Lab 2 Manual

Basic of JavaScript

Md Sabbir Hossain, Md Ismail Bhuiyan

Summer 2025

Contents

1 JavaScript Overview

JavaScript is a **versatile, interpreted programming language** that powers the dynamic behavior of most websites. It runs inside browsers but also on servers (Node.js).

It is one of the most essential programming languages in the modern development world. Here are some key reasons to learn JavaScript:

- **Web Development:** JavaScript is the primary language used for creating interactive and dynamic websites. It's supported by all modern web browsers, making it essential for web developers.
- **Versatility:** With JavaScript, you can develop both front-end and back-end applications. Using technologies like Node.js, you can build full-stack applications with a single language.
- **Large Ecosystem:** JavaScript has a rich ecosystem of libraries and frameworks, such as React, Angular, and Vue, which simplify and accelerate the development process.
- **Cross-Platform Development:** JavaScript is not just for the web. It can be used to create mobile apps (React Native) and even desktop applications (Electron), making it a powerful tool for developers.
- **In-Demand Skill:** JavaScript is one of the most in-demand programming languages in the job market, with vast opportunities for developers.

2 Including JavaScript in Web Pages

2.1 The `<script>` Tag

JavaScript code can be embedded in HTML using the `<script>` tag.

2.1.1 Inline Script

Write JavaScript directly between script tags:

```
<!DOCTYPE html>
<html>
<head><title>Inline JS Example</title></head>
<body>

<h1>Hello!</h1>

<script>
  alert("Hello from inline script!");
</script>

</body>
</html>
```

Listing 1: Inline JavaScript Example

How it works: The browser executes the JS code immediately during HTML parsing.

2.1.2 External Script

Place JavaScript code in a separate file and link it:

```
<!DOCTYPE html>
<html>
<head><title>External JS</title></head>
<body>

<h1>Hi!</h1>

<script src="app.js"></script>

</body>
</html>
```

Listing 2: External JS file linked

Contents of `app.js`:

```
alert("Hello from external file!");
```

Listing 3: External JavaScript file

2.1.3 Attributes `defer` and `async`

- `defer` — delays execution until HTML is fully parsed. Executes scripts in order.
- `async` — runs script asynchronously, no guaranteed order.

Example:

```
<script src="app.js" defer></script>
```

3 Viewing JavaScript in Browsers

To view and interact with JavaScript in your browser, you can use the browser's built-in Developer Tools. Below is a guide for accessing and using these tools in popular browsers:

3.1 Accessing Developer Tools

- **Google Chrome:** Right-click on the page and select `Inspect`, or press `Ctrl + Shift + I` (Windows/Linux) or `Cmd + Option + I` (Mac).

- **Mozilla Firefox:** Right-click on the page and select `Inspect Element`, or press `Ctrl + Shift + I` (Windows/Linux) or `Cmd + Option + I` (Mac).
- **Microsoft Edge:** Right-click on the page and select `Inspect`, or press `Ctrl + Shift + I` (Windows/Linux) or `Cmd + Option + I` (Mac).
- **Safari:** Right-click on the page and select `Inspect Element`, or press `Cmd + Option + I` (Mac). You may need to enable Developer Tools from `Safari Preferences > Advanced` first.

Once the developer tools are open, navigate to the `Console` tab to see any output, errors, and logs from your JavaScript code.

3.2 Using `console.log()` for Debugging

`console.log()` is a simple but powerful debugging tool that allows you to print messages to the console. It helps you inspect variables, trace the flow of execution, and debug issues in your JavaScript code.

3.2.1 Basic Example

Here's a basic example of using `console.log()` to print values to the console:

```
const name = "John";
console.log(name);  // Output: John
```

Explanation:

- `console.log(name)`: Prints the value of the variable `name` to the browser's console.
- The console will display `John` when this line of code runs.

