University of Sunderland

**Faculty of Business and Technology: Programme -**

**NAME: Upendra Khanal**

**STUDENT NUMBER: 240704403**

**MODULE CODE: CET138**

**MODULE TITLE: Full Stack Development**

**MODULE LEADER: David Grey**

**SUBMISSION DATE AND TIME: 29 AUG 2025/23:59 GMT**

**ASSIGNMENT:**

**Academic Misconduct is an offence under university regulations, and this involves:**

- **Plagiarism** – where you use information from another information source (including your previously submitted work) and pass it off as your own. This can be through direct copying, poor paraphrasing and/or absence of citations.
- **Collusion** – where you work too closely, intentionally, or unintentionally, with others to produce work that is similar in nature. This can be through loaning of materials, drafts or through unauthorised use of a fellow student's work.
- **Asking another person to write your assignment** – where you ask another individual or company to complete your work for you, be that paid or unpaid, and submit it as if it were your own.
- **Unauthorised use of artificial intelligence** – where you use artificial intelligence tools to generate your assignment instead of completing it yourself and/or where you have not been given permission to use artificial intelligence tools by your module leader. Please complete the following declaration around you use of artificial intelligence tools in your assignment.

**STATEMENT ON USE OF ARTIFICIAL INTELLIGENCE TOOLS:**

| | |
|---|---|
| • I have used artificial intelligence tools to generate an idea for my assignment: | **NO** |
| • I have used artificial intelligence tools to write my assignment for me: | **NO** |
| • I have used artificial intelligence tools to brainstorm ideas for my assignment: | **NO** |
| • I have used artificial intelligence tools to correct my original assignment: | **NO** |

**DECLARATION**

- I understand that by submitting this piece of work I am declaring it to be my own work and in compliance with the university regulations on Academic Integrity.
- I confirm that I have done this work myself without external support or inappropriate use of resources.
- I understand that I am only permitted to use artificial intelligence tools in line with guidance provided by my Module Leader, and I have not used artificial intelligence tools outside this remit.
- I confirm that this piece of work has not been submitted for any other assignment at this or another institution prior to this point in time.
- I can confirm that all sources of information, including quotations, have been acknowledged by citing the source in the text, along with producing a full list of the sources used at the end of the assignment.
- I understand that academic misconduct is an offence and can result in formal disciplinary proceedings.
- I understand that by submitting this assignment, I declare myself fit to be able to complete the assignment and I accept the outcome of the assessment as valid and appropriate.

**ANY OTHER COMMENTS FROM STUDENT:**

# JavaScript

## What is JavaScript?

JavaScript (JS) is a **high-level, interpreted programming language** primarily used to make web pages **dynamic, interactive, and engaging**. While HTML provides the **structure** of a webpage and CSS defines its **style and layout**, JavaScript adds **behavior**—it enables websites to respond instantly to user actions like **clicks, key presses, scrolling, and form submissions** without needing a full page reload.

One of JavaScript's key strengths is its **ability to manipulate the Document Object Model (DOM)** in real-time. This allows developers to **add, remove, or modify HTML elements** dynamically, create animations, validate user input, and update content instantly.

Modern JavaScript supports **event-driven** and **asynchronous** programming, meaning it can perform tasks such as fetching data from a server (using APIs) without freezing the rest of the page. Features like **Promises** and **async/await** help handle these asynchronous operations in a clean, readable way.

JavaScript is standardized as **ECMAScript (ES)**, with major advancements in ES6 and later versions, including:

- let and const for better variable scoping
- Arrow functions for concise syntax
- Template literals for easier string formatting
- Classes and modules for more structured, reusable code

Although JavaScript runs in a **single thread**, it achieves the illusion of multitasking through the **event loop** and **task queues**, enabling smooth, responsive interfaces.

Beyond browsers, JavaScript is also widely used outside the client side. Thanks to **Node.js**, it powers **server-side applications, backend APIs, command-line tools, and even desktop or mobile apps** (with frameworks like Electron and React Native). This versatility has made JavaScript one of the most **popular and in-demand programming languages** in the world today, forming the backbone of full-stack development alongside HTML and CSS.

## Core JavaScript Concepts (Front-End Focus)

1. **Syntax & Types —** Strings, numbers, booleans, null/undefined, objects/arrays; let/const for block scoping; functions as first-class values.
2. **DOM Manipulation —** Read/update the page via document.getElementById/querySelector, .textContent/.value, .classList, and DOM events.
3. **Events —** Run code in response to user actions (click, input, submit, keydown) with addEventListener or inline handlers.

