# JavaScript Basics Reference Guide

A comprehensive reference for fundamental JavaScript concepts and syntax.

## Table of Contents



## 1. Introduction to JavaScript

JavaScript is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions[1]. It is best known as the scripting language for web pages, but it's also used in many non-browser environments such as Node.js[2].

### Key Characteristics

- **Dynamically typed**: Variables can hold any data type
- **Single-threaded**: Executes one operation at a time
- **Event-driven**: Responds to user interactions and system events
- **Cross-platform**: Runs on multiple platforms and browsers

### Hello World Example

```
// Output to console
console.log("Hello, World!");

// Display in HTML document
document.write("Hello, World!");

// Alert box
alert("Hello, World!");
```

## 2. Variables and Data Types

### Variable Declaration

JavaScript provides three ways to declare variables[3]:

```
// var - function scoped (older way)
var name = "John";

// let - block scoped (modern, mutable)
let age = 25;

// const - block scoped (modern, immutable)
const PI = 3.14159;
```

### Data Types

JavaScript has **7 primitive data types** and **1 reference type**[4]:

| Type | Example | Description |
|---|---|---|
| String | "Hello" or 'Hello' | Text data enclosed in quotes |
| Number | 42 or 3.14 | Integers and decimals |
| Boolean | true or false | Logical values |
| Undefined | undefined | Variable declared but not assigned |
| Null | null | Intentional absence of value |
| Symbol | Symbol("id") | Unique identifier |
| BigInt | 123n | Large integers beyond Number limit |
| Object | {name: "John"} | Collection of key-value pairs |

Table 1: JavaScript Data Types

### Type Checking

```
typeof "Hello" // "string"
typeof 42 // "number"
typeof true // "boolean"
typeof undefined // "undefined"
typeof null // "object" (known quirk)
typeof {name: "John"} // "object"
typeof [1, 2, 3] // "object"
```

# 3. Operators

## Arithmetic Operators

```
let a = 10, b = 3;

console.log(a + b); // Addition: 13
console.log(a - b); // Subtraction: 7
console.log(a * b); // Multiplication: 30
console.log(a / b); // Division: 3.333...
console.log(a % b); // Modulus (remainder): 1
console.log(a ** b); // Exponentiation: 1000
```

## Assignment Operators

```
let x = 5;

x += 3; // x = x + 3 → 8
x -= 2; // x = x - 2 → 6
x *= 4; // x = x * 4 → 24
x /= 3; // x = x / 3 → 8
x %= 5; // x = x % 5 → 3
```

## Comparison Operators

| Operator | Name | Description |
|---|---|---|
| == | Equality | Compares values (type coercion) |
| === | Strict Equality | Compares values and types |
| != | Inequality | Not equal (type coercion) |
| !== | Strict Inequality | Not equal (no type coercion) |
| > | Greater than | Left operand greater than right |
| < | Less than | Left operand less than right |
| >= | Greater or equal | Left greater than or equal to right |
| <= | Less or equal | Left less than or equal to right |

Table 2: Comparison Operators

```
5 == "5" // true (type coercion)
5 === "5" // false (different types)
10 > 5 // true
10 <= 10 // true
```

### Logical Operators

```
// AND (&&) - true if both operands are true
true && true // true
true && false // false

// OR (||) - true if at least one operand is true
true || false // true
false || false // false

// NOT (!) - inverts the boolean value
!true // false
!false // true
```

### Increment and Decrement

```
let count = 5;

count++; // Post-increment: count = 6
count--; // Post-decrement: count = 5
++count; // Pre-increment: count = 6
--count; // Pre-decrement: count = 5
```

## 4. Control Flow Statements

### If-Else Statements

```
let age = 18;

if (age >= 18) {
console.log("You are an adult");
} else if (age >= 13) {
console.log("You are a teenager");
} else {
console.log("You are a child");
}
```

### Ternary Operator

```
let status = (age >= 18) ? "adult" : "minor";
```

### Switch Statement

```
let day = 3;
let dayName;

switch (day) {
case 1:
dayName = "Monday";
break;
case 2:
dayName = "Tuesday";
break;
```

```
case 3:
dayName = "Wednesday";
break;
default:
dayName = "Invalid day";
}
```

