Contents

Introduction to JavaScript

Introduction

JavaScript is a versatile, high-level programming language primarily used for creating dynamic and interactive web applications. It is one of the core technologies of the web, alongside HTML and CSS. Initially developed for client-side scripting, JavaScript is now also widely used for server-side programming.

Features of JavaScript:

- **Lightweight and interpreted**: JavaScript does not require compilation, making it fast and easy to use.
- **Object-oriented**: JavaScript supports objects, prototypes, and classes.
- **Supports asynchronous programming**: With promises and async/await, JavaScript handles asynchronous operations efficiently.
- **Platform-independent**: It can run on any device with a web browser.
- **Event-driven programming**: It reacts to user actions like clicks and keypresses.

JavaScript Syntax

JavaScript syntax consists of rules defining how programs should be written.

Basic Syntax Rules:

- **Statements** are written inside <script> tags when used in an HTML document.
- **Statements end** with a semicolon (;), though it is optional.
- **Variables are declared** using var, let, or const.
- **Comments** are added using // for single-line and /* */ for multi-line.
- **Code blocks** are enclosed in {}.

```
// Declaring a variable
let message = "Hello, JavaScript!";
console.log(message);
```

Data Types in JavaScript

JavaScript supports different types of data:

1. **Primitive Data Types:**
    - String: Represents text values ("Hello")
    - Number: Represents integers and floating points (10, 10.5)
    - Boolean: Represents true or false

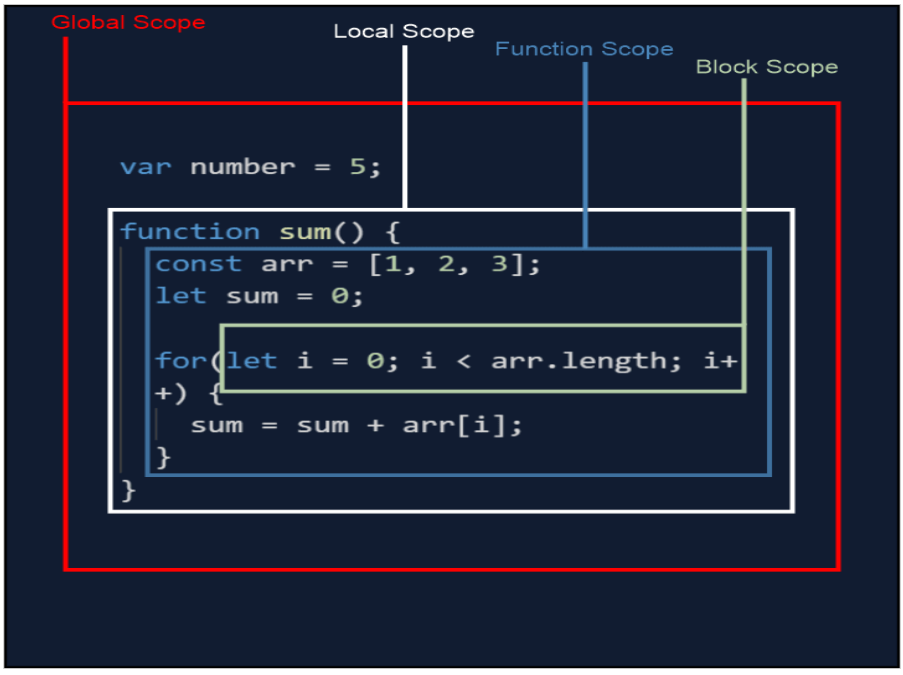
- Undefined: A variable declared but not assigned a value
- Null: Represents an empty or unknown value
- BigInt: Used for large integers
- Symbol: Used to create unique identifiers

2. **Non-Primitive (Reference) Data Types:**
   - Object: A collection of properties ({name: "John", age: 25})
   - Array: A list of values ([1, 2, 3, 4])
   - Function: A reusable block of code

Difference Between var**,** let**, and** const **in JavaScript:**

Feature	var	let	const
Scope	Function-scoped	Block-scoped	Block-scoped
Re-declaration	Allowed	Not allowed	Not allowed
Reassignment	Allowed	Allowed	Not allowed
Hoisting	Hoisted with undefined	Hoisted but not initialized	Hoisted but not initialized
Use Case	Older JavaScript code	Preferred for variables that change	Preferred for constants

Scope in java:



Conditions in JavaScript

JavaScript uses conditional statements to execute code based on conditions.

If Statement

Executes a block of code if a condition is true.

```
let age = 18;
if (age >= 18) {
    console.log("You are an adult");
}
```

If-Else Statement

Provides an alternative block if the condition is false.

```
let age = 16;
if (age >= 18) {
    console.log("You can vote");
} else {
    console.log("You cannot vote");
}
```

Switch Statement

Used when there are multiple possible values.

```
let day = "Monday";
switch (day) {
    case "Monday":
        console.log("Start of the workweek");
        break;
    case "Friday":
        console.log("Weekend is near");
        break;
    default:
        console.log("Regular day");
}
```

Loops in JavaScript

Loops execute a block of code multiple times.

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

While Loop

Executes a block of code while a condition is true.

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

Do-While Loop

Executes the block of code at least once before checking the condition.

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

Alert Boxes in JavaScript

JavaScript provides alert boxes to display messages to users.

- alert(): Displays a pop-up box with a message.

```
// Declaring a variable
let message = "Hello, JavaScript!";
console.log(message);
```

- confirm(): Displays a dialog box with OK and Cancel buttons.

```
let result = confirm("Are you sure?");
console.log(result); // true or false
```

- prompt(): Takes user input.

```
let name = prompt("Enter your name:");
console.log("Hello, " + name);
```

Operators in JavaScript

JavaScript supports various operators:

1. Arithmetic Operators

Used to perform mathematical operations.

Operator	Description	Example	Output
+	Addition	5 + 3	8
-	Subtraction	10 - 4	6
*	Multiplication	6 * 2	12
/	Division	9 / 3	3
%	Modulus (Remainder)	10 % 3	1
++	Increment	let a = 5; a++	6
--	Decrement	let b = 8; b--	7

```
let a = 10;
let b = 3;
console.log(a + b); // 13
console.log(a % b); // 1
```

2. Comparison Operators

Used to compare values and return true or false.

Operator	Description	Example	Output
==	Equal to (checks value)	5 == "5"	true
!=	Not equal	10 != 5	true
>	Greater than	8 > 4	true
<	Less than	3 < 7	true
>=	Greater than or equal	6 >= 6	true
<=	Less than or equal	4 <= 5	true

===	Strict equal (checks value & type)	5 === "5"	false
!==	Strict not equal	5 !== "5"	true

```
console.log(10 > 5); // true
console.log(5 === "5"); // false
```

3. Logical Operators

Used to combine multiple conditions.

Operator	Description	Example	Output
&&	AND (both conditions must be true)	(5 > 2) && (10 > 5)	true
`		`	OR (at least one condition is true)
!	NOT (negates a condition)	!(5 > 2)	false

```
let x = 5, y = 10;
console.log(x > 0 && y > 5); // true
console.log(x < 0 || y > 5); // true
console.log(!(x < 0)); // true
```

4. Assignment Operators

Used to assign values to variables.

Operator	Description	Example	Equivalent to	Output
=	Assign	a = 10	-	10
+=	Add and assign	a += 5	a = a + 5	15
-=	Subtract and assign	a -= 3	a = a - 3	12
*=	Multiply and assign	a *= 2	a = a * 2	24

/=	Divide and assign	a /= 4	a = a / 4	6
%=	Modulus and assign	a %= 3	a = a % 3	0

```
let num = 10;
num += 5; // num = num + 5
console.log(num); // 15
```

6. Ternary Operator

A shorthand for if-else.

Operator	Description	Example	Output
? :	If condition is true, return first value; otherwise, return second	let result = (10 > 5) ? "Yes" : "No";	"Yes"

JavaScript String Operations:

Operation	**Example**	**Output**
Declaring Strings	let str = "Hello";	"Hello"
Concatenation	"John" + " Doe"	"John Doe"
Template Literals	`Hello ${name}`	"Hello John" (if name="John")
String Length	"Hello".length	5
Accessing Characters	"JavaScript"[0]	"J"
Changing Case	"Hello".toUpperCase()	"HELLO"
	"WORLD".toLowerCase()	"world"
Finding Substring	"JavaScript".indexOf("Script")	4
	"Hello World".includes("World")	true
Extracting Substring	"JavaScript".slice(0, 4)	"Java"
	"JavaScript".substring(4, 10)	"Script"
Replacing Text	"I love JavaScript".replace("JavaScript", "Python")	"I love Python"
Splitting Strings	"Apple, Banana".split(", ")	["Apple", "Banana"]
Trimming Spaces	" Hello ".trim()	"Hello"
Comparing Strings	"apple" > "banana"	false

Repeating Strings	"Ha! ".repeat(3)	"Ha! Ha! Ha! "
Checking Start	"JavaScript".startsWith("Java")	true
Checking End	"Hello World".endsWith("World")	true

Functions in JavaScript

Functions allow code reusability and modularity.

Traditional Function

Defined using the **function** keyword.

```
function greet(name) {
    return "Hello, " + name;
}
console.log(greet("John"));
```

Arrow Function

A shorter syntax for defining functions.

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

Feature	Traditional Function	Arrow Function
this keyword	Refers to the function's scope	Inherits from parent scope
Syntax	Longer, function keyword needed	Shorter, concise
Use Case	Good for object methods	Good for callbacks & concise functions