3.2.2 Debugging Objects and Arrays

`console.log()` can also be used to inspect more complex data structures, like objects and arrays:

```
const person = {
    name: "Alice",
    age: 30
};
console.log(person);  // Output: { name: "Alice", age: 30 }

const numbers = [1, 2, 3, 4];
console.log(numbers);  // Output: [1, 2, 3, 4]
```

Explanation:

- `console.log(person)`: Prints the `person` object to the console.

- `console.log(numbers)`: Prints the `numbers` array to the console.
- These allow you to inspect the structure and values of objects and arrays in real-time.

3.2.3 Useful Console Methods

In addition to `console.log()`, there are several other useful console methods:

- `console.error()`: Prints an error message to the console.
- `console.warn()`: Prints a warning message to the console.
- `console.info()`: Prints an informational message to the console.
- `console.table()`: Displays arrays and objects in a tabular format for better readability.

4 Basic Syntax

4.1 Variables

Declare variables using:

- `var` — function scoped, avoid in modern code
- `let` — block scoped, use this for reassignable variables
- `const` — block scoped, use for constants or variables that won't be reassigned

Example:

```
let message = "Hello, world!";
const PI = 3.1415;
var oldVariable = 42;
```

4.2 Data Types

JavaScript data types include:

- Number (integers and floats)
- String
- Boolean (true/false)
- Null (empty value)
- Undefined (uninitialized variable)
- Object (complex structures)
- Symbol (unique identifiers)

4.3 Dynamic Typing

Variables can hold any data type and can be reassigned with different types:

```
let data = 10;          // number
data = "ten";           // now a string
```

5 Operators

5.1 Arithmetic Operators

```
+, -, *, /, %, ++, --
```

Example:

```
let a = 5;
let b = 2;
console.log(a + b); // 7
console.log(a % b); // 1 (modulus)
```

5.2 Assignment Operators

```
=, +=, -=, *=, /=, %=
```

Example:

```
let x = 10;
x += 5;  // x = 15
```

5.3 Comparison Operators

```
==, ===, !=, !==, >, >=, <, <=
```

Example:

```
console.log(5 == "5");  // true (loose equality)
console.log(5 === "5"); // false (strict equality)
```

5.4 Logical Operators

```
&& (AND), || (OR), ! (NOT)
```

6 Control Flow

6.1 Conditional Statements

```
if (condition) {
  // code if true
} else {
  // code if false
}
```

6.2 Switch

```
switch(expression) {
  case value1:
    // code
    break;
  case value2:
    // code
    break;
  default:
    // code
}
```

6.3 Loops

For Loop:

```
for(let i=0; i<5; i++) {
  console.log(i);
}
```

While Loop:

```
let i = 0;
while(i < 5) {
  console.log(i);
  i++;
}
```

Do-While Loop:

```
let i = 0;
do {
  console.log(i);
  i++;
} while(i < 5);
```

7 Functions

7.1 Function Declaration

```
function greet(name) {
  return "Hello " + name;
}

console.log(greet("Alice"));
```

7.2 Function Expression

```
const greet = function(name) {
  return "Hi " + name;
};

console.log(greet("Bob"));
```

7.3 Arrow Functions

```
const greet = name => "Hey " + name;

console.log(greet("Charlie"));
```

7.4 Parameters and Default Values

```
function multiply(a, b=1) {
  return a * b;
}
```

8 Objects

Objects store key-value pairs.

```
const person = {
  name: "Alice",
  age: 25,
  greet: function() {
    console.log("Hello, " + this.name);
  }
};

person.greet();  // Hello, Alice
```

8.1 Accessing and Modifying Properties

```
console.log(person.name);
person.age = 26;
person.city = "New York";
```

8.2 Object Methods and `this`

```
const person = {
  name: "Bob",
  greet() {
    console.log(`Hi, I am ${this.name}`);
  }
};

person.greet();
```

9 Arrays

Ordered lists.

```
let colors = ["red", "green", "blue"];

console.log(colors[0]);  // "red"
colors.push("yellow");  // add item
colors.pop();           // remove last
```

9.1 Looping Over Arrays

```
colors.forEach(color => console.log(color));
```

10 DOM Manipulation

DOM Manipulation refers to the process of interacting with and modifying the Document Object Model (DOM), which is a representation of the structure of an HTML document.

