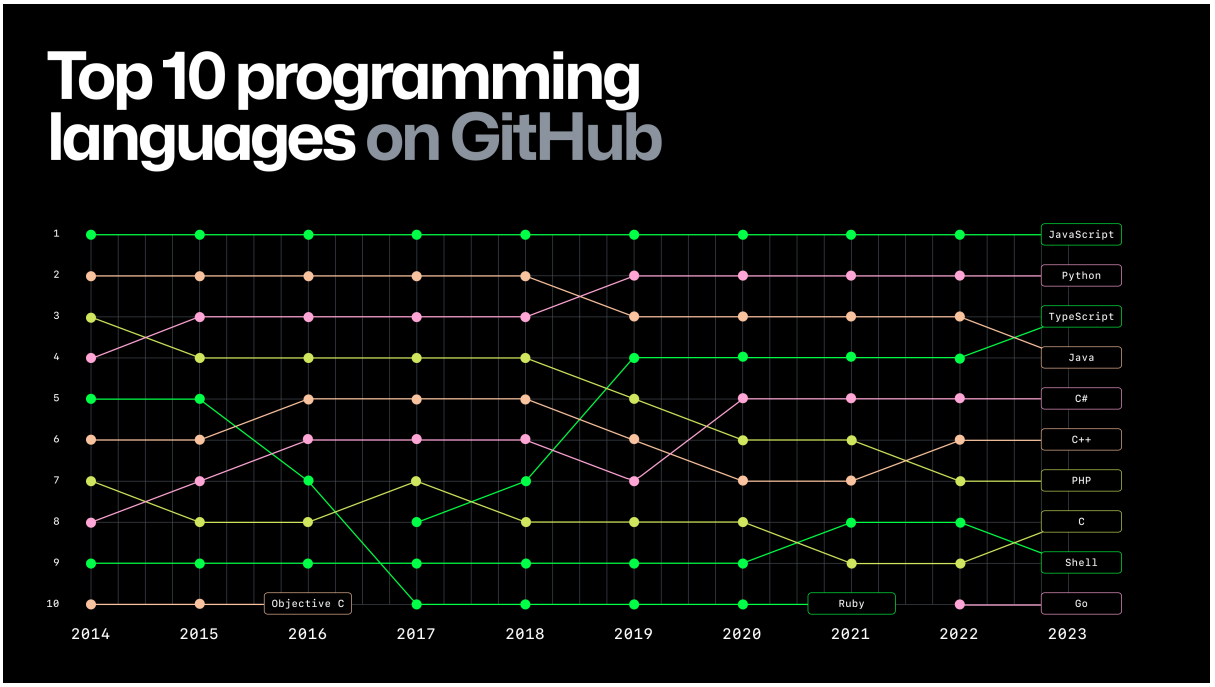
Introduction to HTML5

Baraneetharan Ramasamy

Top Programming, scripting, and markup languages

2023 continues JavaScript's streak as its eleventh year in a row as the most commonly-used programming language. Python has overtaken SQL as the third most commonly-used language, but placing first for those who are not professional developers or learning to code (Other Coders).

> https://survey.stackoverflow.co/2023/#section-most-popular-technologies-programming-scripting-and-markup-languages



> https://github.blog/2023-11-08-the-state-of-open-source-and-ai/#the-most-popular-programming-languages

Web 1.0, Web 2.0, and Web 3.0:

1. **Web 1.0 (Read-Only Web):**
    - **Definition:** Web 1.0 was the earliest version of the Internet. It consisted of static web pages where content creators provided information for users to read.
    - **Characteristics:**
        - Limited interactivity: Users could only consume content.
        - No social features or user-generated content.
        - Examples: Early websites, online brochures, and basic search engines.
    - **Analogy:** Think of it as a digital library where you can read books but not write in them.
2. **Web 2.0 (Participative Social Web):**
    - **Definition:** Web 2.0 introduced user participation and interaction. It's characterized by social media, blogs, wikis, and collaborative platforms.

