Table of Contents

[What is JavaScript? 2](#_Toc199320437)

[Where Can You Use JavaScript? 2](#_Toc199320438)

[The Birth of JavaScript 2](#_Toc199320439)

[Standardization and ECMAScript 2](#_Toc199320440)

[Key Milestones in JavaScript and ECMAScript History 3](#_Toc199320441)

[Basic to Advanced JavaScript Concepts 3](#_Toc199320442)

[1. JavaScript Basics 3](#_Toc199320443)

[2. Variables 4](#_Toc199320444)

[3. Data Types 5](#_Toc199320445)

[4. Operators 8](#_Toc199320446)

[5. Functions 8](#_Toc199320447)

[6. Conditional Statements 13](#_Toc199320448)

[7. Loops 17](#_Toc199320449)

[8. Objects 20](#_Toc199320450)

[9. Arrays 23](#_Toc199320451)

[10. JavaScript String Methods 27](#_Toc199320452)

[11. Math and Number Functions 28](#_Toc199320453)

[12. Date Functions 28](#_Toc199320454)

[13. m 29](#_Toc199320455)

[JavaScript Core Topics List: 29](#_Toc199320456)

[1. Hoisting 29](#_Toc199320457)

[Explanation: 29](#_Toc199320458)

[Syntax Example: 29](#_Toc199320459)

[HTML Example: 29](#_Toc199320460)

[2. Closures 29](#_Toc199320461)

[Explanation: 29](#_Toc199320462)

[Syntax Example: 30](#_Toc199320463)

[HTML Example: 30](#_Toc199320464)

[3. Callback Functions 30](#_Toc199320465)

[Explanation: 30](#_Toc199320466)

[Syntax Example: 30](#_Toc199320467)

[HTML Example: 30](#_Toc199320468)

[4. DOM Tree 31](#_Toc199320469)

[Explanation: 31](#_Toc199320470)

[HTML Example: 31](#_Toc199320471)

# **What is JavaScript?**

JavaScript is a versatile, high-level programming language primarily used for web development. It allows you to create interactive and dynamic web pages. JavaScript is one of the core technologies of the web, alongside HTML and CSS.

# **Where Can You Use JavaScript?**

JavaScript can be used in various areas, including:

1. **Web Development**: Adding interactivity to websites
2. **Server-Side Development**: Using environments like Node.js to build scalable network applications.
3. **Mobile App Development**: Frameworks like React Native allow you to build mobile apps.
4. **Game Development**: Creating browser-based games.
5. **Desktop Applications**: Using frameworks like Electron to build cross-platform desktop apps.
6. **Internet of Things (IoT)**: Controlling hardware devices.
7. **Blockchain**: Developing decentralized applications and smart contracts
8. **Machine Learning and AI**: Implementing machine learning models and AI algorithms using libraries like TensorFlow.js

# **The Birth of JavaScript**

JavaScript was created by **Brendan Eich** in 1995. Initially, it was called “Mocha,” later renamed to “LiveScript,” and finally to “JavaScript” to capitalize on the popularity of Java at the time.

JavaScript was designed to be a lightweight scripting language for adding interactivity to web pages. It quickly became popular due to its ease of use and integration with HTML.

Jscript

xyz

### Standardization and ECMAScript

As JavaScript gained popularity, there was a need to standardize the language to ensure consistency across different web browsers. In 1996, Netscape and Brendan Eich took JavaScript to the European Computer Manufacturers Association (ECMA) to create a standardized version of the language. This led to the formation of the ECMA Technical Committee 39 (TC39), which was responsible for developing the language specification.

In 1997, the first edition of the ECMAScript standard (ECMA-262) was published. **ECMAScript** is the official name of the language specification, while JavaScript is the implementation of that specification in web browsers.

**JavaScript is like a car**: Imagine JavaScript as a specific model of a car, like a Toyota Corolla. It’s a tangible product that people use and interact with.

**ECMAScript is like the blueprint**: ECMAScript, on the other hand, is the blueprint or design specification for that car. It defines how the car should be built, what features it should have, and how it should function.

**Different Implementations**: Just as different manufacturers can build cars based on the same blueprint, different web browsers (like Chrome, Firefox, and Safari) implement JavaScript based on the ECMAScript specification. This ensure that JavaScript code runs consistently across different browsers.

### Key Milestones in JavaScript and ECMAScript History

* **1995**: Brendan Eich creates JavaScript at Netscape.
* **1996**: JavaScript is taken to ECMA for standardization.
* **1997**: The first edition of ECMAScript (ECMA-262) is published.
* **1999**: ECMAScript 3 (ES3) is released, bringing significant improvements.
* **2009**: ECMAScript 5 (ES5) is released, introducing many new features.
* **2015**: ECMAScript 6 (ES6), also known as ECMAScript 2015, is released, bringing major enhancements like let, const, arrow functions, classes, and modules.
* **Ongoing**: New versions of ECMAScript are released annually, with continuous improvements and new features.
* **ESNext……>**
* **TypeScript** ?? – Angular ??

# **Basic to Advanced JavaScript Concepts**

### JavaScript Basics

#### **Reserved Keywords**

JavaScript has a set of reserved keywords that cannot be used as variable names, function names, or identifiers.

break, case, catch, class, const, continue, debugger, default, delete, do, else, export, extends, finally, for, function, if, import, in, instanceof, let, new, return, super, switch, this, throw, try, typeof, var, void, while, with, yield, enum, await, implements, interface, package, private, protected, public, static

#### **Internal JavaScript**

Written directly inside HTML using <script> tag.