In simple terms, it allows you to dynamically change the content, structure, or style of a webpage using JavaScript. This is a key part of making web pages interactive.

10.1 Selecting Elements

To manipulate HTML elements, you first need to select them. JavaScript provides several methods to select elements from the DOM:

- `getElementById("id")` — Selects an element by its `id`.
- `getElementsByClassName("class")` — Selects all elements with the given `class`.
- `getElementsByTagName("tag")` — Selects all elements with the given `tag name`.
- `querySelector("selector")` — Selects the first matching element using any CSS selector.
- `querySelectorAll("selector")` — Selects all matching elements using a CSS selector.

Examples

Multiple ways to select elements from the DOM:

- **By ID:** Select a single element with a unique `id`

```
const elementById = document.getElementById("myId");
```

- **By Class Name:** Select multiple elements with the same class

```
const elementsByClass = document.getElementsByClassName("myClass");
// returns an HTMLCollection of matching elements
```

- **By Tag Name:** Select elements by their tag, like `div`, `p`, etc.

```
const elementsByTag = document.getElementsByTagName("p");
// returns an HTMLCollection
```

- **Using CSS Selectors:** More flexible, supports any CSS selector

```
// Select the first matching element by class
const firstElement = document.querySelector(".myClass");

// Select all matching elements by class
const allElements = document.querySelectorAll(".myClass");
// returns a NodeList

// Select the first matching element by ID
const elementById = document.querySelector("#myId");

//IDs should be unique, But querySelectorAll also works if multiple
   elements share the same ID
const allById = document.querySelectorAll("#myId");
```

Differences:

- `getElementById` returns a single element.
- `getElementsByClassName` and `getElementsByTagName` return live HTMLCollections.

- `querySelectorAll` returns a static NodeList.

10.2 Changing Content

Once you've selected an element, you can modify its content using either `textContent` or `innerHTML`.

10.2.1 Changing Text Content

You can change the text inside an element using the `textContent` property.

Example:

```
const element = document.getElementById("myId");
element.textContent = "New Text";
```

Explanation: The text inside the element with ID `myId` is replaced with `"New Text"`.

10.2.2 Changing HTML Content

To modify the HTML structure within an element (including tags like `<b>`, `<p>`, etc.), you can use the `innerHTML` property.

Example:

```
const element = document.getElementById("myId");
element.innerHTML = "<b>Bold Text</b>";
```

Explanation: This changes the content of the element with ID `myId` to include bold text.

10.3 Changing Styles

You can change the visual appearance of elements using the `style` property, which lets you access individual CSS properties.

10.3.1 Changing an Individual Style Property

Example:

```
const element = document.getElementById("myId");
element.style.color = "red";
```

Explanation: This changes the text color of the element with ID `myId` to red.

10.3.2 Changing Multiple Style Properties

You can modify multiple styles at once:

Example:

```
element.style.color = "blue";
element.style.fontSize = "20px";
element.style.fontWeight = "bold";
```

Explanation: This changes the color, font size, and font weight of the element with ID `myId`.

10.4 More DOM Manipulation Commands

- **Change Attributes:**

```
    document.getElementById("elementId").setAttribute("src", "
       newImage.jpg");
    document.getElementById("elementId").setAttribute("href", "
       newURL.com");
```

- **Modify Classes:**

```
    document.getElementById("elementId").classList.add("
       newClass");
    document.getElementById("elementId").classList.remove("
       oldClass");
    document.getElementById("elementId").classList.toggle("
       toggleClass");
```

- **Hide/Show Elements:**

```
    document.getElementById("elementId").style.display = "none"
       ;
    document.getElementById("elementId").style.display = "block
       ";
```

