INTRODUCTION TO JAVASCRIPT

JavaScript (JS) is a powerful, high-level programming language primarily used to create interactive and dynamic websites. It is supported by all modern web browsers and enables developers to manipulate HTML, CSS, and handle user interactions.

Key Features of JavaScript

- **Lightweight & Flexible** – Runs efficiently in web browsers.
- **Interpreted Language** – No need for compilation.
- **Object-Oriented** – Uses prototypes and objects.
- **Event-Driven** – Responds to user actions like clicks and keystrokes.
- **Cross-Platform** – Works on all devices and operating systems.

---

1. JavaScript Basics

1.1 Where to Write JavaScript?

JavaScript can be included in:

1. **Inline** (inside an HTML tag)
2. **Internal** (inside <script> in HTML)
3. **External** (linked as a separate .js file)

🔹 **1. Inline JavaScript (inside an HTML tag)**

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

<button onclick="alert('Hello from inline JavaScript!')">Click
Me</button>

</body>
</html>
```

✅ *JavaScript is directly written inside the HTML tag's attribute (`onclick`).*

🔹 **2. Internal JavaScript (inside `<script>` tag in the HTML file)**

```
<!DOCTYPE html>
<html>
<head>
  <title>Internal JS Example</title>
  <script>
    function greet() {
      alert("Hello from internal JavaScript!");
    }
  </script>
</head>
<body>

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

</body>
</html>
```

✅ *JavaScript is written inside a `<script>` tag in the HTML document.*

🔹 **3. External JavaScript (linked from a separate `.js` file)**

🔸 **HTML file (index.html)**

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

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

</body>
</html>
```

🔸 **External JavaScript file (script.js)**

```
function greet() {
  alert("Hello from external JavaScript!");
}
```

✅ *JavaScript is placed in a separate `.js` file and linked to the HTML using the `src` attribute.*

1.2 JavaScript Output Methods

JavaScript provides multiple ways to display output:

Method	Description	Example
console.log()	Prints to the browser console	console.log("Hello, Console!");
alert()	Displays a pop-up message	alert("Welcome!");
document.write()	Writes directly to the webpage	document.write("Hello, world!");
innerHTML	Modifies an HTML element	document.getElementById("demo").innerHTML = "New text!";

✅ **console.log()**

What it does:
It **prints messages to the browser's console**, which is helpful for developers to test and debug their code.

Example:

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

🔍 *Think of it like a secret message for you (the developer) to see, not the user. You can open the browser console by pressing F12 and clicking on the "Console" tab.*

✅ **alert()**

What it does:
It **shows a pop-up message box** on the screen to grab the user's attention.

Example:

alert("Welcome!");

🔔 *This is useful for simple messages like greetings or warnings. The user has to click "OK" to continue.*

✅ **document.write()**

What it does:
It **writes directly onto the webpage** when the page is loading.

Example:

document.write("Hello, world!");

📝 *It places text right on the web page, but it's not used much in modern websites because it can replace the whole page content if used after the page loads.*

✅ **innerHTML**

What it does:
It **changes the content inside an HTML element** (like a paragraph or a div).

Example:

document.getElementById("demo").innerHTML = "New text!";

💡 *This means: "Find the element with the ID 'demo' and change what's inside it to say 'New text!'".*

---

2. JavaScript Variables

2.1 Function Scope vs. Block Scope

Function Scope

Function scope means that variables declared using var are accessible only within the function they are declared in. These variables cannot be accessed outside that function.

function exampleFunction() {

```
    var message = "Hello, function scope!";
    console.log(message); // Accessible here
}
exampleFunction();
console.log(message); // ❌ ERROR: message is not defined outside the function
```

Block Scope

Block scope means that variables declared with let and const are limited to the block {} they are defined in.

```
if (true) {
    let blockScopedVar = "I exist inside this block";
    console.log(blockScopedVar); // ✅ Works inside block
}
console.log(blockScopedVar); // ❌ ERROR: blockScopedVar is not defined
```

2.2 JavaScript Data Types

Definition:

Data types specify the kind of values a variable can hold. JavaScript has two major categories: **Primitive** and **Non-Primitive** data types.

Primitive Data Types

These store single values and are immutable (cannot be changed).

Type	Definition	Example
String	Represents textual data	"Hello"
Number	Represents numerical values	25
Boolean	Represents true/false values	true, false
Null	Represents an empty or unknown value	null
Undefined	Represents an uninitialized variable	undefined
Symbol	Represents unique values	Symbol("unique")

Non-Primitive Data Types

These store collections of data or complex structures.