### Loops

#### For Loop

```
// Standard for loop
for (let i = 0; i < 5; i++) {
console.log(i); // 0, 1, 2, 3, 4
}

// For...of loop (arrays)
let fruits = ["apple", "banana", "orange"];
for (let fruit of fruits) {
console.log(fruit);
}

// For...in loop (objects)
let person = {name: "John", age: 30};
for (let key in person) {
console.log(key + ": " + person[key]);
}
```

#### While Loop

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

#### Do-While Loop

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

## 5. Functions

### Function Declaration

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

console.log(greet("Alice")); // "Hello, Alice!"
```

### Function Expression

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

### Arrow Functions (ES6)

```
// Single parameter, single expression
const square = x => x * x;

// Multiple parameters
const add = (a, b) => a + b;

// Multiple statements
const greetUser = (name) => {
let greeting = "Hello, " + name;
return greeting + "!";
};
```

### Default Parameters

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

console.log(greet()); // "Hello, Guest"
console.log(greet("John")); // "Hello, John"
```

### Rest Parameters

```
function sum(...numbers) {
return numbers.reduce((total, num) => total + num, 0);
}

console.log(sum(1, 2, 3, 4)); // 10
```

## 6. Arrays

### Creating Arrays

```
// Array literal
let fruits = ["apple", "banana", "orange"];

// Array constructor
let numbers = new Array(1, 2, 3, 4, 5);

// Empty array
let empty = [];
```

## Array Methods

| Method | Description |
|---|---|
| push() | Adds elements to the end |
| pop() | Removes and returns last element |
| shift() | Removes and returns first element |
| unshift() | Adds elements to the beginning |
| slice() | Returns a shallow copy of a portion |
| splice() | Adds/removes elements at any position |
| concat() | Merges two or more arrays |
| indexOf() | Returns first index of element |
| includes() | Checks if array contains element |
| join() | Joins all elements into a string |

Table 3: Common Array Methods

```
let fruits = ["apple", "banana"];

// Add elements
fruits.push("orange"); // ["apple", "banana", "orange"]
fruits.unshift("mango"); // ["mango", "apple", "banana", "orange"]

// Remove elements
fruits.pop(); // ["mango", "apple", "banana"]
fruits.shift(); // ["apple", "banana"]

// Access elements
console.log(fruits[0]); // "apple"
console.log(fruits.length); // 2
```

## Array Iteration Methods

```
let numbers = [1, 2, 3, 4, 5];

// forEach - execute function for each element
numbers.forEach(num => console.log(num));

// map - create new array with transformed elements
let doubled = numbers.map(num => num * 2);
// [2, 4, 6, 8, 10]

// filter - create new array with elements that pass test
let evens = numbers.filter(num => num % 2 === 0);
// [2, 4]

// reduce - reduce array to single value
let sum = numbers.reduce((total, num) => total + num, 0);
// 15
```

```
// find - return first element that satisfies condition
let found = numbers.find(num => num > 3);
// 4

// some - check if at least one element passes test
let hasEven = numbers.some(num => num % 2 === 0);
// true

// every - check if all elements pass test
let allPositive = numbers.every(num => num > 0);
// true
```

# 7. Objects

## Creating Objects

```
// Object literal
let person = {
firstName: "John",
lastName: "Doe",
age: 30,
isEmployed: true
};

// Constructor function
function Person(firstName, lastName, age) {
this.firstName = firstName;
this.lastName = lastName;
this.age = age;
}

let user = new Person("Jane", "Smith", 25);
```

## Accessing Properties

```
// Dot notation
console.log(person.firstName); // "John"

// Bracket notation
console.log(person["lastName"]); // "Doe"

// Dynamic property access
let prop = "age";
console.log(person[prop]); // 30
```