- **Change Value of Form Elements:**

```
    document.getElementById("inputId").value = "New Input Value
       ";
    document.getElementById("selectId").value = "option2";
```

- **Add New Elements:**

```
    const newDiv = document.createElement("div");
    newDiv.textContent = "This is a new div";
    document.body.appendChild(newDiv);
```

- **Remove Elements:**

```
    document.getElementById("elementId").remove();
```

- **Move Elements to New Parent:**

```
const newParent = document.getElementById("newParentId");
const elementToMove = document.getElementById("
   elementToMove");
newParent.appendChild(elementToMove);
```

10.5 Best Practices

When manipulating the DOM:

- Always check if the element exists before changing its content or style.
- Avoid using inline styles directly in HTML and try to manipulate styles via JavaScript only when necessary.
- Be cautious with `innerHTML` when dealing with user inputs to avoid security vulnerabilities like XSS (Cross-Site Scripting).

The DOM provides a powerful way to interact with and modify web pages dynamically. Understanding how to select, change content, and manipulate styles will allow you to create interactive and dynamic user experiences on the web.

11 Events

In JavaScript, **events** are actions or occurrences that happen in the system you are programming for. These events can be triggered by user interactions (such as clicks or keypresses) or system activities (like when a page loads or an element is resized).

Events are typically associated with DOM elements and allow developers to execute code in response to user interactions or other actions. For example, you can set up an event listener to listen for a `click` on a button, and when the event occurs, a function will be executed.

- **Event Listener:** A function that waits for a specific event to occur (e.g., `click`, `keydown`).
- **Event Handler:** The function that runs when the event is triggered.
- **Event Object:** Contains information about the event, such as which element was clicked or which key was pressed.
- **Event Propagation:** Events can propagate through the DOM in two phases: `capturing` (top to bottom) and `bubbling` (bottom to top).
- **Default Action:** Some events, like form submissions, have default actions that can be prevented using `event.preventDefault()`.

11.1 Adding Event Listeners

The following JavaScript code adds an event listener to a button with the ID `myButton`. When the button is clicked, it triggers an alert saying "Button clicked!":

```
const btn = document.getElementById("myButton");
btn.addEventListener("click", function() {
  alert("Button clicked!");
});
```

Explanation:

- `getElementById("myButton")`: Selects the button element by its ID.
- `addEventListener("click", ...)`: Attaches a `click` event listener to the button.
- `alert("Button clicked!")`: Displays a pop-up message when the button is clicked.

11.2 Common JavaScript Event Listeners

- `click`: Fired when an element is clicked.
- `dblclick`: Fired when an element is double-clicked.
- `keydown`: Fired when a key is pressed down.
- `keyup`: Fired when a key is released.
- `mouseover`: Fired when the mouse pointer enters an element.
- `mouseout`: Fired when the mouse pointer leaves an element.
- `focus`: Fired when an element gains focus.
- `blur`: Fired when an element loses focus.
- `submit`: Fired when a form is submitted.
- `change`: Fired when the value of an input, select, or textarea changes.
- `input`: Fired when the value of an input field is changed (real-time).
- `load`: Fired when a resource and its dependent resources have finished loading.
- `resize`: Fired when the window or an element is resized.
- `scroll`: Fired when an element or the window is scrolled.
- `contextmenu`: Fired when the right mouse button is clicked (context menu).
- `touchstart`: Fired when a touch point is placed on the touch surface.
- `touchend`: Fired when a touch point is removed from the touch surface.
- `touchmove`: Fired when a touch point moves across the touch surface.

- `drag`: Fired when an element is being dragged.
- `drop`: Fired when an element is dropped after being dragged.
- `pointerdown`: Fired when a pointer (mouse, pen, or touch) is placed on the surface.
- `pointerup`: Fired when a pointer is removed from the surface.
- `pointermove`: Fired when a pointer moves across the surface.