Type	Definition	Example
Object	Stores key-value pairs	{name: "John", age: 25}
Array	Stores ordered lists of values	["Apple", "Banana"]
Function	A block of reusable code	function() {}

---

3. JavaScript Operators

Definition:

Operators perform operations on variables and values.

Arithmetic Operators

Used for mathematical calculations.

```
let a = 10, b = 5;
console.log(a + b); // 15 (Addition)
console.log(a - b); // 5 (Subtraction)
console.log(a * b); // 50 (Multiplication)
console.log(a / b); // 2 (Division)
console.log(a % b); // 0 (Modulus, remainder)
```

Assignment Operators

Used to assign or update values of variables.

```
let x = 10;
x += 5; // Equivalent to x = x + 5
console.log(x); // 15
```

Comparison Operators

Used to compare values and return true or false.

```
console.log(10 == "10"); // true (Loose comparison)
console.log(10 === "10"); // false (Strict comparison)
```

The difference lies in how JavaScript compares the **values** and their **types**.

✅ == (Loose Comparison)

- This operator checks if the **values are equal**, but **ignores the type**.
- JavaScript automatically **converts the types** so it can compare them.

```
console.log(10 == "10"); // true
```

JavaScript converts the string `"10"` into the number `10`, then compares:
`10 == 10` → `true`

✅ === (Strict Comparison)

- This operator checks if the **values AND the types** are the same.
- **No type conversion** is done.

```
console.log(10 === "10"); // false
```

`10` is a number, `"10"` is a string. Since the types are different, it returns `false`.

⚡ Summary:

Expression	Type Conversion	Compares Value	Compares Type	Result
`10 == "10"`	✅ Yes	✅ Yes	❌ No	`true`
`10 === "10"`	❌ No	✅ Yes	✅ Yes	`false`

---

console.log(5 > 3); // true

console.log(5 <= 5); // true

Logical Operators

Used to combine multiple conditions.

console.log(true && false); // false (AND operator)

console.log(true || false); // true (OR operator)

console.log(!true); // false (NOT operator)

✅ `&&` – AND Operator

- Returns `true` **only if both sides are true**.

```
true && true   // ✅ true
true && false  // ❌ false
false && false // ❌ false
false && true  // ❌ false
```

So yes, you're correct:

- `true && true = true`
- `false && false = false`

✅ `||` – OR Operator

- Returns `true` **if at least one side is true**.

```
true || true   // ✅ true
true || false  // ✅ true
false || true  // ✅ true
false || false // ❌ false
```

So you're also right:

- `true || false = true`
- `false || false = false`

✅ `!` – NOT Operator

- Flips the boolean value.

```
!true  // ❌ false
```

`!false // ✅ true`

So:

- `!true = false`
- `!false = true`

🌟 Summary Table

Expression	Result
`true && true`	✅ true
`true && false`	❌ false
`false && false`	❌ false
`true	
`false	
`!true`	❌ false
`!false`	✅ true

---

7. JavaScript Events and Interactive Features

Definition:

JavaScript can be used to handle user interactions and make websites more dynamic. Below are some key interactive features JavaScript enables:

1. Image Slideshow

<img id="slide" src="image1.jpg" width="300">

<button onclick="nextSlide()">Next</button>

<script>

let images = ["image1.jpg", "image2.jpg", "image3.jpg"];

let index = 0;

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

🔹 Line 1:

```
<img id="slide" src="image1.jpg" width="300">
```

- This creates an **image** on the page.
- `id="slide"` gives the image an **identifier** so we can find it later with JavaScript.
- `src="image1.jpg"` tells the browser to display the image file named `image1.jpg`.
- `width="300"` sets the width of the image to **300 pixels**.

🔹 Line 2:

```
<button onclick="nextSlide()">Next</button>
```

- This creates a **button** that says "**Next**".
- When the button is clicked, it calls a JavaScript function called `nextSlide()`.

🔹 Line 3:

```
<script>
```

- This starts a **JavaScript block**.
- Everything between `<script>` and `</script>` is code that runs in the browser.

