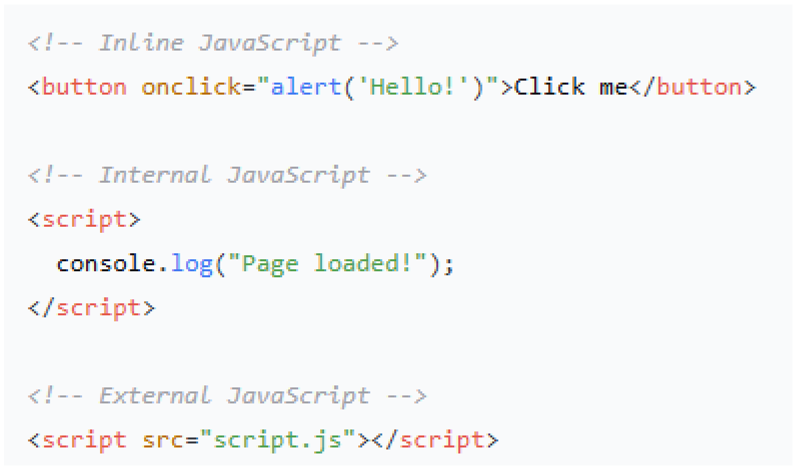
Javascript Fundamentals

JavaScript is a programming language that runs in web browsers, allowing web pages to become interactive, dynamic, and responsive. Unlike HTML (structure) and CSS (styling), JavaScript adds behavior to web pages.

Key Characteristics

- Client-Side Execution: Runs entirely in the user's browser. It enables dynamic content and interactive elements without constant communication with a server.
- Interactivity: Responds to user actions (clicks, typing, scrolling)
- Dynamic Updates: Can modify page content without reloading
- Event-Driven: Reacts to events like clicks, form submissions, etc.
- Versatile: Used for everything from simple animations to complex web applications



```html
<!-- Inline JavaScript -->
<button onclick="alert('Hello!')">Click me</button>

<!-- Internal JavaScript -->
<script>
  console.log("Page loaded!");
</script>

<!-- External JavaScript -->
<script src="script.js"></script>
```

Common Uses

- Form Validation: Check user input before submission
- Dynamic Content: Update page without refreshing
- Animations: Create interactive animations
- API Calls: Fetch data from servers

Simple Example:

```html
<!DOCTYPE html>
<html>
<head>
   <title>JS Demo</title>
</head>
<body>
   <h1 id="greeting">Hello Visitor!</h1>
   <button onclick="changeText()">Change Text</button>
```

```
    <button onclick="showTime()">Show Time</button>

    <script>
      function changeText() {
        document.getElementById("greeting").innerHTML = "Welcome to JavaScript!";
      }

      function showTime() {
        const now = new Date();
        alert("Current time: " + now.toLocaleTimeString());
      }
    </script>
</body>
</html>
```

Variable

Containers for Storing Data. 3 ways: **var, let, const**

var: Declares a function-scoped or globally-scoped variable, optionally initializing it to a value

```
// Function-scoped or globally-scoped
var company = "TechCorp";
var yearFounded = 1995;

// Can be re-declared (can cause bugs)
var score = 100;
var score = 200; // ✅ Allowed - overwrites previous
console.log(score); // 200

// Hoisted to top of function/global scope
console.log(x); // undefined (not an error!)
var x = 10;

// Function scope example
function testVar() {
    if (true) {
        var insideBlock = "I'm accessible";
    }
    console.log(insideBlock); // ✅ Works - var is function scoped
}
testVar();
```

let: Declares a block-scoped local variable, optionally initializing it to a value

```
// Block-scoped
let counter = 0;
counter = 1; // ✅ Reassignment allowed

// Cannot be re-declared in same scope
let total = 100;
// let total = 200; ❌ SyntaxError: Identifier 'total' has already been declared

// But can be redeclared in different blocks
if (true) {
    let total = 200; // ✅ Different scope
    console.log(total); // 200
}
console.log(total); // 100

// Block scope example
function testLet() {
    if (true) {
        let secret = "I'm hidden";
        console.log(secret); // ✅ Works inside block
    }
    // console.log(secret); ❌ ReferenceError: secret is not defined
}
```