- **Characteristics:**
    - User-generated content: People could create, share, and comment on content.
    - Social networking: Platforms like Facebook, Twitter, and YouTube emerged.
    - Interactivity: Users actively engaged with websites.
- **Analogy:** Imagine a bustling town square where people share ideas, collaborate, and express themselves.

3. **Web 3.0 (Read, Write, Execute Web):**
    - **Definition:** Web 3.0 is still evolving but aims for a decentralized, intelligent web. It involves semantic understanding, AI, and blockchain technologies.
    - **Characteristics:**
        - Decentralization: Moving away from centralized platforms.
        - AI integration: Smart assistants, personalized recommendations.
        - Semantic web: Machines understand context and meaning.
    - **Analogy:** Picture a self-organizing, interconnected network of smart devices, where data flows seamlessly.


Difference b/w
Web 1.0, Web 2.0, and Web 3.0
Web 1.0
Basic Web Pages
Html
Ecommerce
Java & Javascript
1990-2005
Web 2.0
Social Media
User Generated Content
Mobile Access
Global internet access
Corps Monetizing your data
High-speed communication
High-quality Camera & Video Apps
2006- PRESENT DAY
Web 3.0
Semantic Web
dApps
Users Monetize their data
NFTs
VR & AR (Metaverse)
Permissionless Blockchains
Artificial Intelligence
Interoperability
IMMINENT

> https://enlear.academy/web-1-0-vs-web-2-0-vs-web-3-0-e428cfe09dde

Browser API:

Browser APIs (also known as **Web APIs**) are essential for building web applications and browser extensions. They provide access to various functionalities within web browsers. Here are some key points:

1. **DOM Manipulation:**
   - **Document Object Model (DOM)** APIs allow you to interact with HTML, CSS, and JavaScript in a web page.
   - You can create, modify, and delete elements, change styles, and handle events using DOM methods.
2. **Network Requests:**
   - The **Fetch API** enables making HTTP requests (fetching data) from a server.
   - **XMLHttpRequest** (XHR) is an older API for the same purpose.
3. **Client-Side Storage:**

- **localStorage** and **sessionStorage** allow you to store data on the client side.
- **IndexedDB** provides a more powerful database for structured data storage.

4. **Media and Device Access:**
   - **MediaStream API** lets you access audio and video streams from devices (e.g., webcam, microphone).
   - **Geolocation API** provides location information based on the user's device.
5. **Notifications and Alerts:**
   - **Notification API** allows you to display native notifications to users.
   - **Alerts** can be shown using the `alert()` function (though it's less common).
6. **Other APIs:**
   - **Canvas API** for drawing graphics and animations.
   - **Web Storage API** for managing data (localStorage, sessionStorage).
   - **History API** for manipulating browser history.
   - **Web Workers API** for running background tasks.

Remember, these APIs are built into modern browsers, making it easier for developers to create rich web experiences!

