
Introduction to JavaScript

JavaScript (JS) is a **high-level**, **interpreted** programming language that makes web pages interactive.
It works alongside:

- **HTML** (structure of the webpage)
- **CSS** (styling and design of the webpage)

Why Learn JavaScript?

- **Adds interactivity**: Animations, form validation, and more.
- **Dynamic content updates** without reloading the page.
- **Cross-browser support**: Works on all modern web browsers.
- **Rich ecosystem**: Popular libraries like **React** and **Vue.js**.
- **Full-Stack Development**: JavaScript runs on both client-side and server-side with **Node.js**.

Vanilla JavaScript

Vanilla JavaScript refers to using **pure JavaScript** without any libraries or frameworks like jQuery, React, or Angular. It is the raw, unmodified version of JavaScript and is essential for understanding the core language features.

Benefits of Vanilla JS:

- Lightweight and fast.
- No extra overhead from libraries.
- Essential for learning JavaScript fundamentals.
- Works across all browsers.

Versions:

ECMAScript is official name of the language. Think of ECMAScript as the **blueprint** and JavaScript as the **real-world application** of that blueprint.

ES1, ES2, ES3, ES4, ES5 versions and last additions in 2020 of ES6.

JavaScript Basics

1. Variables and Data Types

Variables store data in JavaScript. Declare variables using:

- var (older versions of JS)
- let (recommended for values that may change)
- const (used for constant values)

Example:

```
let name = "John";     // String
const age = 25;        // Number
var isStudent = true;  // Boolean
```

Data Types

1. **Primitive Types** (Immutable):
    - **String**: "Hello"
    - **Number**: 42, 3.14
    - **Boolean**: true, false
    - **Undefined**: A declared but uninitialized variable.
    - **Null**: Represents an empty value.
    - **Symbol**: A unique, immutable identifier.
2. **Non-Primitive Types** (Mutable):
    - **Object**: { key: value }
    - **Array**: [1, 2, 3]
    - **Function**: Reusable code blocks.

---

2. Rules for Identifiers

- Must start with a letter, underscore _ , or dollar sign $.
- Can include letters, numbers, _ , or $ after the first character.
- No spaces allowed.

- **Case-sensitive**: myVar and myvar are different.
- Cannot use reserved keywords (e.g., function, if, let).

---

3. Displaying Output

1. **Console**: Use console.log() for debugging.
2. **Webpage Content**: Modify HTML content using innerHTML.
3. **Pop-up Alerts**: Use alert() for simple notifications.
4. **Write to the Document**: Use document.write() (not recommended for modern projects).

Example:

```
console.log("Hello, World!");  // Console output
alert("Welcome!");             // Pop-up alert
```

---

4. Operators

JavaScript supports the following types of operators:

Type	Example
Arithmetic	+, -, *, /, %
Assignment	=, +=, -=
Comparison	==, ===, !=, >, <
Logical	`&& (AND),
Increment/Decrement	++, --

Example:

```
let x = 10;
let y = 5;
console.log(x + y);  // Outputs: 15
console.log(x > y);  // Outputs: true
```

---

5. Conditional Statements

Conditional statements control the flow of execution based on a condition.

Syntax:

```
if (condition) {
  // Code when condition is true
} else if (anotherCondition) {
  // Code when another condition is true
} else {
  // Code when all conditions are false
}
```

Example:

```
let marks = 85;

if (marks >= 90) {
   console.log("Grade A");
} else if (marks >= 75) {
   console.log("Grade B");
} else {
   console.log("Grade C");
}
```

---

6. Loops

Loops are used to execute a block of code repeatedly.

Types of Loops

1. **For Loop**: Runs a block of code a specific number of times.

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

2. **While Loop**: Runs as long as a condition is true.

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

3. **Do-While Loop**: Executes the code at least once.

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

---

7. Functions

Functions are reusable code blocks that perform specific tasks.