<script>

alert("This is internal JavaScript!");

</script>

#### **External JavaScript**

Stored in a separate .js file.

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

### Variables

In JavaScript, a variable is a container that stores a value.

Here are some key aspects of variables:

* **Declaration:** You create a variable using the let, const, or (in older code) var keywords, followed by

the variable's name.

* + Example: let myVariable;
* **Assignment:** You store a value in a variable using the assignment operator (=).
  + Example: myVariable = 5;
* **Types:** JavaScript is dynamically typed, meaning a variable can hold different types of values (numbers, strings, booleans, objects, etc.) at different times.
  + Example:
    - myVariable = 5; (holds a number)
    - myVariable = "Hello"; (now holds a string)
* **Scope:** The scope of a variable defines where it can be accessed in your code.
  + let and const create block-scoped variables. It accessible within the nearest set of curly braces {}
  + var creates function-scoped variables (accessible within the entire function).
  + **Global Scope:** Outside any function
  + **Local Scope:** Inside a function
  + **Block Scope:** Inside {} when using let or const

let a = 10; // Global

let c = 15; // global

function test() {

let b = 20; // Local

c = 25;

console.log(c) // 25

}

console.log(c) // 25 // global

* **Mutability**:
  + Variables declared with let can be reassigned new values later in the code.
  + Variables declared with const cannot be reassigned after their initial assignment (they are constant).

###### Let

Used to declare block-scoped variables.

let x = 10;

if (true) {

let x = 20;

console.log(x); // Output: 20

}

console.log(x); // Output: 10

###### Const

Used to declare block-scoped constants. The value of a const variable cannot be changed through reassignment.

const y = 30;

y = 40; // Error: Assignment to constant variable.

###### Var

Function-scoped or globally-scoped variable. It’s generally avoided in modern JavaScript due to its lack of block-scoping.

var z = 50;

if (true) {

var z = 60;

console.log(z); // Output: 60

}

console.log(z); // Output: 60

### Data Types

JavaScript is a dynamically typed language, meaning you don’t need to specify the data type explicitly when declaring a variable. The data type of a variable is determined automatically based on the assigned value. Let’s explore variables and data types in ES6 in detail.

JavaScript has several built-in data types:

###### Number

Represents both integer and floating-point numbers.

let age = 25; // Integer

let price = 19.99; // Floating-point

###### String

Represents a sequence of characters.

let myname = "Gaurab Kumar";

let greeting = `Hello, ${myname}!`; // Template literal

###### Boolean

Represents a logical entity and can have two values: true or false.

let isActive = true;

let isComplete = false

###### Undefined

A variable that has been declared but not assigned a value.

let x;

console.log(x); // Output: undefined

###### Null

Represents the intentional absence of any object value.

let y = null;

console.log(y); // Output: null

###### Symbol

A unique and immutable primitive value.

let sym = Symbol("unique"); console.log(sym); // Output: Symbol(unique)

###### Object

A collection of properties.

let person = { name: "John", age: 30

};

console.log(person.name); // Output: John

###### Array

A special type of object used to store ordered collections.

let colors = ["red", "green", "blue"]; console.log(colors[0]); // Output: red

###### Example: HTML and JavaScript Using Variables and Data Types

Here’s an example that demonstrates the use of various variables and data types in an HTML file with embedded JavaScript:

<!DOCTYPE html>

<html>

<head>

<title>JavaScript Variables and Data Types</title>

</head>

<body>

<h1>JavaScript Variables and Data Types</h1>

<p id="demo"></p>

<script>

// Number

let age = 25;

let price = 19.99;

// String

let name = "John Doe";

let greeting = `Hello, ${name}!`;

// Boolean

let isActive = true;

// Undefined let x;

// Null

let y = null;

// Symbol

let sym = Symbol("unique");

// Object

let person = { name: "John", age: 30

};

// Array

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

// Displaying the values document.getElementById("demo").innerHTML = `

<strong>Number:</strong> Age = ${age}, Price = ${price}<br>

<strong>String:</strong> ${greeting}<br>

<strong>Boolean:</strong> Is Active = ${isActive}<br>

<strong>Undefined:</strong> x = ${x}<br>

<strong>Null:</strong> y = ${y}<br>

<strong>Symbol:</strong> ${sym.toString()}<br>

<strong>Object:</strong> Name = ${person.name}, Age =

${person.age}<br>

<strong>Array:</strong> Colors = ${colors.join(", ")}

`;

</script>

</body>

</html>

In this example, we declare various variables using let and const, and then display their values in the HTML content using JavaScript. This demonstrates how different data types can be used and manipulated in JavaScript.

### Operators

#### ***Arithmetic***

**+, -, \*, /, %, ++, --**

let x = 10, y = 5;

console.log(x + y); // 15

#### Assignment

**=, +=, -=, \*=, /=**

let z = 5;

z += 2; // 7

#### Comparison

**==, ===, !=, !==, >, <, >=, <=**

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

console.log(5 === "5"); // false

#### Logical

**&&, ||, !**

let a = true, b = false;

console.log(a && b); // false

#### Example:

### Functions

**Functions** are blocks of code designed to perform a particular task and allowing you to encapsulate code for reuse. ES6 and newer versions of ECMAScript introduced several new features and improvements for functions. Let’s explore these in detail.

###### Function Declarations

A function declaration defines a function with the specified parameters.

function greet(name) {

return `Hello, ${name}!`;

}

console.log(greet("John")); // Output: Hello, John!