> https://developer.mozilla.org/en-US/docs/Web/API

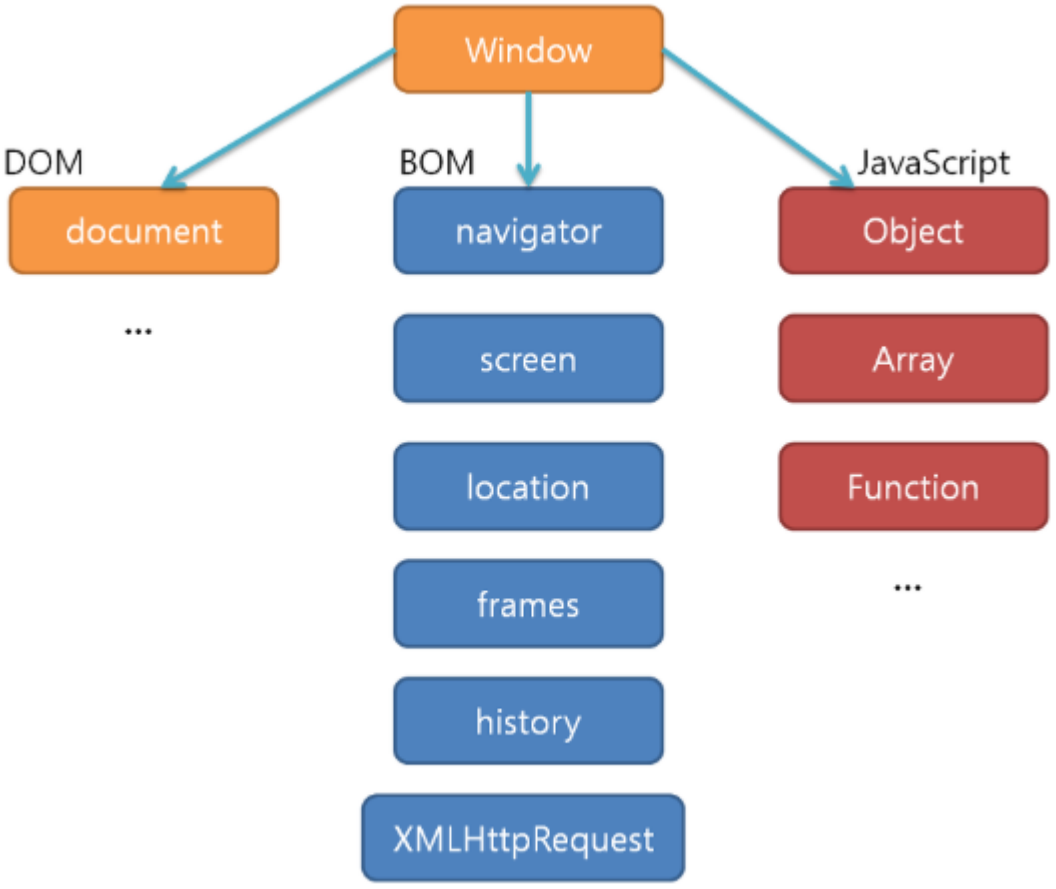
BOM vs DOM:

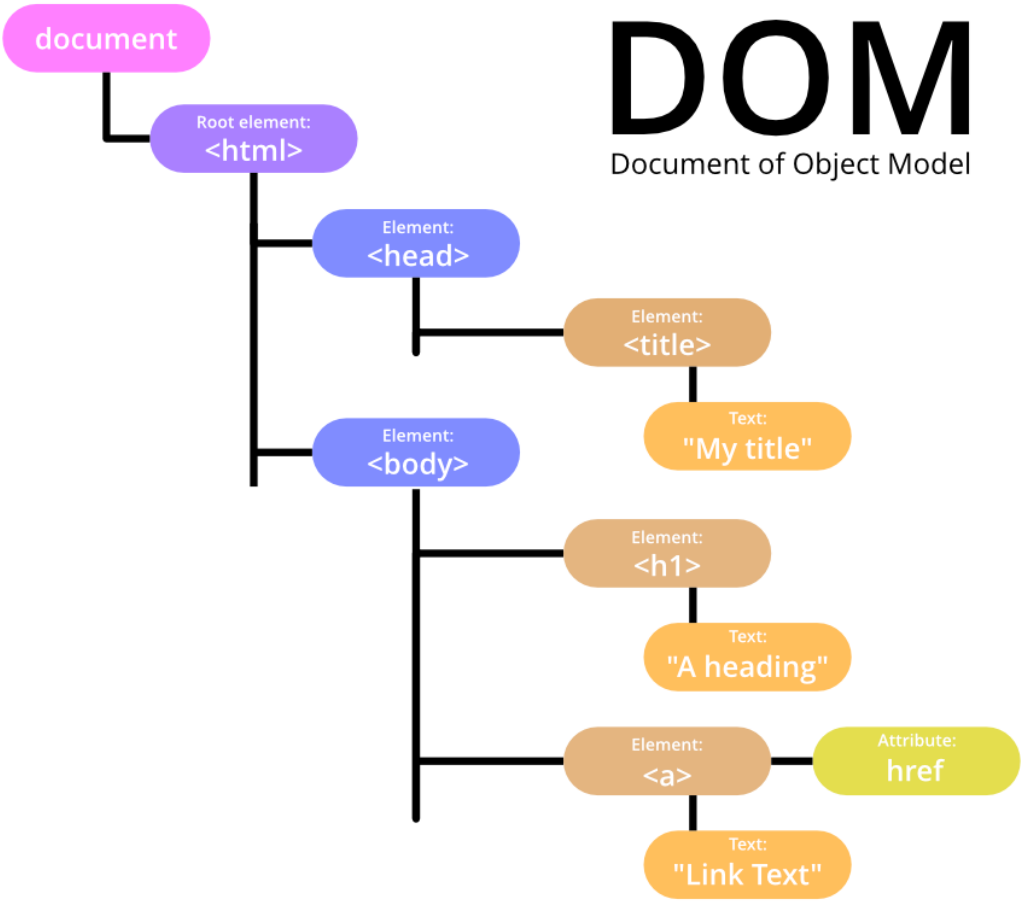
Browser Object Model (BOM) and the **Document Object Model (DOM)**:

1. **BOM (Browser Object Model):**
   - The BOM deals with everything outside the web page content itself.
   - Objects like `navigator`, `history`, `screen`, `location`, and `document` (which is part of BOM) allow JavaScript to interact with the browser.

- BOM provides information about the browser window, screen dimensions, and user's location.
- Example: You can change the browser's URL using `window.location`.

2. **DOM (Document Object Model):**
   - The DOM represents the structure and content of an HTML or XML document.
   - It allows you to manipulate elements (like paragraphs, divs, links) within a web page.
   - DOM methods let you create, modify, or delete elements dynamically.
   - Example: You can change the text of a paragraph using `document.getElementById('myParagraph').textContent = 'New text';`.





In summary, BOM deals with browser-related features, while DOM focuses on document content manipulation.

> https://medium.com/@fknussel/dom-bom-revisited-cf6124e2a816

Rendering Engine vs JavaScript Engine:

1. **Browser Engine (Rendering Engine):**
    - A **browser engine** (also known as a **rendering engine**) is a core software component within a web browser.
    - Its primary responsibility is to load files (such as HTML, CSS, and JavaScript) from a remote server or local disk and display them to users, enabling interaction.

- Different browsers use different engines. For example:
  - **Firefox**: Uses the **Gecko** engine.
  - **Chrome**: Uses **Blink**, which is a fork of **WebKit**.
- These engines convert raw bytes of data into a **Document Object Model (DOM)** object, which represents the structured content of a webpage.

2. **JavaScript Rendering:**
   - When you visit a webpage, the browser handles JavaScript in three steps:
     1. **Parsing**: The JavaScript engine analyzes the code to understand its structure and syntax.
     2. **Compiling**: It converts the script into executable code.
     3. **Execution**: The browser runs the code, resulting in a fully rendered webpage1.
3. **Global Illumination (GI)**:
   - Babylon.js, a powerful WebGL-based graphics engine, now supports basic GI.
   - GI enhances realism by allowing light and shadows to bounce around environments, creating lifelike scenes.
4. **Gaussian Splat Rendering**:
   - Babylon.js introduces Gaussian Splatting, an advanced technique for capturing and displaying volumetric data with unmatched visual fidelity and performance.
5. **Ragdoll Physics**:
   - Babylon.js 7.0 adds support for ragdoll animation, allowing skeletal rigged assets to collapse and move realistically.
6. **WebXR Support**:
   - Babylon.js continues to support WebXR, making it easy to create immersive web experiences.
   - New features include full-screen GUI, touchable UI elements, world scale, and simultaneous use of hands and controllers.

7. **Apple Vision Pro Support**:
    - Babylon.js 7.0 fully supports Apple's Vision Pro, blending real and virtual worlds for Apple fans.
8. **Advanced Animation System Updates**:
    - Babylon.js 7.0 enhances the animation engine, unlocking powerful capabilities for real-time animations on the web.

Browser	Rendering Engine	JavaScript Engine
Chrome	Blink (fork of WebKit)	V8 (JavaScript engine)
Firefox	Gecko	SpiderMonkey
Safari	WebKit	JavaScriptCore
Edge (Chromium)	Blink (fork of WebKit)	V8 (JavaScript engine)
Opera	Blink (fork of WebKit)	V8 (JavaScript engine)

> https://blog.logrocket.com/how-browser-rendering-works-behind-scenes/

Node.js is a cross-platform, open-source JavaScript runtime environment that can run on Windows, Linux, Unix, macOS, and more. Node.js runs on the V8 JavaScript engine, and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting.