## Object Methods

```
let calculator = {
a: 10,
b: 5,
add: function() {
return this.a + this.b;
},
```

```
subtract() { // ES6 shorthand
return this.a - this.b;
}
};

console.log(calculator.add()); // 15
console.log(calculator.subtract()); // 5
```

### Object Destructuring (ES6)

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

// Extract properties
let {name, age, city} = person;
console.log(name); // "John"
console.log(age); // 30
```

### Spread Operator (ES6)

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

// Merge objects
let merged = {...obj1, ...obj2};
// {a: 1, b: 2, c: 3, d: 4}

// Clone object
let clone = {...obj1};
```

## 8. DOM Manipulation

The Document Object Model (DOM) is a programming interface for HTML documents[5]. JavaScript can manipulate the DOM to dynamically change content, structure, and styling.

### Selecting Elements

```
// Select by ID
let element = document.getElementById("myId");

// Select by class name
let elements = document.getElementsByClassName("myClass");

// Select by tag name
let paragraphs = document.getElementsByTagName("p");

// Query selector (CSS selector)
let item = document.querySelector(".myClass");
let items = document.querySelectorAll(".myClass");
```

### Modifying Content

```
// Change text content
element.textContent = "New text";

// Change HTML content
element.innerHTML = "<b>Bold text</b>";

// Change attributes
element.setAttribute("class", "newClass");
element.src = "image.jpg";

// Change styles
element.style.color = "red";
element.style.fontSize = "20px";
```

### Creating and Adding Elements

```
// Create new element
let newDiv = document.createElement("div");
newDiv.textContent = "Hello";
newDiv.className = "myClass";

// Add to document
document.body.appendChild(newDiv);

// Insert before another element
let reference = document.getElementById("existing");
document.body.insertBefore(newDiv, reference);
```

### Removing Elements

```
// Remove element
let element = document.getElementById("myId");
element.remove();

// Remove child element
let parent = document.getElementById("parent");
let child = document.getElementById("child");
parent.removeChild(child);
```

## 9. Events

Events are actions or occurrences that happen in the browser[6]. JavaScript can respond to these events.

### Event Listeners

```
// Add event listener
let button = document.getElementById("myButton");

button.addEventListener("click", function() {
console.log("Button clicked!");
});
```

```
// Arrow function syntax
button.addEventListener("click", () => {
console.log("Button clicked!");
});

// Named function
function handleClick() {
console.log("Button clicked!");
}
button.addEventListener("click", handleClick);
```

### Common Event Types

| Event | Description |
|---|---|
| click | Mouse click on element |
| dblclick | Mouse double-click |
| mouseover | Mouse pointer moves over element |
| mouseout | Mouse pointer moves out of element |
| keydown | Key is pressed down |
| keyup | Key is released |
| submit | Form is submitted |
| change | Form element value changes |
| focus | Element receives focus |
| blur | Element loses focus |
| load | Page finishes loading |

Table 4: Common DOM Events

### Event Object

```
button.addEventListener("click", function(event) {
console.log(event.target); // Element that triggered event
console.log(event.type); // Event type ("click")
console.log(event.clientX); // Mouse X coordinate
console.log(event.clientY); // Mouse Y coordinate

    event.preventDefault();      // Prevent default action
    event.stopPropagation();     // Stop event bubbling

});
```

### Event Delegation

```
// Add listener to parent instead of each child
let list = document.getElementById("myList");

list.addEventListener("click", function(event) {
if (event.target.tagName === "LI") {
console.log("List item clicked:", event.target.textContent);
}
});
```

## 10. Modern JavaScript Features

### Template Literals (ES6)

```
let name = "John";
let age = 30;

// String interpolation
let message = Hello, my name is ${name} and I am ${age} years old.;

// Multi-line strings
let multiline = This is a multi-line string.;
```

### Destructuring

```
// Array destructuring
let [first, second, third] = [1, 2, 3];

// Skip elements
let [a, , c] = [1, 2, 3];

// Rest pattern
let [head, ...tail] = [1, 2, 3, 4, 5];
// head = 1, tail = [2, 3, 4, 5]
```