###### Function Expressions

A function expression defines a function as part of a larger expression, typically assigned to a variable.

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

};

console.log(greet("John")); // Output: Hello, John!

###### Arrow Functions

Arrow functions provide a shorter syntax for writing function expressions. They do not have their own

**this** context.

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

###### Default Parameters

Default parameters allow you to initialize parameters with default values if no value or undefined is passed.

function greet(name = "Guest") { return `Hello, ${name}!`;

}

console.log(greet()); // Output: Hello, Guest! console.log(greet("John")); // Output: Hello, John!

###### Rest Parameters

Rest parameters allow you to represent an indefinite number of arguments as an array.

function sum(...numbers) {

return numbers.reduce((acc, num) => acc + num, 0);

}

console.log(sum(1, 2, 3)); // Output: 6

###### Spread Operator

The spread operator allows an iterable (like an array) to be expanded in places where zero or more arguments or elements are expected.

const numbers = [1, 2, 3];

const newNumbers = [...numbers, 4, 5];

console.log(newNumbers); // Output: [1, 2, 3, 4, 5]

###### Anonymous Functions

Anonymous functions are functions without a name. They are often used as arguments to other functions.

setTimeout(function() {

console.log("This is an anonymous function");

}, 1000);

###### Immediately Invoked Function Expressions (IIFE)

An IIFE is a function that runs as soon as it is defined.

(function() {

console.log("This is an IIFE");

})();

###### Higher-Order Functions

Higher-order functions are functions that take other functions as arguments or return functions as their result.

function higherOrderFunction(callback) { return callback();

}

function sayHello() { return "Hello!";

}

console.log(higherOrderFunction(sayHello)); // Output: Hello!

###### Closures

A closure is a function that retains access to its lexical scope even when the function is executed outside that scope.

function outerFunction() {

let outerVariable = "I'm outside!";

function innerFunction() { console.log(outerVariable);

}

return innerFunction;

}

const myFunction = outerFunction(); myFunction(); // Output: I'm outside!

#### **Example: HTML and JavaScript Project Using Functions**

Let’s create a simple HTML and JavaScript project that demonstrates the use of various functions.

<!DOCTYPE html>

<html>

<head>

<title>ES6 Functions Demo</title>

</head>

<body>

<h1>ES6 Functions Demo</h1>

<p id="output"></p>

<button onclick="showGreeting()">Show Greeting</button>

<button onclick="calculateSum()">Calculate Sum</button>

<button onclick="executeIIFE()">Execute IIFE</button>

<script>

// Function Declaration

function greet(name = "Guest") { return `Hello, ${name}!`;

}

// Arrow Function

const sum = (...numbers) => numbers.reduce((acc, num) => acc + num, 0);

// Anonymous Function

const showGreeting = function() { document.getElementById("output").textContent = greet("John");

};

// IIFE

const executeIIFE = (function() { return function() {

console.log("IIFE executed"); document.getElementById("output").textContent = "IIFE

executed";

};

})();

// Function using Rest Parameters and Spread Operator function calculateSum() {

const numbers = [1, 2, 3, 4, 5]; const result = sum(...numbers);

document.getElementById("output").textContent = `Sum: ${result}`;

}

</script>

</body>

</html>

In this project:

* We define a function greet with a default parameter.
* We use an arrow function sum to calculate the sum of numbers.
* We use an anonymous function showGreeting to display a greeting message.
* We use an IIFE to log a message immediately and update the DOM.
* We use rest parameters and the spread operator in the calculateSum function.

### Conditional Statements

Conditionals are used to perform different actions based on different conditions. They are essential for controlling the flow of your code. Let’s explore the various types of conditionals in ES6 and newer versions of ECMAScript.

###### if...else Statements

The if...else statement executes a block of code if a specified condition is true. If the condition is false, another block of code can be executed.

let age = 18;

if (age >= 18) {

console.log("You are an adult.");

} else {

console.log("You are a minor.");

}

###### else if Statements

The else if statement allows you to test multiple conditions.

let score = 85;

if (score >= 90) { console.log("Grade: A");

} else if (score >= 80) { console.log("Grade: B");

} else if (score >= 70) { console.log("Grade: C");

} else {

console.log("Grade: F");

}

###### Ternary Operator

The ternary operator is a shorthand for the if...else statement. It takes three operands: a condition, an expression to execute if the condition is true, and an expression to execute if the condition is false.

let age = 18;

let message = (age >= 18) ? "You are an adult." : "You are a minor."; console.log(message);

###### switch Statements

The switch statement evaluates an expression and executes code blocks based on matching case values.

let day = 3; let dayName;

switch (day) { case 1:

dayName = "Monday"; break;

case 2:

dayName = "Tuesday"; break;

case 3:

dayName = "Wednesday"; break;

case 4:

dayName = "Thursday"; break;

case 5:

dayName = "Friday"; break;

case 6:

dayName = "Saturday"; break;

case 7:

dayName = "Sunday";

break; default:

dayName = "Invalid day";

}

console.log(dayName); // Output: Wednesday

###### Example: HTML and JavaScript Project Using Conditionals

Let’s create a simple HTML and JavaScript project that demonstrates the use of various conditionals.

