upGrad ENTERPRISE

JavaScript Programming Basics

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

JavaScript Learning Content

1. Introduction to JavaScript

JavaScript (JS) is a **lightweight, interpreted programming language** primarily used to make web pages interactive.
It runs directly in the browser and works alongside **HTML** (structure) and **CSS** (style) to bring web pages to life.

Key Points

- Created by **Brendan Eich** in 1995.
- Initially designed for browsers, now used in **servers (Node.js)**, **mobile apps**, and **desktop applications**.
- It’s an **interpreted** language (no need to compile).
- **Case-sensitive** and supports **object-oriented**, **functional**, and **event-driven** programming.

2. Why Do We Need JavaScript?

HTML and CSS alone can only create static web pages.
JavaScript adds **interactivity**, **logic**, and **dynamic behavior**.

What JavaScript Can Do:

- Validate forms (e.g., check if email is valid before submitting).
- Create dynamic content updates without reloading the page (using AJAX).
- Control multimedia (audio, video).
- Handle user interactions (clicks, scrolls, input).
- Power modern front-end frameworks like **React**, **Angular**, and **Vue**.

3. How to Include JavaScript in a Web Page

There are three main ways:

a) Inline JavaScript

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

b) Internal JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <title>My JS Page</title>
  <script>
    function greet() {
      alert("Welcome to JavaScript!");
    }
  </script>
</head>
<body onload="greet()">
</body>
</html>
```

c) External JavaScript File

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

> Recommended for larger projects to separate logic and design.

4. JavaScript Syntax

The **syntax** defines the set of rules for writing JavaScript code.

Examples:

```
console.log("Hello World!");    // Output text to console
let x = 10;                     // Variable declaration
if (x > 5) {
  console.log("x is greater than 5");
}
```

📄 Notes:

- Statements end with a **semicolon** (`;`).
- Code blocks are enclosed in **curly braces** `{}` .
- JavaScript is **case-sensitive** (`MyVar` ≠ `myvar`).

5. Variables

Variables store data values.

Declaring Variables:

Keyword	Description
`var`	Function-scoped (older, avoid using)
`let`	Block-scoped, can be reassigned
`const`	Block-scoped, cannot be reassigned

Example:

```
let name = "Alice";
const PI = 3.14;
var age = 25;
```

6. Data Types

JavaScript has two major categories:

Primitive and **Non-Primitive (Reference)**

Primitive Types:

- `String` → `"Hello"`
- `Number` → `42`, `3.14`
- `Boolean` → `true`, `false`
- `Undefined` → variable declared but not assigned
- `Null` → intentional empty value

Non-Primitive:

- `Object`
- `Array`
- `Function`

Example:

```
let person = { name: "John", age: 30 };
let colors = ["red", "green", "blue"];
```

7. Control Flow

a) Conditional Statements

```
let age = 18;

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

b) Switch Statement

```
let day = 2;

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

c) Loops

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

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

8. Functions

Functions are **reusable blocks of code** that perform a task.

Syntax:

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

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

Function Expression:

```
const add = function(a, b) {
  return a + b;
};
```

9. Variable Scopes

Scope determines where a variable is accessible.

Types of Scope:

- **Global Scope** – accessible anywhere
- **Function Scope** – accessible inside the function
- **Block Scope** – variables declared with `let` or `const` inside `{}`

Example:

```
let x = 10; // global

function demo() {
  let y = 20; // local
  console.log(x + y);
}

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

10. Arrow Functions

Introduced in **ES6**, arrow functions are shorter function syntax.

Syntax:

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

// For one-line functions:
const square = n => n * n;
```

> Arrow functions do **not** have their own `this` context — they inherit it from the parent scope.

11. Callback Basics

A **callback** is a function passed as an argument to another function, to be executed later.

Example:

```
function greetUser(name, callback) {
  console.log("Hello " + name);
  callback();
}

function sayGoodbye() {
  console.log("Goodbye!");
}

greetUser("Alice", sayGoodbye);
```

Use Case:

Callbacks are heavily used in:

- Event handling
- Asynchronous operations (e.g., `setTimeout`, API calls)

Summary

Concept	Key Takeaway
JavaScript	Adds interactivity to web pages
Inclusion	Inline, Internal, External
Variables	Declared using `var`, `let`, `const`
Data Types	Primitive & Non-Primitive
Control Flow	if, switch, loops
Functions	Reusable code blocks
Scopes	Global, Function, Block
Arrow Functions	Shorter syntax, no own `this`
Callbacks	Functions passed into functions

upGrad ENTERPRISE

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com