### Classes (ES6)

```
class Person {
constructor(name, age) {
this.name = name;
this.age = age;
}
```

```
  greet() {
    return `Hello, I'm ${this.name}`;
  }

  static info() {
```

```
    return "This is the Person class";
  }
```

}

let john = new Person("John", 30);
console.log(john.greet()); // "Hello, I'm John"

### Promises

// Create promise
let promise = new Promise((resolve, reject) => {
setTimeout(() => {
resolve("Success!");
}, 1000);
});

// Use promise
promise
.then(result => console.log(result))
.catch(error => console.error(error));

### Async/Await

async function fetchData() {
try {
let response = await fetch("https://api.example.com/data");
let data = await response.json();
console.log(data);
} catch (error) {
console.error("Error:", error);
}
}

### Modules (ES6)

// export.js
export const PI = 3.14159;
export function add(a, b) {
return a + b;
}

// import.js
import { PI, add } from './export.js';
console.log(PI);
console.log(add(2, 3));

## Best Practices

1. **Use strict mode**: Add 'use strict'; at the beginning of scripts
2. **Use const and let**: Avoid var, prefer const for immutable values
3. **Use meaningful variable names**: Use camelCase for variables and functions
4. **Comment your code**: Write clear comments for complex logic
5. **Handle errors**: Use try-catch blocks for error-prone code
6. **Validate user input**: Always validate and sanitize user input
7. **Keep functions small**: Each function should do one thing well
8. **Use arrow functions**: For shorter syntax and lexical this
9. **Avoid global variables**: Minimize pollution of global scope
10. **Use modern features**: Leverage ES6+ features for cleaner code

## Common Pitfalls to Avoid

- **Type coercion confusion**: Use === instead of ==
- **Forgetting this context**: Be careful with this in callbacks
- **Modifying arrays during iteration**: Create a copy before modifying
- **Not handling async operations**: Use promises or async/await properly
- **Blocking the event loop**: Avoid long-running synchronous operations
- **Memory leaks**: Remove event listeners when no longer needed
- **Ignoring error handling**: Always handle potential errors

## Quick Reference Summary

### Variable Declaration

let x = 10; // Block-scoped, mutable
const y = 20; // Block-scoped, immutable

### Data Types

"string", 42, true, undefined, null, {}, []

### Functions

function name() {}
const name = () => {}

### Arrays

[1, 2, 3].map(), .filter(), .reduce()

### Objects

{key: value}, obj.key, obj["key"]

### Conditionals

```
if (condition) {} else {}
condition ? true : false
```

### Loops

```
for (let i = 0; i < n; i++) {}
while (condition) {}
array.forEach(item => {})
```

### DOM

```
document.querySelector()
element.addEventListener()
element.textContent
```

## Conclusion

This reference guide covers the fundamental concepts of JavaScript that every developer should know[7]. JavaScript continues to evolve with new features added regularly through the ECMAScript standard[8]. To deepen your understanding, practice building projects, read documentation, and explore advanced topics like asynchronous programming, design patterns, and modern frameworks[9].

## References


[1] Mozilla Developer Network. (2024). JavaScript - MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript

[2] W3Schools. (2024). JavaScript Tutorial. https://www.w3schools.com/js/

[3] JavaScript.info. (2024). The Modern JavaScript Tutorial - Variables. https://javascript.info/variables

[4] W3Schools. (2024). JavaScript Data Types. https://www.w3schools.com/js/js_datatypes.asp

[5] Mozilla Developer Network. (2024). Introduction to the DOM. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

[6] W3Schools. (2024). JavaScript Events. https://www.w3schools.com/js/js_events.asp

[7] GeeksforGeeks. (2023). JavaScript Tutorial. https://www.geeksforgeeks.org/javascript/

[8] JavaScript Tutorial. (2024). JavaScript Tutorial - Getting Started and Fundamentals. https://www.javascripttutorial.net

[9] JetBrains. (2024). How to Learn JavaScript: Your Step-by-Step Guide for Beginners. https://blog.jetbrains.com/webstorm/2024/07/how-to-learn-javascript/

# JavaScript Basics Reference Guide

A comprehensive reference for fundamental JavaScript concepts and syntax.