<!DOCTYPE html>

<html>

<head>

<title>Conditionals Demo</title>

</head>

<body>

<h1>Conditionals Demo</h1>

<p id="output"></p>

<button onclick="checkAge()">Check Age</button>

<button onclick="getGrade()">Get Grade</button>

<button onclick="getDayName()">Get Day Name</button>

<script>

// if...else Statement function checkAge() {

let age = prompt("Enter your age:"); if (age >= 18) {

document.getElementById("output").textContent = "You are an

adult.";

minor.";

} else {

document.getElementById("output").textContent = "You are a

}

}

// else if Statement function getGrade() {

let score = prompt("Enter your score:"); if (score >= 90) {

document.getElementById("output").textContent = "Grade: A";

} else if (score >= 80) { document.getElementById("output").textContent = "Grade: B";

} else if (score >= 70) { document.getElementById("output").textContent = "Grade: C";

} else {

document.getElementById("output").textContent = "Grade: F";

}

}

// switch Statement function getDayName() {

let day = prompt("Enter a day number (1-7):"); let dayName;

switch (parseInt(day)) { case 1:

|  |  |  |
| --- | --- | --- |
| dayName | = | "Monday"; |
| break; |  |  |
| case 2: |  |  |
| dayName | = | "Tuesday"; |
| break; |  |  |
| case 3: |  |  |
| dayName | = | "Wednesday"; |
| break; |  |  |
| case 4: |  |  |
| dayName | = | "Thursday"; |
| break; |  |  |
| case 5: |  |  |
| dayName | = | "Friday"; |
| break; |  |  |
| case 6: |  |  |
| dayName | = | "Saturday"; |
| break; |  |  |
| case 7: |  |  |
| dayName | = | "Sunday"; |
| break; |  |  |
| default: |  |  |
| dayName | = | "Invalid day"; |

}

document.getElementById("output").textContent = dayName;

}

</script>

</body>

</html>

In this project:

* We use an if...else statement to check the user’s age.
* We use an else if statement to determine the user’s grade based on their score.
* We use a switch statement to get the name of the day based on a number input.

Feel free to ask if you have any specific questions or need further explanations on any topic!

### Loops

**Loops** are used to repeat a block of code multiple times. Let’s explore these in detail.

###### for Loop

The for loop is the most common loop in JavaScript. It repeats a block of code a specified number of times.

for (let i = 0; i < 5; i++) { console.log(i); // Output: 0 1 2 3 4

}

###### for...in Loop

The for...in loop iterates over the enumerable properties of an object.

const person = { name: "John", age: 30, city: "New York" }; for (let key in person) {

console.log(`${key}: ${person[key]}`);

}

// Output:

// name: John

// age: 30

// city: New York

###### for...of Loop

The for...of loop iterates over iterable objects like arrays, strings, and NodeLists.

const colors = ["red", "green", "blue"]; for (let color of colors) {

console.log(color); // Output: red green blue

}

###### while Loop

The while loop executes a block of code as long as a specified condition is true.

let i = 0; while (i < 5) {

console.log(i); // Output: 0 1 2 3 4 i++;

}

###### do...while Loop

The do...while loop is similar to the while loop, but it executes the block of code at least once before checking the condition.

let i = 0; do {

console.log(i); // Output: 0 1 2 3 4 i++;

} while (i < 5);

###### 5.C. break and continue Statements

The break statement exits the loop entirely, while the continue statement skips the current iteration and proceeds to the next one.

for (let i = 0; i < 5; i++) { if (i === 3) break;

console.log(i); // Output: 0 1 2

}

for (let i = 0; i < 5; i++) { if (i === 3) continue;

console.log(i); // Output: 0 1 2 4

}

###### HTML and JavaScript Project Using Loops

Let’s create a simple HTML and JavaScript project that demonstrates the use of various loops.

<!DOCTYPE html>

<html>

<head>

<title>Loops Demo</title>

</head>

<body>

<h1>Loops Demo</h1>

<p id="output"></p>

<button onclick="showNumbers()">Show Numbers</button>

<button onclick="showProperties()">Show Properties</button>

<button onclick="showColors()">Show Colors</button>

<script>

// for Loop

function showNumbers() { let output = "";

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

}

document.getElementById("output").textContent = output;

}

// for...in Loop

function showProperties() {

const person = { name: "John", age: 30, city: "New York" }; let output = "";

for (let key in person) {

output += `${key}: ${person[key]} `;

}

document.getElementById("output").textContent = output;

}

// for...of Loop function showColors() {

const colors = ["red", "green", "blue"]; let output = "";

for (let color of colors) { output += color + " ";

}

document.getElementById("output").textContent = output;

}

</script>

</body>

</html>

In this project:

* We use a for loop to display numbers from 0 to 4.
* We use a for...in loop to display the properties of an object.
* We use a for...of loop to display the elements of an array.

Feel free to ask if you have any specific questions or need further explanations on any topic!

### Objects

**Objects** are collections of properties. Let’s explore these in detail.

###### Object Literals

Object literals are a simple way to create objects using curly braces {}. const person = {

name: "John", age: 30,

city: "New York"

};

console.log(person.name); // Output: John

###### Property Shorthand

If the property name and variable name are the same, you can use shorthand syntax.

const name = "John"; const age = 30;

const person = { name, age };

console.log(person); // Output: { name: "John", age: 30 }

###### Method Definitions

ES6 allows you to define methods in objects without the function keyword.

const person = { name: "John",

greet() {

return `Hello, ${this.name}!`;

}

};

console.log(person.greet()); // Output: Hello, John!

