example: <a href="https://www.mozilla.org/>link to
Mozilla
      homepage</a>
</ul>
```



Further Reading

1. Elements reference
2. Attributes reference
3. Preloading content with rel="preload"
4. Script loading strategies
5. Content categories
6. CSS Flow Layout
7. How Whitespace is Handled in HTML
8. Cross-Origin Resource Sharing (CORS)
9. HTML Performance Features
10. Handling Common HTML Problems
11. Accessible Rich Internet Applications (ARIA)
12. Quirks Mode and Standards Mode
13. DOM & BOM
14. How Browsers Work
15. HTML Cheatsheet