## Table of Contents



## 1. Introduction to JavaScript

JavaScript is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions[1]. It is best known as the scripting language for web pages, but it's also used in many non-browser environments such as Node.js[2].

### Key Characteristics

- **Dynamically typed**: Variables can hold any data type
- **Single-threaded**: Executes one operation at a time
- **Event-driven**: Responds to user interactions and system events
- **Cross-platform**: Runs on multiple platforms and browsers

### Hello World Example

```
// Output to console
console.log("Hello, World!");

// Display in HTML document
document.write("Hello, World!");

// Alert box
alert("Hello, World!");
```

## 2. Variables and Data Types

### Variable Declaration

JavaScript provides three ways to declare variables[3]:

```
// var - function scoped (older way)
var name = "John";
```

```
// let - block scoped (modern, mutable)
let age = 25;

// const - block scoped (modern, immutable)
const PI = 3.14159;
```

### Data Types

JavaScript has **7 primitive data types** and **1 reference type**[4]:

| Type | Example | Description |
|---|---|---|
| String | "Hello" or 'Hello' | Text data enclosed in quotes |
| Number | 42 or 3.14 | Integers and decimals |
| Boolean | true or false | Logical values |
| Undefined | undefined | Variable declared but not assigned |
| Null | null | Intentional absence of value |
| Symbol | Symbol("id") | Unique identifier |
| BigInt | 123n | Large integers beyond Number limit |
| Object | {name: "John"} | Collection of key-value pairs |

Table 5: JavaScript Data Types

### Type Checking

```
typeof "Hello" // "string"
typeof 42 // "number"
typeof true // "boolean"
typeof undefined // "undefined"
typeof null // "object" (known quirk)
typeof {name: "John"} // "object"
typeof [1, 2, 3] // "object"
```

## 3. Operators

### Arithmetic Operators

```
let a = 10, b = 3;

console.log(a + b); // Addition: 13
console.log(a - b); // Subtraction: 7
console.log(a * b); // Multiplication: 30
console.log(a / b); // Division: 3.333...
```

```
console.log(a % b); // Modulus (remainder): 1
console.log(a ** b); // Exponentiation: 1000
```

### Assignment Operators

```
let x = 5;

x += 3; // x = x + 3 → 8
x -= 2; // x = x - 2 → 6
x *= 4; // x = x * 4 → 24
x /= 3; // x = x / 3 → 8
x %= 5; // x = x % 5 → 3
```

### Comparison Operators

| Operator | Name | Description |
|---|---|---|
| == | Equality | Compares values (type coercion) |
| === | Strict Equality | Compares values and types |
| != | Inequality | Not equal (type coercion) |
| !== | Strict Inequality | Not equal (no type coercion) |
| > | Greater than | Left operand greater than right |
| < | Less than | Left operand less than right |
| >= | Greater or equal | Left greater than or equal to right |
| <= | Less or equal | Left less than or equal to right |

Table 6: Comparison Operators

```
5 == "5" // true (type coercion)
5 === "5" // false (different types)
10 > 5 // true
10 <= 10 // true
```

### Logical Operators

```
// AND (&&) - true if both operands are true
true && true // true
true && false // false

// OR (||) - true if at least one operand is true
true || false // true
false || false // false

// NOT (!) - inverts the boolean value
!true // false
!false // true
```

### Increment and Decrement

```
let count = 5;

count++; // Post-increment: count = 6
count--; // Post-decrement: count = 5
++count; // Pre-increment: count = 6
--count; // Pre-decrement: count = 5
```

## 4. Control Flow Statements

### If-Else Statements

```
let age = 18;

if (age >= 18) {
console.log("You are an adult");
} else if (age >= 13) {
console.log("You are a teenager");
} else {
console.log("You are a child");
}
```