const: Same of let. However, the value of const variables can NOT be changed through reassignment and even can NOT be redeclared.

```
// Block-scoped, cannot be reassigned
const MAX_USERS = 1000;
// MAX_USERS = 2000;  TypeError: Assignment to constant variable

// Must be initialized when declared
// const PI;  SyntaxError: Missing initializer in const declaration

// However, objects and arrays declared with const can be modified
const user = { name: "Alice", age: 25 };
user.age = 26; //  Allowed - modifying property
user.city = "NYC"; //  Allowed - adding property
// user = { name: "Bob" };  Not allowed - reassigning the variable

const colors = ["red", "green"];
colors.push("blue"); //  Allowed - modifying array
// colors = ["purple"];  Not allowed - reassigning the variable
```

Data Type

1. Primitive values: string, number, boolean, Special numeric (Infinity, -Infinity, NaN) bigInt, hugeNumber, null etc.
2. Non-Primitive (Reference): Object - Collection of key-value pairs, array, functions, Date, regular expressions

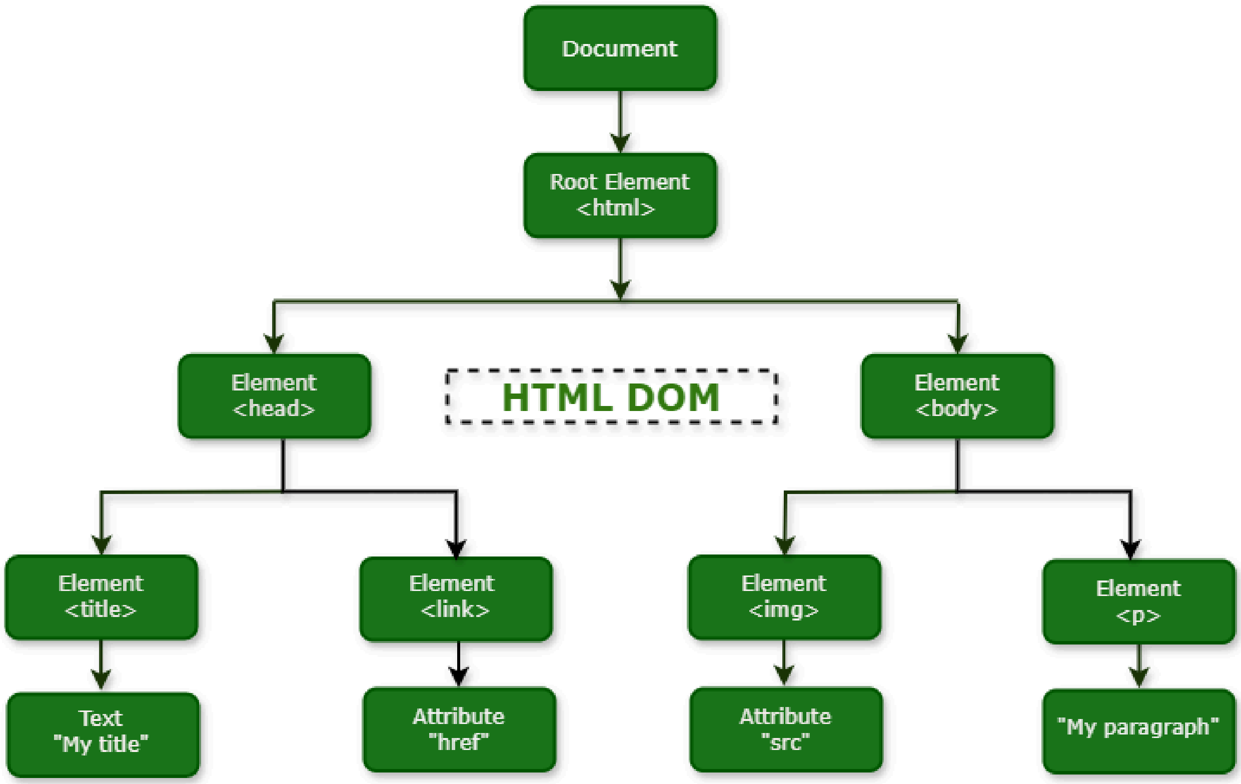
variables and data types.js

Function Call

basic function call syntax.js

DOM

DOM as a tree representation of an HTML page :The DOM (Document Object Model) is the browser's in-memory representation of a web page. Think of it as a hierarchical tree where every piece of the page is a node in that tree. The root is the Document node, then branches for <html>, <head>, <body>, and so on. Each element becomes a node with parent/child and sibling relationships:



This tree model lets scripts and the browser traverse, inspect, modify, add, or remove parts of the page easily.

How browsers interpret HTML into objects (parsing → DOM)

1. **Fetch HTML**: browser receives the HTML document.
2. **Tokenization & Parsing**: the HTML parser reads the markup and converts it into tokens (start tags, end tags, text).