🔹 Line 4:

```
let images = ["image1.jpg", "image2.jpg", "image3.jpg"];
```

- We are creating a **list (array)** of image filenames.
- The array holds three strings (image paths):
    - `images[0]` is `"image1.jpg"`
    - `images[1]` is `"image2.jpg"`
    - `images[2]` is `"image3.jpg"`

🔹 Line 5:

```
let index = 0;
```

- We create a variable `index` and set it to `0`.
- This means we are starting with the **first image** in the list: `images[0]` → `"image1.jpg"`

🔹 Line 6–8:

```
function nextSlide() {
    index = (index + 1) % images.length;
    document.getElementById("slide").src = images[index];
}
```

🔸 Line 6:

```
function nextSlide() {
```

- This defines a **function** named `nextSlide()` that will run when the button is clicked.

🔸 Line 7:

```
index = (index + 1) % images.length;
```

- This increases the `index` by 1.
- `images.length` is 3 (because we have 3 images).
- The `%` is the **modulo operator**. It keeps the index within 0 to 2.
    - If `index` goes beyond 2, this line **resets it to 0**.
    - So:
        - 0 + 1 = 1 → `images[1]`
        - 1 + 1 = 2 → `images[2]`
        - 2 + 1 = 3 → `3 % 3 = 0` → `images[0]`

🔸 Line 8:

```
document.getElementById("slide").src = images[index];
```

- This finds the image by its `id` ("slide") and changes its `src` (source).
- That updates the picture on the screen to the **next image** in the array.

🟢 Final Behavior:

- When the page loads, it shows `"image1.jpg"`.
- Every time you click the **Next** button:
  - It changes the image to the next one in the array.
  - When it reaches the last image, it goes back to the first.

---

2. Playing Videos

```
<video id="video" width="400" controls>
  <source src="video.mp4" type="video/mp4">
</video>
<button onclick="document.getElementById('video').play()">Play</button>
<button onclick="document.getElementById('video').pause()">Pause</button>
```

3. Form Validation

```
<form onsubmit="return validateForm()">
  <input id="name" type="text" placeholder="Enter your name">
  <button type="submit">Submit</button>
</form>
<script>
  function validateForm() {
    let name = document.getElementById("name").value;
    if (name === "") {
      alert("Name cannot be empty");
      return false;
    }
    return true;
  }
</script>
```

🔹 2. Playing Videos

HTML:

```
<video id="video" width="400" controls>
  <source src="video.mp4" type="video/mp4">
</video>
```

🔍 **Explanation**:

- <video> is used to embed a video on the web page.
- id="video" allows us to access this video in JavaScript.
- width="400" sets the width of the video to 400 pixels.
- controls adds default browser video controls (play, pause, volume, etc.).
- Inside it, the <source> element:
  - src="video.mp4" tells the browser which file to play.

- type="video/mp4" tells the browser the type of the file.

Buttons:

<button onclick="document.getElementById('video').play()">Play</button>

<button onclick="document.getElementById('video').pause()">Pause</button>

🔍 **Explanation**:

- The **Play** button:
  - When clicked, it runs this line:
  - document.getElementById('video').play()
  - This tells the browser: Find the element with ID "video" and **play it**.
- The **Pause** button:
  - Runs this line:
  - document.getElementById('video').pause()
  - This tells the browser to **pause the video**.

✅ **So it works like this:**

- You see a video player and two buttons.
- If you click "Play" — video starts.
- If you click "Pause" — video stops temporarily.

🔹 **3. Form Validation**

HTML Form:

<form onsubmit="return validateForm()">

  <input id="name" type="text" placeholder="Enter your name">

  <button type="submit">Submit</button>

</form>

🔍 **Explanation**:

- This is a simple **form** with a single input.

- The input has:
    - id="name" so JavaScript can find it.
    - type="text" makes it a text box.
    - placeholder="Enter your name" shows light gray text as a hint.
- The form uses onsubmit="return validateForm()":
    - When the user clicks **Submit**, the validateForm() function is called.

JavaScript:

```
<script>
  function validateForm() {
    let name = document.getElementById("name").value;
    if (name === "") {
      alert("Name cannot be empty");
      return false;
    }
    return true;
  }
</script>
```

🔍 **Explanation**:

- When the form is submitted, this function checks the input.
- let name = document.getElementById("name").value;:
    - Gets the value typed into the input field.
- if (name === ""):
    - Checks if the name is **empty**.
    - If it is:
        - Shows an **alert** saying "Name cannot be empty".
        - Returns false, which **prevents the form from submitting**.
- If there *is* a name typed:

    - return true; – allows the form to be submitted.

✅ **So it works like this:**

- If the user leaves the input blank and clicks Submit:
    - It shows an alert and doesn't submit the form.
- If the user types something and clicks Submit:
    - The form submits normally.