### Ternary Operator

```
let status = (age >= 18) ? "adult" : "minor";
```

### Switch Statement

```
let day = 3;
let dayName;

switch (day) {
case 1:
dayName = "Monday";
break;
case 2:
dayName = "Tuesday";
break;
case 3:
dayName = "Wednesday";
break;
default:
dayName = "Invalid day";
}
```

### Loops

### For Loop

```
// Standard for loop
for (let i = 0; i < 5; i++) {
console.log(i); // 0, 1, 2, 3, 4
}

// For...of loop (arrays)
let fruits = ["apple", "banana", "orange"];
for (let fruit of fruits) {
console.log(fruit);
}

// For...in loop (objects)
let person = {name: "John", age: 30};
for (let key in person) {
console.log(key + ": " + person[key]);
}
```

### While Loop

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

### Do-While Loop

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

# 5. Functions

## Function Declaration

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

console.log(greet("Alice")); // "Hello, Alice!"
```

## Function Expression

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

### Arrow Functions (ES6)

```
// Single parameter, single expression
const square = x => x * x;

// Multiple parameters
const add = (a, b) => a + b;

// Multiple statements
const greetUser = (name) => {
let greeting = "Hello, " + name;
return greeting + "!";
};
```

### Default Parameters

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

console.log(greet()); // "Hello, Guest"
console.log(greet("John")); // "Hello, John"
```

### Rest Parameters

```
function sum(...numbers) {
return numbers.reduce((total, num) => total + num, 0);
}

console.log(sum(1, 2, 3, 4)); // 10
```

## 6. Arrays

### Creating Arrays

```
// Array literal
let fruits = ["apple", "banana", "orange"];

// Array constructor
let numbers = new Array(1, 2, 3, 4, 5);

// Empty array
let empty = [];
```

### Array Methods

| Method | Description |
|---|---|
| push() | Adds elements to the end |
| pop() | Removes and returns last element |
| shift() | Removes and returns first element |
| unshift() | Adds elements to the beginning |
| slice() | Returns a shallow copy of a portion |
| splice() | Adds/removes elements at any position |
| concat() | Merges two or more arrays |
| indexOf() | Returns first index of element |
| includes() | Checks if array contains element |
| join() | Joins all elements into a string |

Table 7: Common Array Methods

```
let fruits = ["apple", "banana"];

// Add elements
fruits.push("orange"); // ["apple", "banana", "orange"]
fruits.unshift("mango"); // ["mango", "apple", "banana", "orange"]

// Remove elements
fruits.pop(); // ["mango", "apple", "banana"]
fruits.shift(); // ["apple", "banana"]

// Access elements
console.log(fruits[0]); // "apple"
console.log(fruits.length); // 2
```

### Array Iteration Methods

```
let numbers = [1, 2, 3, 4, 5];

// forEach - execute function for each element
numbers.forEach(num => console.log(num));

// map - create new array with transformed elements
let doubled = numbers.map(num => num * 2);
// [2, 4, 6, 8, 10]

// filter - create new array with elements that pass test
let evens = numbers.filter(num => num % 2 === 0);
// [2, 4]

// reduce - reduce array to single value
let sum = numbers.reduce((total, num) => total + num, 0);
// 15
```

```
// find - return first element that satisfies condition
let found = numbers.find(num => num > 3);
// 4

// some - check if at least one element passes test
let hasEven = numbers.some(num => num % 2 === 0);
// true

// every - check if all elements pass test
let allPositive = numbers.every(num => num > 0);
// true
```

# 7. Objects

## Creating Objects

```
// Object literal
let person = {
firstName: "John",
lastName: "Doe",
age: 30,
isEmployed: true
};

// Constructor function
function Person(firstName, lastName, age) {
this.firstName = firstName;
this.lastName = lastName;
this.age = age;
}

let user = new Person("Jane", "Smith", 25);
```

## Accessing Properties

```
// Dot notation
console.log(person.firstName); // "John"

// Bracket notation
console.log(person["lastName"]); // "Doe"

// Dynamic property access
let prop = "age";
console.log(person[prop]); // 30
```