###### Computed Property Names

You can use expressions as property names by enclosing them in square brackets []. const prop = "name";

const person = { [prop]: "John", age: 30

};

console.log(person.name); // Output: John

###### Object Destructuring

Object destructuring allows you to extract properties from an object and assign them to variables.

const person = { name: "John", age: 30 }; const { name, age } = person; console.log(name); // Output: John console.log(age); // Output: 30

###### Spread Operator

The spread operator ... allows you to copy properties from one object to another.

const person = { name: "John", age: 30 };

const newPerson = { ...person, city: "New York" };

console.log(newPerson); // Output: { name: "John", age: 30, city: "New York" }

###### Object.assign()

The Object.assign() method copies properties from one or more source objects to a target object.

const person = { name: "John" };

const details = { age: 30, city: "New York" };

const newPerson = Object.assign({}, person, details);

console.log(newPerson); // Output: { name: "John", age: 30, city: "New York" }

###### HTML and JavaScript Project Using Objects

Let’s create a simple HTML and JavaScript project that demonstrates the use of various object features.

<!DOCTYPE html>

<html>

<head>

<title>Objects Demo</title>

</head>

<body>

<h1>Objects Demo</h1>

<p id="output"></p>

<button onclick="showPerson()">Show Person</button>

<button onclick="updatePerson()">Update Person</button>

<script>

// Object Literal const person = {

name: "John", age: 30,

city: "New York", greet() {

return `Hello, ${this.name}!`;

}

};

// Displaying the object function showPerson() {

document.getElementById("output").textContent = JSON.stringify(person, null, 2);

}

// Updating the object using spread operator and property shorthand function updatePerson() {

const newDetails = { age: 31, city: "Los Angeles" }; Object.assign(person, newDetails); document.getElementById("output").textContent =

JSON.stringify(person, null, 2);

}

</script>

</body>

</html>

In this project:

1. We define an object person using an object literal.
2. We use a method greet within the object.
3. We display the object properties in the HTML content.
4. We update the object properties using Object.assign() and the spread operator.

Feel free to ask if you have any specific questions or need further explanations on any topic!

### Arrays

**Arrays** are ordered collections of items. Let’s explore these in detail.

###### Creating Arrays

You can create arrays using array literals or the Array constructor.

// Array literal

const fruits = ["Apple", "Banana", "Cherry"];

// Array constructor

const numbers = new Array(1, 2, 3, 4, 5);

###### Accessing and Modifying Elements

You can access and modify array elements using their index.

const fruits = ["Apple", "Banana", "Cherry"]; console.log(fruits[0]); // Output: Apple

fruits[1] = "Blueberry";

console.log(fruits); // Output: ["Apple", "Blueberry", "Cherry"]

###### Array Methods

ES6 introduced several new methods for arrays, making it easier to manipulate and work with them.

##### forEach()

Executes a provided function once for each array element.

const fruits = ["Apple", "Banana", "Cherry"]; fruits.forEach(fruit => console.log(fruit));

// Output: Apple Banana Cherry

##### map()

Creates a new array with the results of calling a provided function on every element.

const numbers = [1, 2, 3, 4, 5];

const doubled = numbers.map(num => num \* 2);

console.log(doubled); // Output: [2, 4, 6, 8, 10]

##### filter()

Creates a new array with all elements that pass the test implemented by the provided function.

const numbers = [1, 2, 3, 4, 5];

const even = numbers.filter(num => num % 2 === 0); console.log(even); // Output: [2, 4]

##### reduce()

Executes a reducer function on each element of the array, resulting in a single output value.

const numbers = [1, 2, 3, 4, 5];

const sum = numbers.reduce((total, num) => total + num, 0); console.log(sum); // Output: 15

##### find()

Returns the value of the first element that satisfies the provided testing function.

const numbers = [1, 2, 3, 4, 5];

const found = numbers.find(num => num > 3); console.log(found); // Output: 4

##### findIndex()

Returns the index of the first element that satisfies the provided testing function.

const numbers = [1, 2, 3, 4, 5];

const index = numbers.findIndex(num => num > 3); console.log(index); // Output: 3

##### some()

Tests whether at least one element in the array passes the test implemented by the provided function.

const numbers = [1, 2, 3, 4, 5];

const hasEven = numbers.some(num => num % 2 === 0); console.log(hasEven); // Output: true

##### every()

Tests whether all elements in the array pass the test implemented by the provided function.

const numbers = [1, 2, 3, 4, 5];

const allEven = numbers.every(num => num % 2 === 0); console.log(allEven); // Output: false

##### includes()

Determines whether an array includes a certain value among its entries.

const fruits = ["Apple", "Banana", "Cherry"]; const hasBanana = fruits.includes("Banana"); console.log(hasBanana); // Output: true

##### concat()

Merges two or more arrays.

const array1 = [1, 2, 3];

const array2 = [4, 5, 6];

const merged = array1.concat(array2); console.log(merged); // Output: [1, 2, 3, 4, 5, 6]

##### slice()

Returns a shallow copy of a portion of an array into a new array object.

const fruits = ["Apple", "Banana", "Cherry", "Date"]; const sliced = fruits.slice(1, 3); console.log(sliced); // Output: ["Banana", "Cherry"]

##### splice()

Changes the contents of an array by removing or replacing existing elements and/or adding new elements.

const fruits = ["Apple", "Banana", "Cherry"]; fruits.splice(1, 1, "Blueberry");

console.log(fruits); // Output: ["Apple", "Blueberry", "Cherry"]

#### HTML and JavaScript Project Using Arrays

Let’s create a simple HTML and JavaScript project that demonstrates the use of various array methods.

