# JavaScript Fundamentals and DOM Events

ISS Lab 8

# Introduction to JavaScript

- JavaScript is a versatile, high-level programming language that powers modern web development.
- It enables dynamic and interactive behavior in web applications, such as form validation, animations, and real-time updates.
- Introduced by Netscape in 1995, it has evolved into one of the most widely used languages today.
- Key Uses:
  - Frontend: Manipulates the DOM (Document Object Model) to create interactive UIs.
  - Backend: Runs on Node.js for server-side applications.
  - Mobile Apps: Frameworks like React Native allow cross-platform mobile app development.
  - Game Development: Libraries like Three.js enable 3D graphics and game creation.

# JavaScript Engines & Execution

- JavaScript runs on engines like V8 (Chrome, Node.js), SpiderMonkey (Firefox), and JavaScriptCore (Safari).
- Code Execution Process:
  1. **Parsing**: The engine reads the code and breaks it into tokens (keywords, variables, etc.).
  2. **Compilation**: Converts tokens into intermediate bytecode using Just-In-Time (JIT) compilation.
  3. **Execution**: The compiled bytecode is executed by the engine.
- JIT Compilation improves performance by compiling frequently used code into machine code.
- JavaScript follows a single-threaded, event-driven model:
  - Single-threaded means only one task can run at a time.
  - Event-driven ensures non-blocking behavior through the event loop and callback queue.

# Variables and Data Types

- Variables are containers for storing data values.
- Declaration Keywords:
  - `var`: Legacy keyword with function scope (avoid in modern code).
  - `let`: Block-scoped variable (preferred for mutable values).
  - `const`: Block-scoped constant (immutable value after declaration).
- Data Types:
  - **Primitive**: Basic types stored directly in memory.
    - `String`, `Number`, `Boolean`, `Undefined`, `Null`, `Symbol`, `BigInt`.
  - **Non-Primitive**: Complex types stored as references.
    - `Objects`, `Arrays`, `Functions`.

# Operators in JavaScript

- Operators perform operations on variables and values.
- Categories:
  - **Arithmetic**: Perform math operations (`+`, `-`, `*`, `/`, `%`).
  - **Comparison**: Compare values (`==`, `===`, `!=`, `!==`, `>`, `<`, `>=`, `<=`).
  - **Logical**: Combine conditions (`&&`, `||`, `!`).
  - **Assignment**: Assign values (`=`, `+=`, `-=`, `*=`, `/=`).
- **Type Coercion vs Strict Equality**:
  - `==`: Compares values after type coercion.
  - `===`: Compares both value and type (strict equality).

```
1 console.log(10 + 5);          // 15
2 console.log(10 == "10");      // true (type coercion)
3 console.log(10 === "10");     // false (strict equality)
```

# Control Flow

- Control flow determines the order in which code is executed.
- Conditional Statements:
  - `if-else`: Executes code based on a condition.
  - `switch`: Matches a value against multiple cases.
- Loops:
  - `for`: Iterates a fixed number of times.
  - `while`: Repeats while a condition is true.
  - `do-while`: Executes at least once before checking the condition.
  - `for...of`: Iterates over iterable objects (arrays, strings).
  - `for...in`: Iterates over object properties.

```
// if-else example
let age = 20;
if (age > 18) {
    console.log("Adult");
} else {
    console.log("Minor");
}

// for loop example
for (let i = 0; i < 5; i++) {
    console.log("Iteration: ", i);
}

// while loop example
let count = 0;
while (count < 3) {
    console.log("Count: ", count);
    count++;
}
```

```
// do-while loop example
let num = 5;
do {
    console.log("Number is: ", num);
    num--;
} while (num > 0);

// for...of loop example
let fruits = ["Apple", "Banana", "Cherry"];
for (let fruit of fruits) {
    console.log("Fruit: ", fruit);
}

// for...in loop example
let person = {name: "Alice", age: 22, city: "New York"};
for (let key in person) {
    console.log(key + ": " + person[key]);
}
```

# Functions in JavaScript

- Functions are reusable blocks of code that perform specific tasks.
- Function Declaration:
  - Defined using the `function` keyword.
  - Can be invoked by its name.
- Arrow Functions (ES6):
  - Shorter syntax with implicit return for single-line functions.
  - Lexical `this` binding.
- Callback Functions:
  - Functions passed as arguments to other functions.
  - Commonly used in asynchronous programming.

```
1  // Function Declaration
2  function greet(name) {
3      return `Hello, ${name}`;
4  }
5
6  // Arrow Function
7  const greet = (name) => `Hello, ${name}`;
8
9  // Callback Example
0  setTimeout(() => console.log("Delayed Message"), 1000);
```