History and evolution of HTML:

HTML (Hypertext Markup Language) has come a long way since its inception. Here's a brief overview:

1. **HTML 1.0 (1993)**:
    - Created by Sir Tim Berners-Lee in late 1991.
    - Not officially released until 1995 as **HTML 2.0**.

- Aimed at sharing readable information accessible via web browsers.
- Limited features, but it laid the foundation for web pages.

2. **HTML 2.0 (1995)**:
    - Included all features of HTML 1.0.
    - Became the standard markup language for designing websites.
    - Refinements and improvements over HTML 1.0.
3. **HTML 3.0 (1997)**:
    - Introduced by Dave Raggett.
    - Added powerful features for webmasters.
    - However, some features slowed down browsers.
4. **HTML 4.01 (1999)**:
    - A widely used and successful version.
    - Major update before HTML 5.0.
    - HTML 4.01 provided robust features for web development.
5. **HTML 5 (2012)**:
    - The current standard.
    - Extended version of HTML 4.01.
    - Introduced new elements, APIs, and multimedia support.
    - Enabled rich web applications and responsive design.

> https://en.wikipedia.org/wiki/HTML

Emmet:

Emmet is a powerful coding toolkit that streamlines the creation of extensive HTML code. It allows developers to rapidly generate large HTML blocks using concise shortcuts. Essentially, Emmet integrates CSS selectors with HTML, making coding faster and more efficient.

Here are some key features of Emmet:

1. **Abbreviations**:

- You can use shorthand expressions (abbreviations) that expand into complete HTML or CSS structures.
- For example, `nav>ul>li` expands to:

```
<nav>
  <ul>
    <li></li>
  </ul>
</nav>
```

2. **Multiplication**:
   - Quickly create repetitive elements, such as lists or tables.
   - Example: `ul>li*5` generates an unordered list with 5 list items.
3. **Climb-up**:
   - Move up the DOM hierarchy using `^`.
   - Example: `div+div>p>span+em^bq` creates nested elements.
4. **Custom attributes**:
   - Add attributes like `title`, `href`, or custom ones.
   - Example: `p [title="Hello world"]` results in `<p title="Hello world"></p>`.
5. **Implicit tag names**:
   - Emmet transforms unknown abbreviations into tags.
   - For instance, `foo` becomes `<foo></foo>`.

> https://docs.emmet.io/cheat-sheet/

Key features of HTML5:

HTML5 introduced several key features that revolutionized web development. Here are some notable ones:

1. **Semantic Elements**:
   - HTML5 introduced semantic tags like `<header>`, `<nav>`, `<article>`, and `<footer>`.

- These elements provide better structure and meaning to web content.

2. **Audio and Video Support**:
   - `<audio>` and `<video>` tags allow seamless integration of multimedia content.
   - No need for third-party plugins like Flash.
3. **Canvas API**:
   - The `<canvas>` element enables dynamic graphics and animations.
   - Developers can draw shapes, images, and manipulate pixels directly.
4. **Local Storage (localStorage)**:
   - Allows web applications to store data on the client side.
   - Persistent storage even after the browser is closed.
5. **Geolocation API**:
   - Provides access to user's geographical location.
   - Useful for location-based services and maps.
6. **Web Workers**:
   - Enables background processing without blocking the main UI thread.
   - Ideal for computationally intensive tasks.
7. **Responsive Design**:
   - HTML5 encourages responsive layouts using media queries.
   - Websites adapt to different screen sizes and devices.
8. **Form Enhancements**:
   - New input types like `date`, `email`, and `number`.
   - Validation attributes for better user experience.
9. **Web Storage (sessionStorage)**:
   - Similar to `localStorage`, but data persists only during a session.
   - Useful for temporary data storage.
10. **Drag-and-Drop API**:
    - Simplifies implementing drag-and-drop functionality.

- Enhances user interactions.