<!DOCTYPE html>

<html>

<head>

<title>Arrays Demo</title>

</head>

<body>

<h1>Arrays Demo</h1>

<p id="output"></p>

<button onclick="showFruits()">Show Fruits</button>

<button onclick="doubleNumbers()">Double Numbers</button>

<button onclick="filterEvenNumbers()">Filter Even Numbers</button>

<button onclick="sumNumbers()">Sum Numbers</button>

<script>

// Array of fruits

const fruits = ["Apple", "Banana", "Cherry", "Date"];

// Array of numbers

const numbers = [1, 2, 3, 4, 5];

// Displaying the array of fruits function showFruits() {

document.getElementById("output").textContent = fruits.join(", ");

}

// Doubling the numbers using map function doubleNumbers() {

const doubled = numbers.map(num => num \* 2); document.getElementById("output").textContent = doubled.join(", ");

}

// Filtering even numbers using filter function filterEvenNumbers() {

const even = numbers.filter(num => num % 2 === 0); document.getElementById("output").textContent = even.join(", ");

}

// Summing the numbers using reduce function sumNumbers() {

const sum = numbers.reduce((total, num) => total + num, 0); document.getElementById("output").textContent = `Sum: ${sum}`;

}

</script>

</body>

</html>

In this project:

* We define arrays of fruits and numbers.
* We use the map method to double the numbers.
* We use the filter method to get even numbers.
* We use the reduce method to sum the numbers.
* We display the results in the HTML content.

### JavaScript String Methods

A **string** in JavaScript is a **sequence of characters** used to represent text. Strings are **enclosed in quotes**: single (' '), double (" "), or backticks (` ` for template literals).

let name = "Gaurab";

let message = 'Welcome!';

let greeting = `Hello, ${name}`;

#### Commonly Used JavaScript Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| length | Returns the length of the string |
| charAt(index) | Returns character at specific index |
| charCodeAt(index) | Returns Unicode of the character |
| concat(str) | Joins two or more strings |
| includes(substr) | Checks if string contains substring |
| startsWith(substr) | Checks if string starts with substring |
| endsWith(substr) | Checks if string ends with substring |
| indexOf(substr) | Returns first index of substring |
| lastIndexOf(substr) | Returns last index of substring |
| toUpperCase() | Converts to uppercase |
| toLowerCase() | Converts to lowercase |
| trim() | Removes whitespace from both ends |
| slice(start, end) | Extracts part of a string |
| substring(start, end) | Similar to slice but no negative index |
| replace(search, replace) | Replaces text |
| repeat(count) | Repeats the string |
| split(separator) | Splits string into array |
| match(regex) | Searches with regex |
| padStart(len, str) | Pads start of string |
| padEnd(len, str) | Pads end of string |

#### Example using string methods

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>JavaScript String Methods Demo</title>

<style>

body { font-family: Arial, sans-serif; padding: 20px; }

input, button { margin: 10px 0; padding: 5px; width: 100%; }

pre { background: #f4f4f4; padding: 10px; border-radius: 5px; white-space: pre-wrap; }

</style>

</head>

<body>

<h2>Enter a String:</h2>

<input type="text" id="inputStr" placeholder="Type something..." />

<button onclick="applyStringMethods()">Show String Methods</button>

<h3>Result:</h3>

<pre id="output"></pre>

<script>

function applyStringMethods() {

const str = document.getElementById("inputStr").value;

const output = document.getElementById("output");

let result = `Original: "${str}"\n\n`;

result += `Length: ${str.length}\n`;

result += `charAt(2): ${str.charAt(2)}\n`;

result += `charCodeAt(2): ${str.charCodeAt(2)}\n`;

result += `concat(" World"): ${str.concat(" World")}\n`;

result += `includes("test"): ${str.includes("test")}\n`;

result += `startsWith("H"): ${str.startsWith("H")}\n`;

result += `endsWith("!"): ${str.endsWith("!")}\n`;

result += `indexOf("a"): ${str.indexOf("a")}\n`;

result += `lastIndexOf("a"): ${str.lastIndexOf("a")}\n`;

result += `toUpperCase(): ${str.toUpperCase()}\n`;

result += `toLowerCase(): ${str.toLowerCase()}\n`;

result += `trim(): "${str.trim()}"\n`;

result += `slice(1, 4): ${str.slice(1, 4)}\n`;

result += `substring(1, 4): ${str.substring(1, 4)}\n`;

result += `replace("a", "@"): ${str.replace("a", "@")}\n`;

result += `repeat(2): ${str.repeat(2)}\n`;

result += `split(" "): ${str.split(" ").join(", ")}\n`;

result += `padStart(10, "\*"): ${str.padStart(10, "\*")}\n`;

result += `padEnd(10, "-"): ${str.padEnd(10, "-")}\n`;

output.textContent = result;

}

</script>

</body>

</html>

### Math and Number Functions

#### Math

console.log(Math.max(5, 9, 3)); // 9

console.log(Math.random()); // Random number between 0-1

console.log(Math.round(4.7)); // 5

#### Number

let num = 123.456;

console.log(num.toFixed(2)); // "123.46"

console.log(Number("45")); // 45

### Date Functions

let now = new Date();

console.log(now.toDateString()); // e.g., "Thu May 22 2025"

console.log(now.getFullYear()); // 2025

### m

Here is the extracted list of JavaScript topics you mentioned:

### JavaScript Core Topics List:

1. Hoisting
2. Closures
3. Callback Functions
4. DOM Tree
5. Document Object Model (DOM)
6. DOM Methods
7. Event Listeners
8. Event Bubbling and Delegation
9. Arrays and Array Methods
10. Regular Expressions
11. Window Objects
12. Error Handling
13. Asynchronous JavaScript

## 1. Hoisting

### Explanation:

Hoisting is JavaScript's default behavior of moving declarations to the top of the current scope (script or function).