4. **Control Flow & Functions —** Conditionals, loops, pure functions, and higher-order functions (map/filter/reduce) for clean logic.
5. **Asynchronous JS —** Timers (setTimeout), Promises, fetch, async/await to handle non-blocking operations.
6. **Error Handling —** try/catch/finally and guard clauses to keep UI stable when inputs are invalid.
7. **Security Awareness —** Avoid executing untrusted strings; always validate/sanitize before inserting into the DOM.

## Types of JavaScript

- **Inline JavaScript** — Written directly inside an HTML element using the onclick, onchange, or other event attributes. Useful for very small demos, but not recommended for large projects.
- **Internal JavaScript** — Written inside a <script> tag within the <head> or <body> of the HTML document. Suitable for small projects or simple scripts.
- **External JavaScript** — Written in a separate .js file and linked using <script src="script.js">. This is best practice because it separates concerns, improves maintainability, and allows caching.

## JavaScript Example — Hosted on UOS Webspace

Live Example Link:

https://upendra11223.github.io/CET138-A1-ePortfolio/javascript-demo/javascript-demo.html

This demo is a simple four-function calculator. The page has a read-only display and a grid of numeric and operator buttons. All behavior—building numbers, selecting an operator, computing the result, and clearing—is implemented in plain (vanilla) JavaScript. No external libraries are used.

## Detailed Breakdown of JavaScript in the Example

### 1. Calculator State & Display Access

```
let currentInput = '0';
let previousValue = null;
let operator = null;
const display = document.getElementById('display');
```

• **currentInput —** String value shown on screen; starts at '0' and updates as the user types.
• **previousValue —** Stores the first operand once an operator is selected; null when none is stored.
• **operator —** One of '+', '-', '*', '/'; null means no operator chosen yet.
• **display —** A DOM reference to the read-only input element so the UI can be updated as state changes.

**2. Rendering the Display**

```
function updateDisplay() {
  display.value = currentInput;
}
```

• **updateDisplay() —** Centralizes UI updates. Any change to the calculator state calls this function so the display stays in sync.

**3. Building Numbers (appendValue)**

```
function appendValue(digit) {
  if (currentInput === '0') {
    currentInput = digit;
  } else {
    currentInput += digit;
  }
  updateDisplay();
}
```

• **appendValue(digit) —** Concatenates the pressed digit to the current number. Replaces the initial '0' on first input so values do not start with unnecessary leading zeros. Uses JavaScript string concatenation (+=).

**4. Decimal Handling (appendDecimal)**

```
function appendDecimal() {
  if (!currentInput.includes('.')) {
    currentInput += '.';
    updateDisplay();
  }
}
```

• **appendDecimal() —** Prevents users from inserting more than one decimal point in the same number, which would create an invalid value.

**5. Choosing an Operator (setOperator)**

```
function setOperator(op) {
  if (operator && previousValue !== null) {
    calculateResult();
  }
  previousValue = parseFloat(currentInput);
  operator = op;
  currentInput = '0';
  updateDisplay();
}
```

• **setOperator(op) —** Saves the current number as the first operand and sets the operator. If an operation is already pending, it computes that result first (supports chaining like 2 + 3 × 4). Uses the in-built parseFloat() to convert string input to a number.

### 6. Calculating the Result (calculateResult)

```
function calculateResult() {
  if (operator === null || previousValue === null) return;

  const currentValue = parseFloat(currentInput);
  let result;

  switch (operator) {
    case '+': result = previousValue + currentValue; break;
    case '-': result = previousValue - currentValue; break;
    case '*': result = previousValue * currentValue; break;
    case '/':
      result = currentValue === 0 ? 'Error' : (previousValue / currentValue);
      break;
  }

  currentInput = String(result);
  previousValue = null;
  operator = null;
  updateDisplay();
}
```

• **calculateResult() —** Parses the second operand, performs the operation with a switch statement, and writes the outcome back to the display. Division by zero returns 'Error' to avoid Infinity. The result is converted to a string with String(result) for display.

### 7. Clearing State (clearDisplay)

```
function clearDisplay() {
  currentInput = '0';
  previousValue = null;
  operator = null;
  updateDisplay();
}
```

• **clearDisplay() —** Resets all state back to defaults and refreshes the display. This is bound to the C (clear) button.

### Built-in Functions & APIs Used

• **document.getElementById() —** Browser DOM API to fetch elements by ID.
• **parseFloat() —** Converts string input into a number for arithmetic.

- **String() —** Converts computed numeric result back to a string for the display.
- **switch —** Clear multi-branch control flow for the four operators.