> https://www.tutorialrepublic.com/html-tutorial/

JavaScript in web development:

JavaScript plays a crucial role in web development, both on the front-end and back-end. Here's why it's essential:

1. **Front-End Interactivity**:
   - JavaScript enables dynamic content on web pages.
   - It responds to user actions (like clicks) and updates the page without requiring a full reload.
   - For example, interactive forms, real-time chat, and image sliders are all powered by JavaScript.
2. **User Experience Enhancement**:
   - Animations, transitions, and smooth scrolling make websites more engaging.
   - JavaScript allows you to create responsive, user-friendly interfaces.
3. **Client-Side Validation**:
   - Validate user input before submitting forms to the server.
   - Improve data quality and prevent errors.
4. **DOM Manipulation**:
   - JavaScript interacts with the Document Object Model (DOM).
   - You can add, modify, or delete elements on a page dynamically.
5. **Asynchronous Requests**:
   - AJAX (Asynchronous JavaScript and XML) allows fetching data from servers without page reloads.
   - Essential for building modern, responsive web applications.
6. **Frameworks and Libraries**:
   - JavaScript frameworks like React, Angular, and Vue.js simplify complex tasks.

- They enhance productivity and maintainability.

7. **Back-End with Node.js**:
    - Node.js allows server-side JavaScript execution.
    - It’s efficient for handling I/O-intensive tasks, making it a cornerstone technology.

> https://developer.mozilla.org/en-US/docs/Web/API

How a browser loads an HTML page with JavaScript:

1. **DNS Resolution**:
    - The browser first resolves the domain name (like “www.example.com”) to an IP address using DNS (Domain Name System).
    - This allows the browser to know where to send the HTTP request.
2. **TCP/TLS Handshakes**:
    - The browser establishes a TCP (Transmission Control Protocol) connection with the server.
    - If the website uses HTTPS (TLS/SSL), the browser performs a TLS handshake to securely exchange encryption keys.
3. **Fetch Webpage from Server**:
    - The browser sends an HTTP request to the server for the specific webpage (e.g., “index.html”).
    - The server responds with the HTML content of the page.
4. **Browser Parses and Renders HTML Response**:
    - The browser parses the HTML document sequentially, creating a Document Object Model (DOM) tree.
    - For each external resource (images, CSS files, JavaScript files), the browser issues additional HTTP requests.
    - JavaScript files are downloaded and executed in the order they appear in the HTML.

- CSS files are downloaded and applied to style the page.
- Inline JavaScript (within `<script>` tags) is executed as soon as it's encountered.
- The DOM tree is updated based on the HTML and JavaScript execution.
- Images are downloaded and displayed as they are encountered.

> https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work

Event bubbling vs Event capturing:

1. **Event Bubbling:**
   - **Bubbling** occurs when an event starts from the innermost element (e.g., a button) and then propagates outward to outer elements (e.g., parent containers).
   - In other words, the event first gets handled by the innermost element and then bubbles up to its ancestors.
   - Bubbling is typically used when you want to handle events at all levels of the DOM hierarchy.
2. **Event Capturing (Trickling):**
   - **Capturing** is the opposite of bubbling. It starts from the outermost element (e.g., the document or a container) and moves toward the target element (e.g., a button).
   - In this case, the event is first captured by the outermost element and then trickles down to the target.
   - Capturing is typically used when you want to handle events before they reach the target element.
3. **Example:**
   - Imagine a structure with a `<div>` containing a `<ul>` with an `<li>` inside:**HTML**

```
<div>
  <ul>
```

```
    <li>Click me!</li>
  </ul>
</div>
```

- If a click event occurs on the `<li>` element:
  - In **capturing mode**, the event flows from the `<div>` to the `<ul>` and finally to the `<li>`.
  - In **bubbling mode**, the event flows from the `<li>` to the `<ul>` and then to the `<div>`.

> **Capture** goes from outer to inner (trickle down), and **bubble** goes from inner to outer.