12 ES6+ Features

ES6+ (ECMAScript 2015 and beyond) introduces modern JavaScript features that enhance the language's functionality, readability, and maintainability. Key features include:

- **Block-scoped Variables:** `let` and `const` replace `var`, providing better control over variable scoping.
- **Arrow Functions:** More concise function syntax that automatically binds `this` from the surrounding context.
- **Template Literals:** A cleaner way to embed expressions inside strings using backticks (`` ` ``) and `${}` for interpolation.
- **Destructuring:** Allows easy unpacking of values from arrays and objects into variables.
- **Spread Operator:** Enables easy copying, merging, or expanding arrays and objects using `...`.
- **Default Parameters:** Function parameters can have default values, making them more flexible.
- **Promises:** A modern way to handle asynchronous code, reducing callback hell.
- **Async/Await:** Makes asynchronous code look synchronous, simplifying handling of promises.
- **Modules:** Supports importing and exporting code between files to improve organization and reusability.

ES6+ features make JavaScript more powerful, readable, and aligned with modern programming practices.

12.1 Let and Const

```
// block scoped variables
let x = 10;
const y = 20; // cannot reassign
```

12.2 Template Literals

```
let name = "Alice";
console.log(`Hello, ${name}!`);
```

12.3 Destructuring

```
const person = {name: "Bob", age: 30};
const {name, age} = person;
console.log(name, age);
```

12.4 Spread Operator

```
const arr1 = [1, 2];
const arr2 = [...arr1, 3, 4];
```

12.5 Default Parameters

```
function greet(name = "Guest") {
  console.log(`Hello, ${name}`);
}
```

13 Error Handling

Error handling in JavaScript is done using `try`, `catch`, and `finally` blocks.

```
try {
  // risky code
  let x = y; // y is undefined
} catch (error) {
  console.error("Error caught:", error);
} finally {
  console.log("Always runs");
}
```

13.1 Explanation

- **try**: Executes code that may throw an error.
- **catch**: Handles the error if one occurs.
- **finally**: Always runs, regardless of success or failure.

Example:

```
try {
  let num = 10;
  let result = num / 0; // Division by zero (Infinity)
  console.log(result);
} catch (error) {
  console.error("Error:", error);
} finally {
  console.log("Always runs");
}
```

14 Some Tips

- **Use `let` and `const` Instead of `var`:**
  - `let` and `const` provide block scope, unlike `var`.
- **Use `const` for Constants:**
  - Use `const` for variables that shouldn't be reassigned.
- **Avoid Global Variables:**
  - Minimize global variables to avoid conflicts.
- **Use Arrow Functions:**
  - Arrow functions offer a concise syntax and proper `this` context.
- **Use `===` for Comparisons:**
  - Always use strict equality to avoid type coercion.
- **Always Handle Errors:**
  - Use `try-catch` or `Promises.catch()` to handle errors.
- **Use `Array.map()` and `Array.filter()`:**
  - Use these methods for cleaner, more readable array manipulations.

15 Websites for Learning JavaScript

- **W3Schools**: A beginner-friendly website with tutorials, examples, and exercises for JavaScript.
- **MDN Web Docs**: Comprehensive and authoritative resource for JavaScript documentation and tutorials.

- **JavaScript.info**: A detailed and organized guide for mastering JavaScript, suitable for beginners to advanced learners.
- **Codecademy**: Interactive platform offering a structured JavaScript course with hands-on exercises.
- **CSS-Tricks**: Not only for CSS but also has great JavaScript tutorials and articles for both beginners and experts.
- **Learn JavaScript Online**: A focused platform offering free lessons, exercises, and tutorials specifically on JavaScript.

16 Mini Projects you can try

Here are some mini projects you can try to enhance your JavaScript skills:

- **To-Do List:** Build a web app that allows users to add, delete, and mark tasks as complete.
- **Calculator:** Create a simple calculator that performs basic arithmetic operations (addition, subtraction, multiplication, division).
- **Form Validator:** Validate user input on a form for correct formats (like email or phone number) and ensure that required fields are filled.