## Object Methods

```
let calculator = {
a: 10,
b: 5,
add: function() {
return this.a + this.b;
},
```

```
subtract() { // ES6 shorthand
return this.a - this.b;
}
};

console.log(calculator.add()); // 15
console.log(calculator.subtract()); // 5
```

### Object Destructuring (ES6)

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

// Extract properties
let {name, age, city} = person;
console.log(name); // "John"
console.log(age); // 30
```

### Spread Operator (ES6)

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

// Merge objects
let merged = {...obj1, ...obj2};
// {a: 1, b: 2, c: 3, d: 4}

// Clone object
let clone = {...obj1};
```

## 8. DOM Manipulation

The Document Object Model (DOM) is a programming interface for HTML documents[5]. JavaScript can manipulate the DOM to dynamically change content, structure, and styling.

### Selecting Elements

```
// Select by ID
let element = document.getElementById("myId");

// Select by class name
let elements = document.getElementsByClassName("myClass");

// Select by tag name
let paragraphs = document.getElementsByTagName("p");

// Query selector (CSS selector)
let item = document.querySelector(".myClass");
let items = document.querySelectorAll(".myClass");
```

### Modifying Content

```
// Change text content
element.textContent = "New text";

// Change HTML content
element.innerHTML = "<b>Bold text</b>";

// Change attributes
element.setAttribute("class", "newClass");
element.src = "image.jpg";

// Change styles
element.style.color = "red";
element.style.fontSize = "20px";
```

### Creating and Adding Elements

```
// Create new element
let newDiv = document.createElement("div");
newDiv.textContent = "Hello";
newDiv.className = "myClass";

// Add to document
document.body.appendChild(newDiv);

// Insert before another element
let reference = document.getElementById("existing");
document.body.insertBefore(newDiv, reference);
```

### Removing Elements

```
// Remove element
let element = document.getElementById("myId");
element.remove();

// Remove child element
let parent = document.getElementById("parent");
let child = document.getElementById("child");
parent.removeChild(child);
```

## 9. Events

Events are actions or occurrences that happen in the browser[6]. JavaScript can respond to these events.

### Event Listeners

```
// Add event listener
let button = document.getElementById("myButton");

button.addEventListener("click", function() {
console.log("Button clicked!");
});
```

```
// Arrow function syntax
button.addEventListener("click", () => {
console.log("Button clicked!");
});

// Named function
function handleClick() {
console.log("Button clicked!");
}
button.addEventListener("click", handleClick);
```

### Common Event Types

| Event | Description |
|---|---|
| click | Mouse click on element |
| dblclick | Mouse double-click |
| mouseover | Mouse pointer moves over element |
| mouseout | Mouse pointer moves out of element |
| keydown | Key is pressed down |
| keyup | Key is released |
| submit | Form is submitted |
| change | Form element value changes |
| focus | Element receives focus |
| blur | Element loses focus |
| load | Page finishes loading |

Table 8: Common DOM Events

### Event Object

```
button.addEventListener("click", function(event) {
console.log(event.target); // Element that triggered event
console.log(event.type); // Event type ("click")
console.log(event.clientX); // Mouse X coordinate
console.log(event.clientY); // Mouse Y coordinate

    event.preventDefault();      // Prevent default action
    event.stopPropagation();     // Stop event bubbling

});
```

### Event Delegation

```
// Add listener to parent instead of each child
let list = document.getElementById("myList");

list.addEventListener("click", function(event) {
if (event.target.tagName === "LI") {
console.log("List item clicked:", event.target.textContent);
}
});
```

## 10. Modern JavaScript Features

### Template Literals (ES6)

```
let name = "John";
let age = 30;

// String interpolation
let message = Hello, my name is ${name} and I am ${age} years old.;

// Multi-line strings
let multiline = This is a multi-line string.;
```

### Destructuring

```
// Array destructuring
let [first, second, third] = [1, 2, 3];

// Skip elements
let [a, , c] = [1, 2, 3];

// Rest pattern
let [head, ...tail] = [1, 2, 3, 4, 5];
// head = 1, tail = [2, 3, 4, 5]
```

### Classes (ES6)

```
class Person {
constructor(name, age) {
this.name = name;
this.age = age;
}
```

```
  greet() {
    return `Hello, I'm ${this.name}`;
  }

  static info() {
```

```
    return "This is the Person class";
  }
```

}

let john = new Person("John", 30);
console.log(john.greet()); // "Hello, I'm John"

### Promises

// Create promise
let promise = new Promise((resolve, reject) => {
setTimeout(() => {
resolve("Success!");
}, 1000);
});

// Use promise
promise
.then(result => console.log(result))
.catch(error => console.error(error));

### Async/Await

async function fetchData() {
try {
let response = await fetch("https://api.example.com/data");
let data = await response.json();
console.log(data);
} catch (error) {
console.error("Error:", error);
}
}

### Modules (ES6)

// export.js
export const PI = 3.14159;
export function add(a, b) {
return a + b;
}

// import.js
import { PI, add } from './export.js';
console.log(PI);
console.log(add(2, 3));

## Best Practices

1. **Use strict mode**: Add 'use strict'; at the beginning of scripts
2. **Use const and let**: Avoid var, prefer const for immutable values
3. **Use meaningful variable names**: Use camelCase for variables and functions
4. **Comment your code**: Write clear comments for complex logic
5. **Handle errors**: Use try-catch blocks for error-prone code
6. **Validate user input**: Always validate and sanitize user input
7. **Keep functions small**: Each function should do one thing well
8. **Use arrow functions**: For shorter syntax and lexical this
9. **Avoid global variables**: Minimize pollution of global scope
10. **Use modern features**: Leverage ES6+ features for cleaner code

## Common Pitfalls to Avoid

- **Type coercion confusion**: Use === instead of ==
- **Forgetting this context**: Be careful with this in callbacks
- **Modifying arrays during iteration**: Create a copy before modifying
- **Not handling async operations**: Use promises or async/await properly
- **Blocking the event loop**: Avoid long-running synchronous operations
- **Memory leaks**: Remove event listeners when no longer needed
- **Ignoring error handling**: Always handle potential errors

## Quick Reference Summary

### Variable Declaration

let x = 10; // Block-scoped, mutable
const y = 20; // Block-scoped, immutable

### Data Types

"string", 42, true, undefined, null, {}, []

### Functions

function name() {}
const name = () => {}

### Arrays

[1, 2, 3].map(), .filter(), .reduce()

### Objects

{key: value}, obj.key, obj["key"]

### Conditionals

```
if (condition) {} else {}
condition ? true : false
```

### Loops

```
for (let i = 0; i < n; i++) {}
while (condition) {}
array.forEach(item => {})
```

### DOM

```
document.querySelector()
element.addEventListener()
element.textContent
```

## Conclusion

This reference guide covers the fundamental concepts of JavaScript that every developer should know[7]. JavaScript continues to evolve with new features added regularly through the ECMAScript standard[8]. To deepen your understanding, practice building projects, read documentation, and explore advanced topics like asynchronous programming, design patterns, and modern frameworks[9].

## References


[1] Mozilla Developer Network. (2024). JavaScript - MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript

[2] W3Schools. (2024). JavaScript Tutorial. https://www.w3schools.com/js/

[3] JavaScript.info. (2024). The Modern JavaScript Tutorial - Variables. https://javascript.info/variables

[4] W3Schools. (2024). JavaScript Data Types. https://www.w3schools.com/js/js_datatypes.asp

[5] Mozilla Developer Network. (2024). Introduction to the DOM. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

[6] W3Schools. (2024). JavaScript Events. https://www.w3schools.com/js/js_events.asp

[7] GeeksforGeeks. (2023). JavaScript Tutorial. https://www.geeksforgeeks.org/javascript/

[8] JavaScript Tutorial. (2024). JavaScript Tutorial - Getting Started and Fundamentals. https://www.javascripttutorial.net

[9] JetBrains. (2024). How to Learn JavaScript: Your Step-by-Step Guide for Beginners. https://blog.jetbrains.com/webstorm/2024/07/how-to-learn-javascript/