JavaScript page lifecycle:

1. **Page Lifecycle API**:
   - The **Page Lifecycle API** brings features commonly found in mobile operating systems to the web. It allows browsers to safely freeze and discard background pages to conserve resources.
   - Developers can handle these interventions without affecting the user experience.
2. **Phases of JavaScript Event**:
   - During the lifecycle of a JavaScript event, there are three phases:
     - **Capturing Phase**: The event descends to the element.
     - **Target Phase**: The event reaches the element.
     - **Bubbling Phase**: The event bubbles up from the element.
   - These phases help you understand how events propagate through the DOM.
3. **Page Lifecycle in JavaScript**:
   - Page lifecycle events occur when a page is loaded. These events are triggered by the browser.

- Common events include:
  - `DOMContentLoaded`: Fired when the initial HTML document has been completely loaded and parsed.
  - `load`: Triggered when all external resources (like images and stylesheets) have finished loading.
  - `beforeunload`: Occurs before the user navigates away from the page.
  - `unload`: Fired when the page is about to be unloaded (e.g., when the user closes the tab or navigates to a different site).

> https://javascript.info/onload-ondomcontentloaded

Event listeners in JavaScript:

Adding an event listener in JavaScript is easy. All you need to do is select the element you want to listen for events on, and then attach an event listener to that element. Here's an example:

```
const button = document.querySelector("#myButton");
button.addEventListener("click", () => {
  console.log("Button clicked!");
});
```

In this example:

- We select the button element with the ID `"myButton"`.
- We attach a `click` event listener to it.
- When the button is clicked, the specified function (in this case, an arrow function) is executed, logging `"Button clicked!"` to the console.

> https://javascript.info/introduction-browser-events

Modifying the document:

1. **Creating Elements**:
   - You can create new elements using `document.createElement(tag)`. For instance:

```
<body>
  <script>
    let div = document.createElement('div');
    div.innerHTML = "Hello, world!";
    document.body.appendChild(div);
  </script>
</body>
```

- This code snippet creates a new `<div>` element with the text “Hello, world!” and appends it to the `<body>`.

2. **Adding Text to Elements**:
   - To add text to an element, use `document.createTextNode(text)`. For example:

```
<body>
  <div id="container"></div>
  <script>
    const div = document.getElementById("container");
    let textNode = document.createTextNode('Here is s
ome text');
    div.appendChild(textNode);
  </script>
</body>
```

   - This adds the text “Here is some text” to the `<div>` with the ID “container” without interpreting it as HTML.

3. **Changing Attributes**:
   - Modify an attribute using `element.setAttribute(name, value)`. Example:

```
<body>
  <div id="firstID"></div>
  <div>new id is: <span id="result"></span></div>
  <script>
    const div = document.getElementById("firstID");
    const result = document.getElementById("result");
    div.setAttribute("id", "newDiv");
    result.innerHTML = div.id;
```

```
  </script>
</body>
```

- This changes the ID of the first `<div>` and displays the updated ID in the `<span>`.

4. **Cloning Elements**:
   - Create a copy of an element using `element.cloneNode(deep)`. Setting `deep` to true clones the element and its descendants:

```
<body>
  <div id="mydiv">one div here, or two divs if clone
d! <span>And here is a span inside the div!</span></d
iv>
  <script>
    const div = document.getElementById("mydiv");
    const clone = div.cloneNode(true);
    document.body.appendChild(clone);
  </script>
</body>
```

   - This clones the entire `<div>` and its contents.

5. **Removing Elements**:
   - To remove an element, use `element.remove()`:

```
<body>
  <div>It's the only thing you see, as the next div i
s removed!</div>
  <div id="mydiv">you don't see me if I'm removed!</d
iv>
  <script>
    const div = document.getElementById("mydiv");
    div.remove();
  </script>
</body>
```

   - The second `<div>` is removed from the DOM.

6. **Practical Example: Building a To-Do List**:
   - Let's create a simple to-do list:

```
<body>
  <script>
    let list = document.createElement('ul');
    document.body.appendChild(list);

    function addItem(text) {
      let item = document.createElement('li');
      item.textContent = text;
      list.appendChild(item);
    }

    addItem('Learn JavaScript');
    addItem('Build a to-do list');
  </script>
</body>
```

   - This dynamically adds items to the list.