3. **Tree construction**: the parser builds the DOM tree from those tokens. For each start tag it creates an element node and pushes it onto the tree; for text it creates text nodes.
4. **Resource discovery:** as it parses it may find external resources (CSS, JS, images).
    a. CSS files are fetched and parsed into the CSSOM (CSS Object Model).
    b. JavaScript may run during parsing (synchronously if <script> without defer/async), and scripts can read/modify the DOM as it's being built.

5. **DOM + CSSOM** → Render Tree: once you have DOM + CSSOM the browser constructs a render tree (only visible nodes, styled).
6. **Layout & Paint**: browser computes layout (geometry) and paints pixels.

Key point: the DOM is created progressively while the parser runs. Scripts executed during parsing can see and change the DOM immediately — that's why script placement (head vs defer/async vs end of body) matters.

Selecting Elements

`document.getElementById():` Selects a single element by its unique ID.
`document.getElementsByClassName():` Selects multiple elements that share the same class. Returns a live HTMLCollection (NOT an array).
`document.getElementsByTagName():` Selects all elements by tag name (like p, div, li, button).
`document.querySelector():` Selects the first matching element using CSS selectors.
`document.querySelectorAll()`

https://codepen.io/FARZANA-TABASSUM-Lecturer-CSE/pen/qEZLRNP

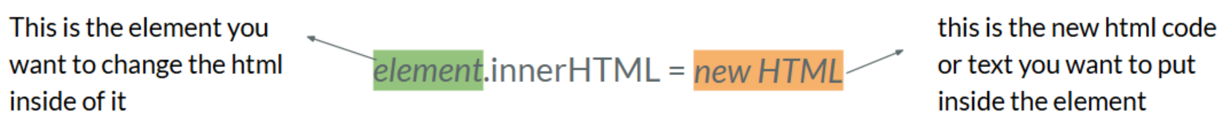
Selecting Children

https://codepen.io/FARZANA-TABASSUM-Lecturer-CSE/pen/qEZLRNP

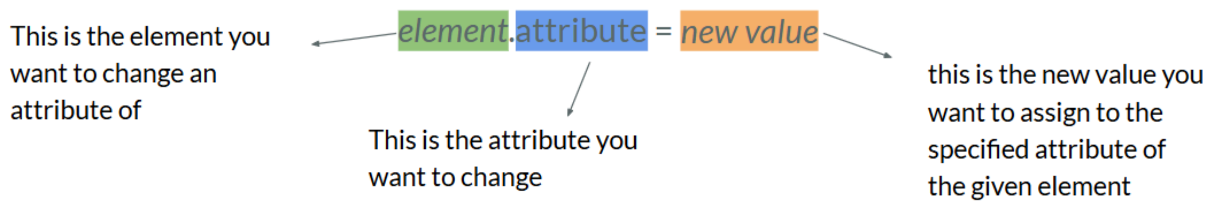
How to read or change DOM content

https://codepen.io/FARZANA-TABASSUM-Lecturer-CSE/pen/XJdopye

Changing HTML Elements


This is the element you want to change the html inside of it
element.innerHTML = new HTML
this is the new html code or text you want to put inside the element

You can also change the value of an HTML attribute.