# Objects & Arrays

- **Objects** store key-value pairs and represent structured data.
  - Keys are strings, and values can be any data type.
  - Access properties using dot notation (`object.property`) or bracket notation (`object["property"]`).
- **Arrays** store ordered lists of data.
  - Indexed starting from 0.
  - Provide methods for manipulation (`push`, `pop`, `map`, `filter`, `reduce`).

```javascript
// Object Example
let person = { name: "Alice", age: 22, city: "New York" };
console.log(person.name); // Alice
console.log(person["age"]); // 22
person.job = "Engineer"; // Adding a new property
console.log(person);

// Array Example
let colors = ["red", "blue", "green"];
console.log(colors[0]); // red
colors.push("yellow"); // Add element
console.log(colors);
colors.pop(); // Remove last element
console.log(colors);

// Iterating over an array
colors.forEach(color => console.log(color));

// Using map to transform an array
let upperColors = colors.map(color => color.toUpperCase());
console.log(upperColors);
```

# DOM Manipulation - Basics

- The Document Object Model (DOM) represents the structure of an HTML document as a tree of nodes.
- Selecting Elements:
  - `document.getElementById(id)`: Selects an element by its ID.
  - `document.querySelector(selector)`: Selects the first matching element.
  - `document.querySelectorAll(selector)`: Selects all matching elements as a NodeList.
- Modifying Elements:
  - Change content: `element.innerHTML = "New Content"`.
  - Change styles: `element.style.color = "red"`.

# DOM Manipulation - Examples

```
1  // Selecting and Modifying Elements
2  document.getElementById("demo").innerHTML = "Hello, World!";
3  document.querySelector("h1").style.color = "blue";
4
5  // Creating and Appending Elements
6  let newDiv = document.createElement("div");
7  newDiv.innerHTML = "This is a new div";
8  document.body.appendChild(newDiv);
9
10 // Modifying Attributes
11 let img = document.querySelector("img");
12 img.setAttribute("src", "new-image.jpg");
13 img.setAttribute("alt", "A beautiful landscape");
```

# DOM Manipulation - Examples

```
1  // Adding and Removing Classes
2  let button = document.querySelector("button");
3  button.classList.add("btn-primary");
4  button.classList.remove("btn-disabled");
5
6  // Traversing the DOM
7  let listItem = document.querySelector("li");
8  console.log(listItem.parentElement); // <ul>
9  console.log(listItem.nextElementSibling); // Next <li>
10 console.log(listItem.previousElementSibling); // Previous <
       li>
11
12 // Removing an Element
13 let oldElement = document.getElementById("old");
14 oldElement.remove();
```

# DOM Manipulation - Events

- Events allow interaction with users (e.g., clicks, mouse movements, keypresses).
- Adding Event Listeners:
  - Use `addEventListener` to attach events to elements.
  - Syntax: `element.addEventListener(event, handler)`.
- Common Events:
  - `click`: Triggered when an element is clicked.
  - `mouseover`: Triggered when the mouse hovers over an element.
  - `keydown`: Triggered when a key is pressed.
  - `submit`: Triggered when a form is submitted.

```
// Button Click Event
document.getElementById("btn").addEventListener("click", () => {
    alert("Button Clicked!");
});

// Mouseover Event
document.getElementById("box").addEventListener("mouseover", () => {
    console.log("Mouse hovered over the box");
});

// Keydown Event
document.addEventListener("keydown", (event) => {
    console.log("Key pressed: ", event.key);
});

// Form Submit Event
document.getElementById("myForm").addEventListener("submit", (event) => {
    event.preventDefault();
    console.log("Form submitted!");
});
```

# Best Practices & Debugging

- Follow best practices to write clean and maintainable code:
  - Use `const` and `let` instead of `var`.
  - Prefer strict equality (===) over loose equality (==).
  - Handle errors gracefully using `try-catch`.
- Debugging Tools:
  - Browser DevTools: Inspect elements, log messages, and debug code.
  - Console Methods: `console.log`, `console.error`, `console.warn`.
  - Breakpoints: Pause execution to inspect variables and step through code.

```
try {
    JSON.parse("invalid JSON");
} catch (error) {
    console.error("Error parsing JSON:", error);
}
```

# Conclusion

- JavaScript is essential for modern web development.
- Key Takeaways:
  - Understand the basics (variables, operators, control flow).
  - Master DOM manipulation and event handling for interactive UIs.
- Keep Practicing:
  - Build small projects (to-do apps, calculators).
  - Experiment with frameworks like React, Vue.js, or Angular.
  - Dive deeper into advanced topics (Web APIs, performance optimization).

# Activity

```
try {
    JSON.parse("Activity8.json");
} catch (error) {
    console.error("Brain - 404 Not Found:", error);
}
```