### Syntax Example:

console.log(x); // undefined

var x = 5;

### HTML Example:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Hoisting Example</title>

</head>

<body>

<h2>Check the console for hoisting behavior</h2>

<script>

console.log(x); // undefined

var x = 10;

</script>

</body>

</html>

## 2. Closures

### Explanation:

A closure gives you access to an outer function’s scope from an inner function.

### Syntax Example:

function outer() {

let count = 0;

return function inner() {

count++;

console.log(count);

};

}

const counter = outer();

counter();

counter();

### HTML Example:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Closure Example</title>

</head>

<body>

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

<script>

function outer() {

let count = 0;

return function inner() {

count++;

console.log(count);

};

}

const counter = outer();

</script>

</body>

</html>

## 3. Callback Functions

### Explanation:

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

### Syntax Example:

function greet(name, callback) {

console.log("Hello " + name);

callback();

}

function sayBye() {

console.log("Goodbye!");

}

greet("John", sayBye);

### HTML Example:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Callback Example</title>

</head>

<body>

<script>

function greet(name, callback) {

document.write("Hello " + name + "<br>");

callback();

}

function sayBye() {

document.write("Goodbye!");

}

greet("Alice", sayBye);

</script>

</body>

</html>

## 4. DOM Tree

### Explanation:

The DOM Tree represents the structure of a web page as a tree of objects. Each HTML element becomes a node in this tree.

### HTML Example:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>DOM Tree Example</title>

</head>

<body>

<div id="container">

<h1>Title</h1>

<p>This is a paragraph.</p>

</div>

</body>

</html>

This tree includes:

* Root: <html>
* Children: <head>, <body>
* Inside <body>: <div> → <h1> and <p>

In JavaScript, you can traverse this tree using DOM methods.

const container = document.getElementById("container");

console.log(container.children);

1. *ES6 Features*

**ES6** (ECMAScript 2015) introduced many new features to JavaScript.

* + **let and const**: Block-scoped variable declarations.
  + **Arrow Functions**: Shorter syntax for writing functions.
  + **Template Literals**: String literals allowing embedded expressions.
  + **Destructuring Assignment**: Unpacking values from arrays or properties from objects.
  + **Default Parameters**: Setting default values for function parameters.
  + **Rest and Spread Operators**: Collecting and spreading elements.
  + **Classes**: Syntactic sugar for creating objects and dealing with inheritance.
  + **Modules**: Importing and exporting functionalities between files.

#### Advanced Concepts

1. *Closures*

A **closure** is a function that retains access to its lexical scope even when the function is executed outside that scope.

*function outerFunction() {*

*let outerVariable = "I'm outside!";*

*function innerFunction() { console.log(outerVariable);*

*}*

*return innerFunction;*

*}*

*const myFunction = outerFunction(); myFunction(); // Output: I'm outside!*

1. *Promises and Async/Await*

**Promises** are used to handle asynchronous operations.

*let promise = new Promise((resolve, reject) => { let success = true;*

*if (success) {*

*resolve("Operation was successful!");*

*} else {*

*reject("Operation failed!");*

*}*

*});*

*promise.then((message) => {*

*console.log(message);*

*}).catch((error) => { console.log(error);*

*});*

**Async/Await** is syntactic sugar for working with promises.

*async function fetchData() { try {*

*let response = await fetch('https://api.example.com/data'); let data = await response.json();*

*console.log(data);*

*} catch (error) { console.log(error);*

*}*

*}*

1. *Prototypes and Inheritance*

**Prototypes** allow you to add properties and methods to objects.

*function Person(name, age) { this.name = name; this.age = age;*

*}*

*Person.prototype.greet = function() { return `Hello, my name is ${this.name}`;*

*};*

*let john = new Person("John", 30); console.log(john.greet()); // Output: Hello, my name is John*

1. *Modules*

**Modules** allow you to organize your code into separate files.

*// In file math.js*

*export function add(a, b) { return a + b;*

*}*

*// In another file*

*import { add } from './math.js'; console.log(add(2, 3)); // Output: 5*

#### Visual Explanation

Here's a simple visual representation of how JavaScript interacts with HTML and CSS:

I'll try to create that.

#### Manipulating the DOM

JavaScript can manipulate the DOM (Document Object Model) to update the content and style of a web page dynamically. The DOM represents the structure of a web page as a tree of objects.

*Example: Changing the Content of an Element*

*<!DOCTYPE html>*

*<html>*

*<head>*

*<title>DOM Manipulation</title>*

*</head>*

*<body>*

*<h1 id="title">Original Title</h1>*

*<button onclick="changeTitle()">Change Title</button>*

*<script>*

*function changeTitle() { document.getElementById("title").textContent = "New Title";*

*}*

*</script>*

*</body>*

*</html>*

In this example, clicking the button changes the text of the <h1> element from "Original Title" to "New Title".

*Example: Changing the Style of an Element*

*<!DOCTYPE html>*

*<html>*

*<head>*

*<title>DOM Manipulation</title>*

*</head>*

*<body>*

*<h1 id="title">Styled Title</h1>*

*<button onclick="changeStyle()">Change Style</button>*

*<script>*

*function changeStyle() { document.getElementById("title").style.color = "red"; document.getElementById("title").style.fontSize = "24px";*

*}*

*</script>*

*</body>*

*</html>*

In this example, clicking the button changes the color and font size of the <h1> element.