Syntax:

```
function functionName(parameters) {
  // Code block
  return result;
}
```

Example:

```
function add(a, b) {
  return a + b;
}
let result = add(2, 3);
console.log(result);  // Outputs: 5
```

Embedding JavaScript in HTML

You can include JavaScript in an HTML file in three ways:

1. **Inline JavaScript**:

   ```
   <button onclick="alert('Hello!')">Click Me</button>
   ```

2. **Internal JavaScript**:
   Add JavaScript inside <script> tags within the HTML file.

   ```
   <script>
   alert('Hello!');
   </script>
   ```

3. **External JavaScript**:
   Link an external .js file to your HTML.

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

DOM Manipulation

The **Document Object Model (DOM)** represents the webpage as a tree-like structure. JavaScript can access and modify elements using the DOM.

Accessing Elements:

- document.getElementById("id")

- document.querySelector("selector")
- document.querySelectorAll("selector")

Example:

DOM Manipulation:

```
<h1 id=”demo”>Leading</h1>
<button onclick=”mydemo()”>ADD</button>
<script>
   Function mydemo(){
        var sum-=5+8;
        document.getElementById("demo").innerHTML=sum;

        }
</script>
```

JavaScript Events

In JavaScript, **events** refer to actions or occurrences that happen in the browser, which JavaScript can respond to. These events could be triggered by user interactions (like clicking, typing, or scrolling), browser actions (like loading a page), or other events from the system (like a network response).

Types of Events in JavaScript:

1. **Mouse Events**: These events occur when the user interacts with a mouse.
    - click: When a mouse button is pressed and released on an element.
    - mouseenter, mouseleave: When the mouse enters or leaves an element.
    - mousemove: When the mouse moves over an element.
2. **Keyboard Events**: These events are triggered by user interaction via the keyboard.
    - keydown: When a key is pressed down.
    - keyup: When a key is released.
3. **Form Events**: These events are related to form interactions.
    - submit: When a form is submitted.
    - focus: When an input field or element gains focus.
    - blur: When an input field or element loses focus.
    - change: When the value of an input element changes.
4. **Window Events**: These events are related to actions in the browser window.
    - load: When the page finishes loading.
    - resize: When the browser window is resized.
    - scroll: When the user scrolls the page.
    - unload: When the user leaves the page (such as closing the tab).
5. **Touch Events**: These events are triggered by touch interactions on mobile devices.
    - touchstart: When a touch is initiated.
    - touchmove: When a touch moves across the screen.
    - touchend: When a touch ends.

Event Handling in JavaScript:

JavaScript provides several ways to handle events:

1. **Inline Event Handling**: You can attach event handlers directly within HTML elements.
2. **Event Listeners**: You can use JavaScript to add event listeners to HTML elements. This method is more flexible and allows multiple handlers for the same event.
3. **Event Object**: When an event is triggered, an event object is automatically passed to the event handler. This object contains information about the event (like which element was clicked, key pressed, etc.).

Example:

Here's a practical example of an Inline Event Handling :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Inline Event Handling</title>
</head>
<body>
  <!-- Inline event handler for a button click -->
  <button onclick="alert('Button clicked!')">Click Me</button>

  <!-- Inline event handler for a text input change -->
  <input type="text" onchange="alert('Text changed!')" placeholder="Type something..." />
</body>
</html>
```

Example:

Here's a practical example of using an event listener to handle a button click event:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript Events</title>
</head>
<body>

  <button id="myButton">Click Me!</button>

  <script>
    // Get the button element
    const button = document.getElementById('myButton');
```

```
    // Add a click event listener to the button
    button.addEventListener('click', function() {
      alert('Button was clicked!');
    });
  </script>

</body>
</html>
```

In this example, when the user clicks the "Click Me!" button, an alert box will appear with the message "Button was clicked!"

Example:

Here's a practical example of event handling using event object:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Input Event Example</title>
</head>
<body>
  <label for="textInput">Type something:</label>
  <input type="text" id="textInput" />

  <script>
    // Select the input element
    const inputField = document.getElementById('textInput');

    // Add an input event listener
    inputField.addEventListener('input', function(event) {
      // Access event properties
      console.log('Input value:', event.target.value);  //The event object allows you to access the current value of the input using event.target.value.
      console.log('Event type:', event.type);          // Type of event, e.g., "input"
    });
  </script>
</body>
</html>
```