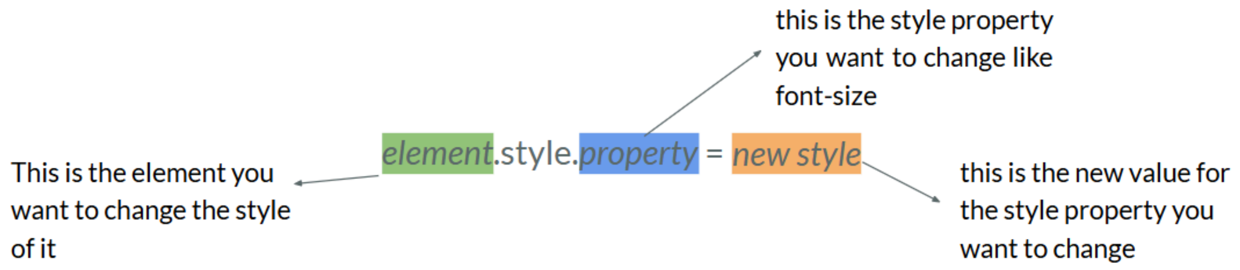
```
element.attribute = new value
```

This is the element you want to change an attribute of

This is the attribute you want to change

this is the new value you want to assign to the specified attribute of the given element

Changing CSS properties



```
element.style.property = new style
```

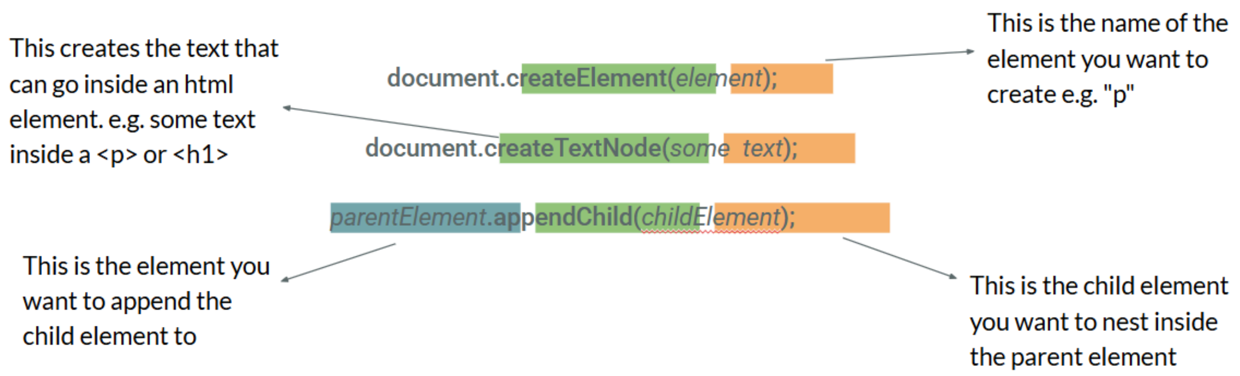
this is the style property you want to change like font-size

This is the element you want to change the style of it

this is the new value for the style property you want to change

As a general rule of thumb, in order to get the style property name in javascript, you should change the CSS property name to camelCase!

Adding HTML Elements



```
document.createElement(element);
document.createTextNode(some text);
parentElement.appendChild(childElement);
```

This creates the text that can go inside an html element. e.g. some text inside a <p> or <h1>

This is the name of the element you want to create e.g. "p"

This is the element you want to append the child element to

This is the child element you want to nest inside the parent element

Removing Existing HTML Elements

```
parentElement.removeChild(childElement)
```

This is the parent element you want to remove one of its children elements

This is the child element you want to remove

Replacing HTML Elements

```
parentElement.replaceChild(newElement, oldElement)
```

This is the parent element you want to replace one of its children elements

This is the new child element you want to add to the parent element by replacing the old one

This is the child element you want to replace