Sure! Since ECMAScript (ES) evolves continuously, there have been several versions released after ES6 (ECMAScript 2015). Here’s a detailed look at the new features introduced in subsequent versions, along with examples in HTML and JavaScript for each topic.

### ECMAScript 2016 (ES7)

\*\*1. Exponentiation Operator (`\*\*`)\*\*

* Provides a shorthand for exponentiation.
* Example: `2 \*\* 3` is equivalent to `Math.pow(2, 3)`.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ES7 Example</title>

</head>

<body>

<script>

const base = 2; const exponent = 3;

const result = base \*\* exponent; console.log(result); // Outputs: 8

</script>

</body>

</html>

```

\*\*2. Array.prototype.includes\*\*

* Checks if an array includes a certain value.
* Example: `array.includes(value)` returns `true` or `false`.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ES7 Example</title>

</head>

<body>

<script>

const fruits = ['apple', 'banana', 'mango']; console.log(fruits.includes('banana')); // Outputs: true console.log(fruits.includes('grape')); // Outputs: false

</script>

</body>

</html>

```

### ECMAScript 2017 (ES8)

\*\*1. Async/Await\*\*

* Provides a way to work with asynchronous code in a more synchronous style.
* Example: `async` function with `await` keyword.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ES8 Example</title>

</head>

<body>

<script>

async function fetchData() {

const response = await fetch('https://jsonplaceholder.typicode.com/posts/1'); const data = await response.json();

console.log(data);

}

fetchData();

</script>

</body>

</html>

```

\*\*2. Object.entries() and Object.values()\*\*

* `Object.entries(obj)` returns an array of `[key, value]` pairs.
* `Object.values(obj)` returns an array of values from the object.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ES8 Example</title>

</head>

<body>

<script>

const obj = { name: 'John', age: 30 };

console.log(Object.entries(obj)); // Outputs: [ ['name', 'John'], ['age', 30] ] console.log(Object.values(obj)); // Outputs: [ 'John', 30 ]

</script>

</body>

</html>

```

\*\*3. String Padding\*\*

* `String.prototype.padStart()` and `String.prototype.padEnd()` for padding strings.
* Example: Pads the start or end of a string to a given length.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ES8 Example</title>

</head>

<body>

<script>

const str = '5';

console.log(str.padStart(4, '0')); // Outputs: '0005'

console.log(str.padEnd(4, '0')); // Outputs: '5000'

</script>

</body>

</html>

```

### ECMAScript 2018 (ES9)

\*\*1. Asynchronous Iteration\*\*

* Allows `for-await-of` loops to work with asynchronous iterables.
* Example: Iterating over async data sources.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ES9 Example</title>

</head>

<body>

<script>

async function\* fetchNumbers() { for (let i = 1; i <= 3; i++) {

await new Promise(resolve => setTimeout(resolve, 1000));

yield i;

}

}

(async () => {

for await (const num of fetchNumbers()) { console.log(num); // Outputs: 1, 2, 3 (each after 1 second)

}

})();

</script>

</body>

</html>

```

\*\*2. Rest/Spread Properties\*\*

* Rest properties for objects allow collecting remaining properties.
* Spread properties for objects allow spreading properties into a new object.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ES9 Example</title>

</head>

<body>

<script>

const { a, b, ...rest } = { a: 1, b: 2, c: 3, d: 4 };

console.log(a); // Outputs: 1 console.log(b); // Outputs: 2 console.log(rest); // Outputs: { c: 3, d: 4 }

const obj1 = { a: 1, b: 2 }; const obj2 = { c: 3, d: 4 };

const combined = { ...obj1, ...obj2 }; console.log(combined); // Outputs: { a: 1, b: 2, c: 3, d: 4 }

</script>

</body>

</html>

```

### ECMAScript 2019 (ES10)

\*\*1. Array.prototype.flat() and Array.prototype.flatMap()\*\*

* `Array.prototype.flat()` flattens nested arrays.
* `Array.prototype.flatMap()` maps and then flattens arrays.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ES10 Example</title>

</head>

<body>

<script>

const nestedArray = [1, [2, [3, 4], 5], 6];

console.log(nestedArray.flat(2)); // Outputs: [1, 2, 3, 4, 5, 6]

const arr = [1, 2, 3, 4];

const doubled = arr.flatMap(x => [x, x \* 2]); console.log(doubled); // Outputs: [1, 2, 2, 4, 3, 6, 4, 8]

</script>

</body>

</html>

```

\*\*2. Object.fromEntries()\*\*

* Converts an array of key-value pairs into an object.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ES10 Example</title>

</head>

<body>

<script>

const entries = [['name', 'John'], ['age', 30]]; const obj = Object.fromEntries(entries);

console.log(obj); // Outputs: { name: 'John', age: 30 }

</script>

</body>

</html>

```

\*\*3. Optional Catch Binding\*\*

* Allows omission of the error parameter in `catch` blocks.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ES10 Example</title>

</head>

<body>

<script> try {

throw new Error('Something went wrong');

} catch {

console.log('An error occurred');

}

</script>

</body>

</html>

```

### ECMAScript 2020 (ES11)

\*\*1. Nullish Coalescing Operator (`??`)\*\*

* Provides a way to handle `null` and `undefined` values.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport"