JAVASCRIPT FUNDAMENTALS

Module code: SWDJF301

RQF Level: 3

Learning Hours 150

Credits: 15

Sector: ICT AND MULTIMEDIA

Trade: SOFTWARE DEVELOPMENT

Prepared by Jacques MUSENGIMANA

Learning outcome 1: Apply Javascript Basic Concepts

Introduction to JavaScript

1.1 Definition of JavaScript

JavaScript is a lightweight, interpreted programming language that is primarily used to create dynamic and interactive content on websites. It is one of the core technologies of the web, alongside HTML and CSS.

Key points about JavaScript:

• It is a **client-side language**, meaning it runs in the user's browser.
• It can also run on the **server-side** with platforms like Node.js.
• JavaScript enables features like dynamic updates, animations, form validations, and much more.

Features of JavaScript:

• **Platform-independent**: Can run on any device with a web browser.
• **Versatile**: Works for front-end (web pages) and back-end (server-side) development. • **Event-driven**: Responds to user actions like clicks, hovers, and keypresses.

1.2 Application of JavaScript

JavaScript is used in a variety of ways across the tech industry. Below are some of its common applications: **a. Web Development**

• **Front-End Development**: JavaScript is used to make websites interactive. Examples include animations, dropdown menus, and form validations.
• **Back-End Development**: Platforms like Node.js allow JavaScript to run on servers, handling requests and managing databases.

b. Mobile Application Development

• Frameworks like **React Native** and **Ionic** allow developers to build cross-platform mobile apps using JavaScript.

c. Game Development

• Libraries like **Three.js** and **Babylon.js** are used to create 2D and 3D games in browsers. **d. Data Visualization**

• Tools like **D3.js** and **Chart.js** help visualize data using interactive charts and graphs. **e. Internet of Things (IoT)**

• JavaScript can be used with IoT platforms like **Johnny-Five** to control hardware devices. **f. Machine Learning**

• Libraries like **TensorFlow.js** allow developers to implement machine learning models directly in the browser.

1.3 Install VS Code and Node.js

To start coding in JavaScript, you need tools that make development easier and efficient. We'll use **Visual Studio Code (VS Code)** and **Node.js**.

a. Install VS Code

Visual Studio Code is a lightweight, powerful code editor with features like syntax highlighting, extensions, and

integrated terminal.

Steps to install VS Code:

1. Visit the official VS Code website: https://code.visualstudio.com.
2. Download the version suitable for your operating system (Windows, Mac, or Linux).
3. Run the downloaded installer and follow the installation steps.
4. Once installed, open VS Code and explore the interface:
    - **Explorer**: View and manage your files.
    - **Terminal**: Run commands directly in the editor.
    - **Extensions**: Add features like debugging, themes, and more.

b. Install Node.js

Node.js (Node) is an open source, cross-platform runtime environment for executing JavaScript code.

It is essential for server-side programming and for running JavaScript development tools.

Steps to install Node.js:

1. Visit the official Node.js website: https://nodejs.org.
2. Download the **LTS (Long-Term Support)** version for stability.
3. Run the installer and follow the on-screen instructions.
4. Verify installation:
    - Open a terminal (or Command Prompt).
    - Type the following commands:
        - **node -v** to check the Node.js version.
        - **npm -v** to check the Node Package Manager version **(npm is included with Node.js). c.**

Setting Up Your First JavaScript File

1. Open VS Code and create a new folder for your project.
2. Inside the folder, create a file named index.js.
3. Write a simple JavaScript program:

```
console.log("Hello, JavaScript!");
```

4. Open the terminal in VS Code (shortcut: **Ctrl +** `) and run:

```
node index.js
```

You should see Hello, JavaScript! printed in the terminal.

Boom! Congratulation. You have successfully written your first JavaScript code.

1.4 JAVASCRIPT key concepts

1.4.1 Variables

A **variable** is a container used to store data values in JavaScript. It allows you to save information and reuse it later in your program.

NOTE: Variables are classified into Global variables and Local variables based on their scope. The main difference between Global and local variables is that global variables can be accessed globally in the entire program, whereas local variables can be accessed only within the function or block in which they are defined.

JavaScript Identifiers An identifier is a sequence of characters in the code that identifies a variable, function, or property.

An identifier is simply a name. In JavaScript, identifiers are used to name constants, variables, properties, functions, and classes and to provide labels for certain loops in JavaScript code.

Declaring Variables in JavaScript

In JavaScript, variables can be declared using three keywords:

1. **Var**: Older way of declaring variables. Has function scope.
2. **Let**: Introduced in ES6, it has block scope.
3. **Const**: Introduced in ES6, used for declaring constants (unchangeable values).

Example:

```
let name = "Alice"; // Declares a variable and assigns it a value
const PI = 3.14; // Declares a constant variable
var age = 25 // Declares a variable using 'var' (older method)
```

Rules for Naming Variables

- Must start with a letter, underscore (_), or dollar sign ($),
- Variable name doesn't start with a number.

- Cannot be a reserved keyword (e.g., var, function)
- Don’t leave space when naming a variable.

Note and good practice for naming variables:

- Case-sensitive (myVariable and myvariable are different)
- Variable name Should be meaningful and descriptive

Valid Examples:

```
let firstName = "Alice";
let _score = 100;
let $price = 50;
```

Invalid Examples:

```
let 123name = "Bob"; // Cannot start with a number
let var = "test"; // Cannot use reserved keywords
```

Best Practices (Should follow):

- Use **camelCase** for naming: userName, totalAmount, studentScore
- Use **meaningful** names:

  - x, y, z → unclear
  - firstName, phoneNumber → clear and meaningful

Variable Initialization

This means assigning a value to a variable at the time of declaration.

```
let city = "Kigali";
```

```
const country = "Rwanda";
```

```
var number = 5;
```

You can also declare a variable without assigning a value:

```
let studentName; // undefined by default
```

```
studentName = "Grace";
```

Re-declaration of a Variable

This depends on the keyword used:

Keyword	Can be Re-declared?	Can be Re-assigned?
var	Yes	No
let	No (in the same scope)	Yes
const	No	No

Example

```
var x = 10;

var x = 20; // Allowed

let y = 10;

// let y = 20; Error: 'y' has already been declared


const z = 30;

// z = 50; Error: Assignment to constant variable
```

Key Points to Emphasize:

- Use let for variables that change, and const for constants.
- Avoid using var in modern JavaScript unless required for legacy code.
- Variable names must be descriptive and follow camelCase instead of leaving spaces.
- let and const are **block-scoped**, which prevents unwanted bugs.

Variable Scope

• **Global Scope:** A variable declared outside any function is available everywhere. • **Function Scope:** Variables declared with var inside a function are local to that function. • **Block Scope:** Variables declared with let and const are limited to the block {} they are declared in.

```
function test() {
  var x = 10; // Function scope
  if (true) {
    let y = 20; // Block scope
```

```
        console.log(y); // Works
    }
    // console.log(y); // Error: y is not defined
}

console.log(x); // Error: x is not accessible outside the function
```

Hoisting in JavaScript

- **var is hoisted** but initialized as undefined.
- **let and const are hoisted** but not initialized (Temporal Dead Zone until declaration line is found). •

Functions:

- ❖ Declaration → fully hoisted (usable before definition).
- ❖ Expression → behaves like variables (var, let, const).

Example:

```
console.log(a); // Undefined (due to hoisting)
var a = 10;
console.log(b); // ReferenceError: Cannot access 'b' before initialization
let b = 20;
greet(); // works fine
function greet() {
  console.log("Hello!");
}
sayHi(); // ❌TypeError (sayHi function is undefined at this point)
var sayHi = function() {
  console.log("Hi!");
};
```

1.4.2 Data Types

Data types define the kind of value a variable can hold. JavaScript is dynamically typed, meaning a variable can hold any data type and can change its type later.

JavaScript supports several **data types**, which define the type of data a variable holds.

Classification of data types:

- **Primitive Data Types:** are the simplest, immutable data types in JavaScript that store single values.

  Example of primitive data types:

  ❖ **String**: Text data (e.g., "Hello")

  ❖ **Number**: Any numeric value (e.g., 42, 3.14)

  ❖ **Boolean**: true or false

  ❖ **Undefined**: A declared variable without a value

  ❖ **Null**: A variable with an intentionally empty value

  ❖ **Symbol**: Unique and immutable value (used for object property keys)

  ❖ **BigInt**: For very large integers beyond Number limits

```
let name = "John"; // String

let score = 99; // Number

let passed = true; // Boolean

let student; // Undefined

let address = null; // Null

let id = Symbol("id"); // Symbol

let bigNum = 123456789012345n; // BigInt
```

- **Non-Primitive Data Types (Reference types):**

  These are more complex types and can hold multiple values. There are also known as derived data types or reference data types.

  Example of Non-Primitive Data Types:

  ❖ **Object**: Collection of key-value pairs

  ❖ **Array**: Ordered list of items (also an object)

  ❖ **Function**: A block of reusable code (also an object)

  ❖ **Date**: Used to show date and time

  ❖ **RegExp**: Regular expressions

  Example:

```
let person = {name: "Ann"}; // Object

let numbers = [1, 2, 3]; // Array

function greet() { alert("Hi"); } // Function

let today = new Date(); // Date object
```

Type Casting (Type Conversion)

Type casting **means converting a value from one data type to another.**

Two Types:

- **Implicit Type Casting** (automatic by JavaScript)
- **Explicit Type Casting** (manually using built-in functions)

Implicit Type Casting (Type Coercion):

JavaScript automatically converts values during expressions.

```
console.log("5" + 2); // "52" (number 2 converted to string)

console.log("5" - 2); // 3 (string "5" converted to number)

console.log(true + 1); // 2 (true is 1)
```

Explicit Type Casting:

Use built-in functions to convert types intentionally.

Method	Converts To	Example
String(value)	String	String(123) ⟶ "123"
Number(value)	Number	Number("456") ⟶ 456
Boolean(value)	Boolean	Boolean(0) ⟶ false
parseInt()	Integer	parseInt("123px") ⟶ 123
parseFloat()	Float	parseFloat("3.14") ⟶ 3.14

1.4.3 Values

A **value** is the data assigned to a variable. Values can be of any data type (e.g., string, number, boolean, etc.).

Example

```
let city = "Kigali"; // "Kigali" is the value

let score = 90; // 90 is the value
```

1.4.4 Operators

An operator is a symbol that performs an action on one or more values (called operands).

Type	Example	Description
Arithmetic	+, -, *, /, %	Perform math operations
Assignment	=, +=, -=, *=, /=, %=	Assign values to variables
Comparison	==, ===, !=, <, >, <=, >=	Compare values
Logical	&&, ||, !	For Decision making
String	+	Concatenates strings
Unary	++, --	Increment or decrement
Ternary	Condition ? statement: statement	Performing condition
Bitwise	&,|,^,~,<<,>>	Perform colculation on binary level

Assignment Operators

Used to assign values to variables.

Operator	Description	Example
=	**Assign**	**x = 10**
+=	**Add and assign**	**x += 5 ⟶ x = x + 5**
-=	**Subtract and assign**	**x -= 3 ⟶ x = x - 3**
***=**	**Multiply and assign**	**X *= 2**
/=	**Divide and assign**	**X /= 2**
%=	**Modulo and assign**	**X %= 2**

```
let a = 5;

a += 3; // a becomes 8
```

Arithmetic Operators:

Used to perform basic math operations.

Operator	Operation	Example	Result
+	Addition	5+3	8
-	Subtraction	10-4	6
*	Multiplication	4*2	8
/	Division	20/5	4
%	Modulus	10%3	1
**	Exponentiation	2**3	8
++	Increment	X++	X+1
--	Decrement	X--	X-1

```
let num = 6;

num++;

console.log(num); // 7
```

String Operator:

The + operator can also be used to concatenate strings.

```
let firstName = "John";
let lastName = "Doe";

console.log(firstName + " " + lastName); // John Doe
```

Comparison Operators:

Used to compare values. Returns a Boolean (true or false).

Operator	Operation	Example	Result
==	Equal to (type is not checked)	5 == "5"	true
===	Strict equal (type & value)	5 === "5"	false
!=	Not equal	4 != 5	true
!==	Strict not equal	4 != “4”	true

>	Greater than	6 > 3	true
<	Less than	2 < 5	true
>=	Greater than or equal to	3 >= 3	true
<=	Less than or equal to	5 <= 4	false

Logical Operators:

Used to combine multiple conditions (returns Boolean).

Operator	Operation	Example	Result
&&	Logical AND	true && false	false
||	Logical OR	true || false	true
!	Logical NOT	!true	false

Example:

```
let age = 20;

let isStudent = true;


if (age >= 18 && isStudent) {

  console.log("Eligible");

}
```

Bitwise Operators:

Used to perform operations on binary numbers.

Operator	Description	Example
&	AND	5 & 1 → 1
|	OR	OR
^	XOR	5 ^ 1 → 4
~	NO	~5 → -6
<<	Left Shift	5 << 1 → 10
>>	Right Shift	5 >> 1 → 2

Bitwise operators are more advanced and often used in low-level programming or

optimization. Ternary Operator (Conditional Operator)

A **shortcut** for writing an if-else statement. It has **three parts**:

```
condition ? value_if_true : value_if_false;
```

Example:

```
let age = 18;

let result = (age >= 18) ? "Adult" : "Minor";

console.log(result); // Adult
```

Example:

```
let x = 5 + 3; // Arithmetic

let isEqual = x == 8; // Comparison
```

1.4.5 Expressions

An **expression** is any valid unit of code that produces a value.

```
3 + 4 // Expression with value 7
"Hello " + "World" // Expression with value "Hello World"
```

Expressions can be:

- **Arithmetic Expression**: 5 + 3
- **String Expression**: "Hello " + "World"
- **Boolean Expression**: a > b
- **Function Call**: Math.max(5, 10)

```
let sum = 4 + 5; // 9

let greeting = "Hi " + "there"; // "Hi there"

let isEqual = (4 === 4); // true
```

1.4.6 Keywords

Keywords are reserved words in JavaScript that have special meanings. They cannot be used as variable names. **Common JavaScript Keywords:**

Keyword	Use
var, let, const	Variable declaration
if, else	Conditional logic
switch	Performing switch case
for, while, do	Loops
break, continue	Control loop flow
try, catch	Handle exceptions
class, new	Object-oriented features
function	Function declaration
return	Return a value from a function

Example:

```
let age = 20;

if (age >= 18) {
  console.log("Adult");
}
```

1.4.6 Comments

Comments are notes in your code that are ignored by the JavaScript engine. Used for documentation and explanation.

Types:

● **Single-line comment**: Used to comment a single line.

It begins with // and continues to the end of the line.

● **Multi-line comment**: Used to comment multiple lines or a block of text.

Starts with /* and ends with */.

Example:

```
// This is a single-line comment

/*
This is a
multi-line comment
*/
```

1.5 Javascript libraries

● React JavaScript
● JQuery
● Three Javascript

1.5.1 React JavaScript

React is an open-source **JavaScript library** developed by **Facebook** used for **building user interfaces (UIs)**, especially **single-page applications (SPAs)**.

React is focused on the **view layer** (what users see) of an application and allows developers to create reusable **UI components**.

Key Features:

● **Component-Based:** UIs are built using reusable components.
● **JSX Syntax:** JavaScript + HTML-like syntax.
● **Virtual DOM:** Improves performance by updating only changed parts of the page. ● **Unidirectional**

Data Flow: Data flows in one direction (top-down), which helps manage state easily.

Use Cases:

- Web apps (Facebook, Instagram)
- Dashboards
- Admin panels
- SPAs (Single Page Applications)

1.5.2 jQuery

jQuery is a fast, small, and feature-rich **JavaScript library** that simplifies HTML document traversal, event handling, animation, and AJAX.

It was widely used before modern frameworks like React, Vue, and Angular became popular.

Key Features:

- **Cross-browser compatibility**
- **DOM manipulation made easy**
- **Simplified AJAX requests**
- **Animations and effects**
- **Chainable syntax**

Use Cases:

- Adding interactivity to web pages
- Making AJAX calls without refreshing the page
- Animating elements
- DOM manipulation

1.5.3 Three.js (Three JavaScript)

Three.js is a **JavaScript library** that allows developers to create **3D graphics** in the browser using **WebGL** (Web Graphics Library).

It abstracts complex 3D mathematics and graphics code, making it easier to create stunning visual effects and 3D scenes.

Key Features:

- Render 3D objects and animations in a web browser

- Support for lights, cameras, and textures
- Import 3D models (like .glb, .obj)
- Works with WebGL to use GPU for rendering

Use Cases:

- 3D games
- Interactive maps and models
- Scientific visualization
- Virtual tours and augmented reality

1.6 Javascript frameworks

JavaScript frameworks provide **structured and scalable** ways to build complex web applications. Unlike libraries, which offer specific functionality, frameworks often define the full architecture of your application.

Example of JavaScript Framework:

- Vue Javascript
- Angular Javascript
- Express Javascript

1.6.1 Vue.js (Vue JavaScript)

Vue.js is a **progressive JavaScript framework** for building **user interfaces (UIs)** and **single-page applications (SPAs)**. It's known for its **simplicity**, **flexibility**, and easy integration.

Developed by **Evan You** in 2014.
Key Features of Vue JS:

- **Two-way data binding** (like Angular JS)
- **Component-based structure**
- **Reactive data system**
- **Easy integration with existing projects**
- **Lightweight and beginner-friendly**

Use Cases:

- Building SPAs and dynamic interfaces
- Admin dashboards

● Mobile-friendly UIs

1.6.2 Angular (Angular JavaScript)

Angular is a **TypeScript-based framework** developed by **Google** for building **robust web applications**. It is a complete framework with powerful tools for large-scale development.

Note: AngularJS (v1) is outdated; **Angular (v2+)** is now the standard.

Key Features:

● **Two-way data binding**
● **Dependency injection**
● **TypeScript support**
● **Modular architecture**
● **Routing and HTTP built-in**
● **Comprehensive tooling (CLI)**

Use Cases:

● Enterprise-level web applications
● E-commerce platforms
● Government and banking systems

1.6.2 Express.js (Express JavaScript)

Express.js is a **minimal and flexible Node.js web application framework** that provides a robust set of features for **backend development**.
Developed by **TJ Holowaychuk**, it's often used to build **RESTful APIs** and **web**

servers. **Key Features:**

● Middleware system
● Routing support
● Easy integration with databases (MongoDB, MySQL)
● Fast and lightweight
● Works with Node.js

Use Cases:

- Backend servers
- RESTful APIs
- Full-stack apps (with React/Vue frontend)
- Real-time chat apps

1.7 JavaScript Runtime Environment

The JavaScript runtime environment is where JavaScript code is executed outside the browser (like on servers or local machines).

1.7.1 Node.js

Definition:
Node.js is a **JavaScript runtime built on Chrome's V8 engine** that allows developers to run JavaScript on the server side.

Key Features:

- Non-blocking, event-driven I/O
- Excellent for real-time applications (like chat apps)
- Comes with **npm** (Node Package Manager)
- Uses modules to organize code

Example:

```
console.log("Running JavaScript on the server!");
```

Use Case: Server-side scripting, APIs, file systems, CLI tools.

1.7.2 V8 Engine

V8 is the **JavaScript engine** developed by Google and used in Chrome and Node.js to compile and execute JavaScript code.

How It Works:

- Compiles JavaScript to **machine code** for faster execution.
- **Memory management** and **garbage collection** are handled internally.
- Powers both browsers (Chrome) and runtimes (Node.js).

Analogy: Think of V8 as the “engine” under the hood — it drives JavaScript’s performance. **1.7**

JavaScript Versions

JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.

ECMAScript is the official name of the language. ECMAScript versions have been abbreviated to ES1, ES2, ES3, ES5, and ES6. Since 2016, versions are named by year (ECMAScript 2016, 2017, 2018, 2019, 2020).

JavaScript has evolved through a series of standardized versions, known as **ECMAScript (ES)**.

Version	Year	Key Features
ES5	2009	strict mode, JSON support, array methods like .forEach()
ES6 (ES2015)	2015	let, const, arrow functions, classes, template literals, promises, modules
ES7	2016	Array.prototype.includes, ** (exponentiation operator)
ES8	2017	async/await, Object.entries(), Object.values()
ES9+ (2018 – now)	Ongoing	Spread operator, optional chaining, nullish coalescing (??), BigInt, top-level await, etc.

Example from ES6:

```
const greet = (name) => `Hello, ${name}`;

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

1.2 Integration of Javascript to HTML

JavaScript can be integrated into an HTML document to add interactivity, dynamic content, and behaviors. ● Using the <script> Tag

1. JavaScript in <head>

Placing JavaScript in the <head> runs the code before the HTML is fully loaded.

```
<!DOCTYPE html>
```

```
<html>
<head>
 <script>
  alert("This runs before page content is visible.");
 </script>
</head>
<body>
 <h1>Welcome!</h1>
</body>
</html>
```

Caution: This may cause issues if JavaScript interacts with elements that haven't been loaded yet.

2. JavaScript in <body>

Placing JavaScript at the **bottom of the body** ensures that the HTML is fully loaded before the script runs.

```
<!DOCTYPE html>
<html>
<head>
 <title>JavaScript in Body</title>
</head>
<body>
 <h2 id="msg">Hello</h2>
 <script>
  document.getElementById("msg").innerHTML = "Changed by JavaScript!";
 </script>
</body>
```

```
</html>
```

This is the recommended placement for inline scripts that manipulate DOM

elements. ● Using External JavaScript

You can place JavaScript in a **separate file** with the .js extension and reference it using the <script src="..."> tag.

Example:

index.html

```
<!DOCTYPE html>
<html>
<head>
  <script src="main.js"></script>
</head>
<body>
  <h1>External JS Example</h1>
</body>
</html>
```

main.js

```
console.log("This code is loaded from an external JS file.");
```

Benefits:

- Cleaner HTML
- Easier maintenance
- Reusability across multiple pages

Using External JavaScript via CDN

A CDN (Content Delivery Network) hosts popular libraries so you can include them without downloading the

files.

Example using jQuery:

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
```

Usage:

```
<script>
  $(document).ready(function() {
    $("body").css("background-color", "lightyellow");
  });
</script>
```

Benefits:

- Fast loading
- No need to store libraries locally

Caution: Requires internet connection.

JavaScript Output

JavaScript can output data or messages in several ways:

Method	Purpose	Example
alert()	Popup message box	Alert(“Hello world”)
console.log()	Output to browser console	console.log("Debug info");
document.write()	Write directly into HTML	document.write("Page text");
innerHTML	Modify HTML elements	element.innerHTML = "New content";

```
<p id="output"></p>
<script>
  document.getElementById("output").innerHTML = "Output using innerHTML";
  console.log("Output using console.log");
</script>
```

Learning outcome 2: Manipulate data with Javascript

2.1 Using Strings in JavaScript

Strings are used to represent text. They are a sequence of characters enclosed in **single (')**, **double (")**, or **backtick (`)** quotes.

2.1.1 String Declaration

You can declare a string in three main ways:

```
let str1 = "Hello"; // using double quotes

let str2 = 'World'; // using single quotes

let str3 = `Hi there`; // using backticks (template literals)
```

All three are valid, but **template literals** (backticks) allow advanced formatting (covered later).

2.1.2 Escape Characters

Used to insert special characters in a string. Escape characters start with a **backslash** \

Escape	Meaning	Example
\'	Single quote	'It\'s fine'
\"	Double quote	"He said \"Hi\""
\\	Backslash	"C:\\Users\\John"
\n	New line	"Hello\nWorld"
\t	Tab	"Hello\tWorld"

Example:

```
let example = "I\'m learning\nJavaScript!";
console.log(example);
```

2.1.3 String Concatenation

Used to combine (join) two or more strings.

Using + operator:

```
let fname = "John";

let lname = "Doe";

let fullName = fname + " " + lname;

console.log(fullName); // John Doe
```

Using += operator:

```
let text = "Hello";

text += " World";

console.log(text); // Hello World
```

2.1.4 String Methods

JavaScript provides **built-in methods** to work with strings:

Method	Description	Example
length	Returns string length	"Hello".length ⟶ 5
toUpperCase()	Converts to uppercase	"hi".toUpperCase() ⟶ HI
toLowerCase()	Converts to lowercase	"HI".toLowerCase() ⟶ hi
trim()	Removes whitespace	" test ".trim() ⟶ test
slice(start, end)	Extracts part of the string	"Hello".slice(0, 3) ⟶ Hel
replace(a, b)	Replaces a with b	"hi hi".replace("hi", "hey") ⟶ hey hi
charAt(index)	Returns character at given index	"abc".charAt(1) ⟶ b
includes(str)	Checks if string contains substring	"hello".includes("ell") ⟶ true

Example:

```
let message = " JavaScript ";

console.log(message.trim().toUpperCase()); // "JAVASCRIPT"
```

2.1.5 String Search Methods

Used to **search** within a string.

Method	Description	Example
indexOf(str)	Returns the first index of str	"hello".indexOf("l") → 2
lastIndexOf(str)	Last occurrence	"hello".lastIndexOf("l") → 3
startsWith(str)	Checks if string starts with str	"hello".startsWith("he") → true
endsWith(str)	Checks if string ends with str	"hello".endsWith("o") → true
search(regex)	Searches using regex	"abc123".search(/\d/) → 3
match(regex)	Matches regex pattern	"abc123".match(/\d+/) → 123

2.1.6 String Template Literals (Backticks)

Introduced in ES6, **template literals** allow:

- Multiline strings
- String interpolation (embed variables directly)

Example:

```
let name = "Alice";

let age = 25;


let intro = `My name is ${name} and I am ${age} years old.`;

console.log(intro);

// Output: My name is Alice and I am 25 years old.
```

Multiline String Example:

```
let msg = `Hello,

Welcome to JavaScript class!

Let's learn strings.`;

console.log(msg);
```

2.2 Using Conditional Statements in JavaScript

Conditional statements are used to **make decisions** in JavaScript — they control the flow of your program based

on conditions.

2.2.1 if Statement

The if statement checks a condition. If the condition is true, the block of code runs.

```
if (condition) {
  // code to run if condition is true
}
```

Example:

```
let age = 20;

if (age >= 18) {
  console.log("You are eligible to vote.");
}
```

2.2.2 if...else Statement

Executes one block if true, another if false.

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

Example:

```
let age = 16;

if (age >= 18) {
```

```
  console.log("You are eligible to vote.");
} else {
  console.log("You are not eligible to
vote."); }
```

2.2.3 if...else if...else Statement

Used for **multiple conditions**.

```
let score = 85;


if (score >= 90) {
  console.log("Grade: A");
} else if (score >= 70) {
  console.log("Grade: B");
} else if (score >= 50) {
  console.log("Grade: C");
} else {
  console.log("Grade: F");
}
```

2.3 Conditional (Ternary) Operator

A **shorter way** of writing if...else.

```
condition ? value_if_true :
value_if_false;
```
Example:

```
let age = 20;
let result = age >= 18 ? "Adult" :
```

```
"Minor"; console.log(result); // Adult
```

Ternary operator is useful for simple conditions, not for complex logic.

2.3 switch Statement

Use switch when you want to check a variable against **many**

values. Syntax:

```
switch (expression) {
  case value1:
    // code to run
    break;
  case value2:
    // code to run
    break;
  default:
    // code if no match
}
```

Example:

```
let day = 3;


switch (day) {
  case 1:
    console.log("Monday");
    break;
  case 2:
```

```
    console.log("Tuesday");
    break;
  case 3:
    console.log("Wednesday");
    break;
  default:
    console.log("Invalid day");
}
```

✧ **break** stops the code from falling through to the next case.

✧ **default** is optional, runs if no case matches.

2.4 Using Loop Functions in JavaScript

Loops allow you to **repeat a block of code** multiple times — useful when you want to iterate through data like arrays, objects, or numbers.

2.4.1 for loop

Used when you know how many times you want to loop.

Syntax:

```
for (initialization; condition; increment) {
  // code to execute
}
```

Example:

```
for (let i = 1; i <= 5; i++) {
  console.log("Count: " + i);
}
```

Note: Use when you want to loop a fixed number of times.

2.5.2 for...in loop

Used to **iterate over object properties** (keys).

Syntax:

```
for (let key in object) {
  // code
}
```

Example:

```
let user = {name: "John", age: 25};


for (let key in user) {
  console.log(key + ": " + user[key]);
}
```

Note: Use for looping through keys in an

object. **2.5.3 for...of loop**

Used to **iterate over iterable values** like arrays, strings, etc.

Syntax:

```
for (let value of iterable) {
  // code
}
```

Example:

```
let fruits = ["apple", "banana", "mango"];
```

```
for (let fruit of fruits) {
  console.log(fruit);
}
```

Note: Use for looping through elements in an array or characters in a string. **2.5.4 while loop**

Repeats code **as long as** the condition is true.

Syntax:

```
while (condition) {
  // code
}
```

Example:

```
let i = 1;
while (i <= 3) {
  console.log("Number: " + i);
  i++;
}
```

While loop is often used when the number of iterations is not known ahead of time. **2.5.5 do...while loop**

Same as while, but **runs the code at least once**, even if the condition is

false. **Syntax:**

```
do {
  // code
} while (condition);
```

Example:

```
let i = 1;

do {
  console.log("Running: " + i);
  i++;
} while (i <= 2);
```

Use when you want the loop body to execute at least once.

Comparison Table:

Loop Type	Best For	Runs At Least Once?
for	Known number of iterations (arrays, counters)	❌
for...in	Object keys	❌
for...of	Arrays, strings, maps	❌
while	Unknown iterations, condition-based	❌
do...while	When code must run at least once	✅

Practice Example (All Loops):

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


// for loop
for (let i = 0; i < colors.length; i++) {
  console.log("Color: " + colors[i]);
}
```

```
// for...of loop

for (let color of colors) {

  console.log("For-of: " + color);
}



// for...in loop

for (let index in colors) {

  console.log("Index: " + index + ", Value: " + colors[index]);

}
```

2.6.1 Using Functions in JavaScript [Practical]

2.6.1 Function Definition

A **function** is a reusable block of code that performs a

task. **Syntax:**

```
function functionName() {

  // code to run

}
```

Example:

```
function greet() {

  console.log("Hello, World!");

}
```

2.6.2 Function Parameters

Parameters are **placeholders** passed into a function to use as

input. **Example:**

```
function greet(name) {
  console.log("Hello, " + name);
}
greet("Alice"); // Output: Hello, Alice
```

2.6.3 Arrow Functions

A **shorter way** to write functions. Introduced in ES6.

Syntax:

```
const add = (a, b) => a + b;
console.log(add(5, 3)); // Output: 8
```

Example with multiple lines:

```
const greet = (name) => {
  console.log("Hello " + name);
};
```

2.6.4 Built-in Functions

JavaScript provides built-in functions like:

✧ alert()
✧ prompt()
✧ parseInt()
✧ Math.max()

Example:

```
let num = parseInt("50");
alert(Math.max(3, 5, 9)); // Output: 9
```

2.6.5 Function Call

To **execute** a function, you call it using parentheses.

Example:

```
function sayHi() {
  console.log("Hi!");
}

sayHi(); // Calling the function
```

2.6.6 Function apply()

apply() is used to call a function with a **specific** this **value** and arguments as an **array**. **Example:**

```
function introduce(lang1, lang2) {
  console.log(`${this.name} knows ${lang1} and ${lang2}`);
}

let dev = { name: "John" };
introduce.apply(dev, ["JavaScript", "Python"]);
```

2.6.7 Function bind()

bind() **returns a new function** with a bound this value. It doesn't execute the function immediately. **Example:**

```
let user = { name: "Mary" };

function showName() {
  console.log(this.name);
```

```
}


let boundFunc = showName.bind(user);

boundFunc(); // Output: Mary
```

2.6.8 Function Closure

A closure is a function that **remembers variables** from its outer scope even after that scope has finished executing.

Example:

```
function outer() {
  let count = 0;
  return function inner() {
    count++;
    console.log(count);
  };
}


const counter = outer();

counter(); // 1

counter(); // 2
```

Closures are useful for data hiding and state management.

2.6.9 Asynchronous Functions

These run **independently of the main code flow**, allowing your app to stay

responsive. **Example using** setTimeout**:**

```
console.log("Start");
```

```
setTimeout(() => {
  console.log("Running async code...");
}, 2000);
console.log("End");
```

2.6.10 Promise Functions

A Promise represents a value that is **not yet available** (asynchronous result).

Example:

```
let promise = new Promise((resolve, reject) => {
  let success = true;
  if (success) resolve("Task done!");
  else reject("Failed!");
});


promise.then(result => console.log(result))
      .catch(error => console.log(error));
```

2.6.11 Async/Await Function

async/await makes working with promises **easier and cleaner**.

Example:

```
function fetchData() {
  return new Promise(resolve => {
    setTimeout(() => resolve("Data fetched"), 2000);
  });
```

```
}
```

```
async function getData() {

  console.log("Start fetching...");
  let data = await fetchData();

  console.log(data);

}

getData();
```

2.7 Using Objects in JavaScript

2.7.1 Definition

An **object** in JavaScript is a **collection of related data and functions** (called properties and methods). It models **real-world entities**, like a person, car, or student.

Think of an object like a container with labels:

```
let person = {

  name: "Alice",

  age: 25,

  greet: function() {

    return "Hello!";

  }

};
```

2.7.2 Syntax of an Object

Objects are defined using **curly braces** {}, with **key-value pairs** inside.

Example:

```
let car = {
```

```
  brand: "Toyota",
  model: "RAV4",
  year: 2021
};
```

Note: Keys (properties) are always strings, and **values** can be strings, numbers, arrays, functions, or even other objects.

2.7.3 Accessing Object Methods and Properties

You can access object data using:

Dot notation:

```
console.log(car.brand); // Output: Toyota
```

Bracket notation:

```
console.log(car["model"]); // Output: RAV4
```

Calling a method:

```
let user = {
  name: "John",
  greet: function() {
    return "Hello, " + this.name;
  }
};

console.log(user.greet()); // Output: Hello, John
```

2.7.4 Object Constructors

A **constructor function** allows you to create multiple objects with the same

structure. **Syntax:**

```
function Person(name, age) {
 this.name = name;
 this.age = age;
}
let p1 = new Person("Alice", 25);
let p2 = new Person("Bob", 30);
console.log(p1.name); // Output: Alice
```

Note: Use constructor functions to create reusable blueprints for objects. **2.7.5 Object Sets (Set)**

A **Set** is a built-in object that stores **unique values** of any type.

Example:

```
let mySet = new Set();

mySet.add("apple");
mySet.add("banana");
mySet.add("apple"); // Duplicate, won't be added

console.log(mySet); // Output: Set(2) {"apple", "banana"}
console.log(mySet.has("apple")); // true
```

Use cases: Removing duplicates, storing unique items.

2.7.6 Object Maps (Map)

A **Map** is a collection of key-value pairs where **keys can be any type** (not just strings).

Example:

```
let userMap = new Map();
```

```
userMap.set("name", "Alice");
```

```
userMap.set("age", 25);
console.log(userMap.get("name")); // Output: Alice
```

```
console.log(userMap.size); // 2
```

Difference between Object and Map:

Feature	Object	Map
Key types	Strings, Symbols	Any type
Ordered	No	Yes
Iterable	Limited	Fully iterable

2.8 Using Arrays in JavaScript

2.8.1 Syntax

An **array** is a special variable that can hold **multiple values** at once. Arrays are **ordered** collections.

Declaring an array:

```
let fruits = ["apple", "banana", "orange"];
```

Array elements are **indexed starting from 0**.

```
console.log(fruits[0]); // Output: "apple"
```

2.8.2 Types of Arrays

In JavaScript, arrays can store elements of:

❖ **Homogeneous types (same type)**

```
let numbers = [1, 2, 3, 4];
```

❖ **Heterogeneous types** (mixed types)

```
let mixed = ["apple", 3, true, null];
```

Multidimensional arrays (array inside an array)

```
let matrix = [
  [1, 2],
  [3, 4]
];

console.log(matrix[0][1]); // Output: 2
```

2.8.3 Common Array Methods

Method	Description	Example
push()	Adds element to end	fruits.push("grape")
pop()	Removes last element	fruits.pop()
shift()	Removes first element	fruits.shift()
unshift()	Adds element to beginning	fruits.unshift("mango")
length	Returns array length	fruits.length
indexOf()	Finds index of a value	fruits.indexOf("banana")
includes()	Checks if a value exists	fruits.includes("apple")
join()	Joins array elements into a string	fruits.join(", ")
slice()	Extracts part of array	fruits.slice(1, 3)
splice()	Add/remove elements from array	fruits.splice(1, 1, "kiwi")
reverse()	Reverses array order	fruits.reverse()
sort()	Sorts array alphabetically/numerically	numbers.sort((a, b) => a - b)

2.8.4 Array Iterations

JavaScript provides several ways to **loop through arrays**.

❖ **for loop**

```
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
```

```
}
```

for...of

```
for (let fruit of fruits) {
  console.log(fruit);
}
```

❖ **forEach()**

```
fruits.forEach(function(fruit) {
  console.log(fruit);
});
```

map()

Returns a **new array** with modified elements.

```
let upperFruits = fruits.map(fruit => fruit.toUpperCase());
console.log(upperFruits);
```

2.9 Using JavaScript in HTML

2.9.1 HTML Events

HTML events are actions that occur in the browser, such as a user clicking a button or loading a page. **Examples of common events:**

❖ onclick
❖ onmouseover
❖ onkeydown
❖ onload

Example:

```
<button onclick="greet()">Click Me</button>
```

```
<script>
  function greet() {
    alert("Hello, world!");
  }
</script>
```

2.9.2 JavaScript HTML Event Listener

The addEventListener() method attaches an event handler without overwriting existing ones.

Syntax:

```
element.addEventListener(event, function, useCapture);
```

Example:

```
<button id="btn">Click</button>


<script>
  document.getElementById("btn").addEventListener("click", () => {
    alert("Button clicked with Event Listener");
  });
</script>
```

2.9.3 Window Object

The window object is the **global object** in the browser, representing the browser window.

■ **Properties of the Window Object**

console

Used for debugging:

```
console.log("Debug info");
```

document

Represents the HTML document:

```
document.getElementById("demo").innerHTML = "Updated text";
```

innerHeight / innerWidth

Returns the interior height/width of the window in pixels.

```
console.log(window.innerHeight);
```

```
console.log(window.innerWidth);
```

length

Returns the number of frames (used in iframes):

```
console.log(window.length);
```

localStorage

Stores key-value data in the browser:

```
localStorage.setItem("user", "John");
```

```
let user = localStorage.getItem("user");
```

location

Gives URL info and allows redirection:

```
console.log(location.href);
```

```
location.href = "https://www.example.com"; // Redirects
```

Methods of the Window Object

Method	Description	Example
alert()	Displays a pop-up alert	alert("Welcome")

confirm()	Displays a confirm dialog	confirm("Are you sure?")
prompt()	Asks for user input	prompt("Your name?")
open()	Opens a new browser window	window.open("https://...")
close()	Closes the current window	window.close()
print()	Opens the print dialog	window.print()
stop()	Stops page from loading	window.stop()
setTimeout()	Runs a function after a delay	setTimeout(fn, 2000)
clearTimeout()	Cancels timeout set	clearTimeout(id)
setInterval()	Runs a function repeatedly at intervals	setInterval(fn, 1000)
clearInterval()	Cancels interval	clearInterval(id)

2.9.4 JavaScript Form Validation

Used to ensure the user provides valid input before submitting the form.

Example:

```
<form onsubmit="return validateForm()">
  <input type="text" id="name" />
  <input type="submit" value="Submit" />
</form>

<script>
  function validateForm() {
    let name = document.getElementById("name").value;
    if (name === "") {
      alert("Name is required!");
      return false;
    }
```

```
  return true;
}
</script>
```

2.9.5 Apply Canvas

The <canvas> element allows for **drawing graphics** using JavaScript.

Introduction

Canvas is part of HTML5. It creates a blank area where JavaScript can draw graphics.

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

Drawing
Access the 2D drawing context with:

```
let canvas = document.getElementById("myCanvas");
let ctx = canvas.getContext("2d");


// Draw a rectangle
ctx.fillStyle = "red";
ctx.fillRect(10, 10, 100, 50);
```

Coordinates

The canvas uses a **coordinate system** (x, y), where:

❖ (0, 0) is the **top-left corner**.
❖ x moves right, y moves down.

Gradients

Gradients add color blending.

```
let grad = ctx.createLinearGradient(0, 0, 200, 0);
```

```
grad.addColorStop(0, "red");

grad.addColorStop(1, "blue");



ctx.fillStyle = grad;

ctx.fillRect(10, 10, 150, 80);
```

Text in Canvas

Canvas allows you to draw **text** using the fillText() and strokeText() methods.

Basic Syntax:

```
ctx.font = "30px Arial"; // Set font size and family

ctx.fillStyle = "black"; // Set fill color
ctx.fillText("Hello Canvas", 50, 50); // Draw filled text



ctx.strokeStyle = "blue"; // Set stroke color

ctx.strokeText("Outlined Text", 50, 100); // Draw text outline
```

Explanation:

❖ font: Sets the size and font type.

❖ fillText(text, x, y): Draws filled text at position (x, y). ❖

strokeText(text, x, y): Draws an outlined version of the text. ❖

You can also measure the text width with:

```
let metrics = ctx.measureText("Hello");

console.log(metrics.width);
```

Images in Canvas

Canvas can display images using the drawImage() method.

Steps to draw an image:

❖ Load the image

❖ Use drawImage() to place it on the canvas

Syntax:

```
let canvas =
document.getElementById("myCanvas"); let ctx =
canvas.getContext("2d");


let img = new Image();
img.src = "image.jpg"; // or an online link


img.onload = function() {
  ctx.drawImage(img, 0, 0, 200, 150); // (image, x, y, width, height)
};
```

Optional Variants of drawImage():

```
ctx.drawImage(image, x, y); // Draw at position
ctx.drawImage(image, x, y, width, height); // Scale image
ctx.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh); // Crop and draw
```

2.10 JavaScript HTML DOM (Document Object Model)

The DOM is a programming interface for HTML and XML documents. It represents the **page as a tree** structure, where each node is an object representing a part of the document.

innerHTML

Used to get or set the HTML content inside an element.

```
document.getElementById("demo").innerHTML = "Hello, World!";
```

getElementById

Returns the **element with the specified ID**.

```
let element = document.getElementById("header");
```

getElementsByClassName

Returns a **live HTMLCollection** of elements with the specified class name.

```
let items = document.getElementsByClassName("menu");
```

You must loop through it to apply changes:

```
for(let i = 0; i < items.length; i++) {
  items[i].style.color = "red";
}
```

getElementsByName

Returns a **NodeList of elements** with the specified name attribute.

```
let inputs = document.getElementsByName("username");
```

getElementsByTagName

Returns **all elements** with the specified tag name.

```
let paragraphs = document.getElementsByTagName("p");
```

querySelector

Returns the **first matching element** based on a CSS selector.

```
let mainTitle = document.querySelector("h1.title");
```

querySelectorAll

Returns **all matching elements** based on a CSS selector as a static

NodeList. `let allButtons = document.querySelectorAll(".btn");`

JavaScript HTML Styles

JavaScript can be used to manipulate styles dynamically.

Animation (manual)

Animations can be created using JavaScript's setInterval() or requestAnimationFrame().

```
let box = document.getElementById("box");

let pos = 0;

let move = setInterval(() => {
  if (pos >= 350) clearInterval(move);
  else {
    pos++;
    box.style.left = pos + "px";
  }
}, 10);
```

You need to add CSS:

```
#box {
  position: absolute;
  width: 50px;
  height: 50px;
  background: blue;
}
```

Transition (CSS triggered by JavaScript)

❖ Use JavaScript to **toggle a class** that has CSS transitions.

```
<div id="fadeBox" style="width:100px;height:100px;background:red;transition:all
1s;"></div> <button onclick="fade()">Fade Out</button>


<script>
 function fade() {
   document.getElementById("fadeBox").style.opacity = "0";
 }
</script>
```

Slide Show

A slideshow can be created by dynamically changing images or

content. **Example (Basic Image Slideshow):**

```
<img id="slide" src="img1.jpg" width="300">


<script>
 let images = ["img1.jpg", "img2.jpg", "img3.jpg"];
 let index = 0;


   setInterval(() => {
   index = (index + 1) % images.length;
   document.getElementById("slide").src = images[index];
 }, 2000);
</script>
```

2.11 Applying Regular Expression in JavaScript

Regular expressions (RegEx) are patterns used to match character combinations in

strings. They are used for:

❖ Validating input (like email, phone number, etc.)

❖ Searching and replacing text

❖ Parsing data

Syntax

```
let pattern = /pattern/flags;
```

```
let regex = new RegExp("pattern", "flags");
```

2.11.1 Modifiers (also called Flags)

Modifiers are characters added at the end of the pattern to change the matching behavior:

Modifier	Meaning
g	Global match (find all matches)
i	Case-insensitive match
m	Multi-line match

Example:

```
let text = "Hello hello";
let result = text.match(/hello/gi); // Matches both 'Hello' and 'hello'
```

Groups

Groups are used to extract portions of a match or apply quantifiers to part of a pattern.

❖ () - Capturing group

❖ (?:...) - Non-capturing group

Example:

```
let date = "2025-05-30";

let regex = /(\d{4})-(\d{2})-(\d{2})/;

let result = date.match(regex);

// result[1] = 2025, result[2] = 05, result[3] = 30
```

Metacharacters

Metacharacters are symbols with special meaning:

Symbol	Meaning
.	Any character except newline
\d	Any digit (0-9)
\w	Word character (letters, digits, _)
\s	Whitespace
^	Beginning of string
$	End of string
\b	Word boundary
\	Escape character

Example:

```
let str = "abc123";
let regex = /\d+/; // Match one or more digits
```

Quantifiers

Quantifiers specify how many times a character or group must appear:

Symbol	Meaning

*	0 or more times
+	1 or more times
?	0 or 1 time
{n}	Exactly n times
{n,}	At least n times
{n,m}	Between n and m times

Example:

```
let regex = /a{2,4}/;

let text = "aaaabc";

text.match(regex); // Matches 'aaa'
```

Example Use Case

Validate Email:

```
let email = "test@example.com";

let pattern = /^[\w.-]+@[\w.-]+\.\w+$/;

console.log(pattern.test(email)); // true
```

Error Handling in JavaScript

Errors can occur while running scripts, and handling them prevents the script from

crashing. **Types of Errors**

Type	Description
SyntaxError	Mistake in the syntax (e.g., missing parenthesis)

ReferenceError	Accessing an undefined variable
TypeError	Using a value in an inappropriate way
RangeError	Number outside its allowed range
EvalError	Error with eval() function

try & catch

Used to handle runtime errors.

```
try {
  // Code that might throw an error
  let result = x + 5;
} catch (error) {
  console.error("An error occurred: " + error.message);
}
```

throw

Used to create custom errors.

```
function checkAge(age) {
  if (age < 18) {
    throw new Error("You must be at least 18 years old");
  }
  return "Access granted";
}

try {
  console.log(checkAge(15));
} catch (e) {
```

```
  console.error(e.message); // Outputs: You must be at least 18 years old
}
```

try…catch…finally

finally block always runs after try and catch, whether there was an error or not.

```
try {
  console.log("Trying code...");
  throw new Error("Something went wrong!");
} catch (err) {
  console.log("Caught an error:", err.message);
} finally {
  console.log("This always runs.");
}
```

Learning outcome 3: Apply Javascript in Project

Preparing Project Environment

Before writing JavaScript code, it's important to properly set up your project environment. A well-organized folder structure helps maintain clean, scalable, and maintainable code.

1. Create Project Folder

This is the main folder where your whole JavaScript-based web project will

reside. ❖ **Steps:**

1. Choose a location on your computer (e.g., Desktop or Documents).
2. Create a new folder and give it a name, e.g., my-js-project.

```
my-js-project/
```

2. Folders and Files Structuring

Inside the main folder, create subfolders and files that separate your code logically.

Recommended Folder Structure:

```
my-js-project/
├── index.html ← Main HTML file
├── /css ← Stylesheets
│   └── style.css
├── /js ← JavaScript files
│   └── script.js
├── /images ← All images used in the project
│   └── logo.png
├── /assets ← Fonts, icons, or other media
├── /lib ← External libraries (e.g., jQuery, Bootstrap JS)
```

Explanation of Key Components

File/Folder	Purpose
index.html	Entry point of the website. All content starts here.
css/style.css	Holds all CSS styles (colors, fonts, layout, etc.).
js/script.js	Main JavaScript file where your logic is written.
/images	Stores logos, illustrations, or any images.
/lib	Useful for including external libraries manually (if not using CDN).
/assets	Optional: store fonts, icons, or sounds.

Example: index.html File

```
<!DOCTYPE html>
```

```
<html lang="en">

<head>

 <meta charset="UTF-8">
 <title>My JavaScript Project</title>

 <link rel="stylesheet" href="css/style.css">

</head>

<body>

 <h1>Hello, JavaScript!</h1>



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

</body>

</html>
```

Creating Pages with HTML

This part focuses on creating structured and interactive web pages using two key HTML components:

- **Tables**: for displaying data in a structured grid format
- **Forms**: for collecting user input

1. Tables in HTML

- **Definition:**

An HTML table allows you to organize and display data in rows and columns.

- **Basic Structure:**

```
<table border="1">

 <thead>

  <tr>

   <th>Name</th>
```

```
    <th>Age</th>
    <th>Country</th>
  </tr>
</thead>
<tbody>
  <tr>
    <td>John</td>
    <td>25</td>
    <td>Rwanda</td>
  </tr>
  <tr>
    <td>Jane</td>
    <td>22</td>
    <td>Canada</td>
  </tr>
</tbody>
</table>
```

- **Key Tags:**

Tag	Description
<table>	Creates a table
<thead>	Groups the header content
<tbody>	Groups the body content
<tr>	Table row
<th>	Table header cell (bold & centered by default)
<td>	Table data cell

● **Attributes:**

✓ border: Adds border around table (or use CSS for styling)

✓ colspan: Merge columns

✓ rowspan: Merge rows

2. Forms in HTML

● **Definition:**

A form in HTML is used to collect data from users through input fields, buttons, checkboxes, etc.

● **Basic Example:**

```
<form action="submit.php" method="post">
 <label for="name">Name:</label><br>
 <input type="text" id="name" name="name" required><br><br>

 <label for="email">Email:</label><br>
 <input type="email" id="email" name="email"><br><br>

 <label for="gender">Gender:</label><br>
 <input type="radio" name="gender" value="Male"> Male
 <input type="radio" name="gender" value="Female"> Female<br><br>

<label for="country">Country:</label><br>
 <select name="country" id="country">
  <option value="rwanda">Rwanda</option>
  <option value="canada">Canada</option>
 </select><br><br>
```

```
<input type="submit" value="Submit">

</form>
```

Key Input Types:

Input type	Usage
text	Single-line text input
email	Email input with validation
radio	Radio buttons (select one)
checkbox	Multiple selection
password	Hidden characters input
submit	Submit the form
reset	Clear all fields

● **Attributes:**

Attribute	Description
action	Where the form data goes (e.g., backend PHP file or API)
method	GET (data in URL) or POST (data in body)
required	Makes a field mandatory
name	Identifier for form data
id	Identifier for styling or scripting

Applying CSS to HTML Pages

CSS (Cascading Style Sheets) is used to **style and layout web pages** — for example, to change fonts, colors, spacing, and positioning of HTML elements.

1. Inline CSS

● **Definition:**

CSS is written **directly inside an HTML tag** using the style attribute.

● **Example:**

```
<p style="color: blue; font-size: 20px;">This is an inline styled paragraph.</p>
```

● **Use Case:**

✧ Useful for **quick testing** or **small one-time styles**.

✧ **Not recommended** for large projects (hard to maintain).

2. Internal CSS

● **Definition:**
CSS is written **inside the** <style> **tag** in the <head> section of the HTML

document. ● **Example:**

```
<!DOCTYPE html>
<html>
<head>
 <style>
  h1 {
   color: green;
   text-align: center;
  }
 </style>
</head>
<body>
 <h1>This is an internally styled heading</h1>
</body>
</html>
```

● **Use Case:**

✧ Useful for **single-page websites** or when styles only apply to one page.

3. External CSS

● **Definition:**

CSS is written in a **separate** .css **file**, and linked to the HTML file using the <link>

tag. ● **Example:**

style.css

```
body {
  background-color: #f2f2f2;

}


p {

  font-family: Arial;

  color: #333;

}
```

index.html

```
<head>

  <link rel="stylesheet" href="style.css">

</head>
```

● **Use Case:**

✧ **Best practice** for styling large websites.

✧ Promotes **reusability** and **cleaner code**.

4. Imported CSS

● **Definition:**

CSS is imported **inside another CSS file** using

@import. ● **Example:**

```
@import url("main-styles.css");


h2 {
 color: orange;
}
```

● **Use Case:**

✧ Useful when you want to **modularize CSS files**.

✧ Note: @import loads styles **after** the main CSS, which may affect performance.

Comparison Summary:

Method	**Where it's written**	**Scope**	**Recommended**
Inline	Inside HTML tag (style)	Single element	No
Internal	Inside <style> in <head>	Single page	Sometimes
External	Separate .css file	Entire website	Yes
Imported	Inside CSS file with @import	Shared style	With care

Apply JavaScript

JavaScript adds **interactivity and dynamic behavior** to HTML web pages. Below are the core building blocks students must learn to apply JavaScript effectively:

1. Variables

● **What are Variables?**

Variables store **data values** that can be used or changed in the program.

● **Syntax:**

```
let name = "Alice"; // Modern way

var age = 25; // Older way

const pi = 3.14; // Constant (cannot be reassigned)
```

● **Example:**

```
let message = "Hello, World!";

console.log(message); // Output: Hello, World!
```

2. Operators

Operators are used to **perform operations** on values and variables.
● **Types:**

- ✧ **Arithmetic:** +, -, *, /, %
- ✧ **Assignment:** =, +=, -=
- ✧ **Comparison:** ==, ===, !=, <, >
- ✧ **Logical:** &&, ||, !

● **Example:**

```
let a = 10;

let b = 5;

let result = a + b; // result is 15
```

3. Conditional Statements

Used to **execute code only if a condition is**

true. ● **If Statement:**

```
let score = 85;
```

```
if (score >= 50) {
  console.log("Pass");
} else {
  console.log("Fail");
}
```

● **Ternary Operator:**

```
let grade = score >= 50 ? "Pass" :
"Fail";
```

● **Switch Statement:**

```
let color = "blue";
switch (color) {
  case "red":
    console.log("Red selected");
    break;
  case "blue":
    console.log("Blue
    selected"); break;
  default:
      console.log("Color not found");
}
```

4. Looping Statements

Loops help **repeat tasks** efficiently.

● **For Loop:**

```
for (let i = 0; i < 5; i++) {
```

```
  console.log("Number:", 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);
```

5. Functions

Functions are **blocks of code** designed to perform a

task. ● **Function Definition:**

```
function greet(name) {
  console.log("Hello, " + name);
}
greet("Alice"); // Output: Hello, Alice
```

● **Arrow Function:**

```
const greet = (name) => {
```

```
  console.log(`Hello, ${name}`);
};
```

6. Objects

Objects are used to **store related data and functions** together.

- **Definition:**

```
let person = {
  name: "John",
  age: 30,
  greet: function() {
    console.log("Hi, I'm " + this.name);
  }
};
console.log(person.name); // Output: John
person.greet(); // Output: Hi, I'm John
```