JavaScript Fundamentals Documentation

Table of Contents

1. Introduction
2. Variables and Data Types
3. Operators
4. Control Flow
5. Functions
6. Arrays and Objects
7. Classes and Object-Oriented Programming
8. Asynchronous JavaScript
9. Promises
10. API Calls
11. Error Handling
12. Modern JavaScript Features
13. Best Practices

Introduction

JavaScript is a versatile, dynamic programming language primarily used for web development. It runs in browsers and can also be used on servers with Node.js. This documentation covers the fundamental concepts you need to master JavaScript programming.

Key Characteristics

- **Dynamic typing**: Variables don't need explicit type declarations
- **Interpreted language**: No compilation step required
- **Event-driven**: Responds to user interactions and system events
- **Prototype-based**: Objects can inherit directly from other objects

---

Variables and Data Types

Variable Declarations

JavaScript provides three ways to declare variables:

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

// let - block-scoped (preferred for mutable variables)
let age = 25;

// const - block-scoped, immutable binding (preferred for constants)
const PI = 3.14159;
```

Data Types

Primitive Types

Number

```
let integer = 42;
let float = 3.14;
let scientific = 1e5; // 100000
let infinity = Infinity;
let notANumber = NaN;
```

String

```
let singleQuote = 'Hello';
let doubleQuote = "World";
let templateLiteral = `Hello ${name}!`;
let multiline = `This is a
multiline string`;
```

Boolean

```
let isTrue = true;
let isFalse = false;
```

Undefined and Null

```
let undefinedVar; // undefined
let nullVar = null; // null
```

Symbol (ES6+)

```
let sym = Symbol('description');
let uniqueId = Symbol.for('id');
```

BigInt (ES2020+)

```
let bigNumber = 123n;
let anotherBig = BigInt(123);
```

Complex Types

Object

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

Array

```
let numbers = [1, 2, 3, 4, 5];
let mixed = [1, "hello", true, null];
```

Type Checking

```
typeof 42;          // "number"
typeof "hello";     // "string"
typeof true;        // "boolean"
typeof undefined;   // "undefined"
typeof null;        // "object" (known quirk)
typeof {};          // "object"
typeof [];          // "object"
typeof function(){} // "function"

// Better array checking
Array.isArray([]);  // true
```

Operators

Arithmetic Operators

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

a + b;  // 13 (addition)
a - b;  // 7 (subtraction)
a * b;  // 30 (multiplication)
a / b;  // 3.333... (division)
```

```
a % b;  // 1 (modulo)
a ** b; // 1000 (exponentiation)

// Increment/Decrement
a++; // post-increment
++a; // pre-increment
a--; // post-decrement
--a; // pre-decrement
```

Comparison Operators

```
// Equality
5 == "5";   // true (loose equality)
5 === "5";  // false (strict equality)
5 != "5";   // false (loose inequality)
5 !== "5";  // true (strict inequality)

// Relational
5 > 3;      // true
5 < 3;      // false
5 >= 5;     // true
5 <= 3;     // false
```

Logical Operators

```
true && false;  // false (AND)
true || false;  // true (OR)
!true;          // false (NOT)

// Short-circuit evaluation
false && expensive(); // expensive() not called
true || expensive();  // expensive() not called
```

Assignment Operators

```
let x = 5;
x += 3;  // x = x + 3 (8)
x -= 2;  // x = x - 2 (6)
x *= 2;  // x = x * 2 (12)
x /= 4;  // x = x / 4 (3)
x %= 2;  // x = x % 2 (1)
x **= 3; // x = x ** 3 (1)
```

Control Flow

Conditional Statements

If Statement

```
if (condition) {
    // code to execute if condition is true
} else if (anotherCondition) {
    // code to execute if anotherCondition is true
} else {
    // code to execute if all conditions are false
}
```

Ternary Operator

```
let result = condition ? valueIfTrue : valueIfFalse;
let status = age >= 18 ? "adult" : "minor";
```

Switch Statement

```
switch (day) {
    case 'Monday':
        console.log("Start of work week");
        break;
    case 'Friday':
        console.log("TGIF!");
        break;
    case 'Saturday':
    case 'Sunday':
        console.log("Weekend!");
        break;
    default:
        console.log("Regular day");
}
```

Loops

For Loop

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

// For...of (arrays/iterables)
for (let item of array) {
    console.log(item);
}
```

```
// For...in (object properties)
for (let key in object) {
    console.log(key, object[key]);
}
```

While Loop

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

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

Break and Continue

```
for (let i = 0; i < 10; i++) {
    if (i === 3) continue; // skip iteration
    if (i === 7) break;    // exit loop
    console.log(i);
}
```

Functions

Function Declarations

Function Declaration

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

Function Expression

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

```
};
```

Arrow Functions (ES6+)

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

// Concise arrow function
const greet = name => `Hello, ${name}!`;

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

// No parameters
const random = () => Math.random();
```

Function Parameters

Default Parameters

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

Rest Parameters

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

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

Destructuring Parameters

```
function createUser({name, age, email}) {
    return {
        name,
        age,
        email,
        id: Date.now()
    };
}
```

```
createUser({name: "John", age: 30, email: "john@example.com"});
```

Higher-Order Functions

```
// Function that takes another function as parameter
function operate(a, b, operation) {
    return operation(a, b);
}

const add = (x, y) => x + y;
const multiply = (x, y) => x * y;

operate(5, 3, add);      // 8
operate(5, 3, multiply); // 15

// Function that returns another function
function createMultiplier(multiplier) {
    return function(x) {
        return x * multiplier;
    };
}

const double = createMultiplier(2);
double(5); // 10
```

Closures

```
function outerFunction(x) {
    // Inner function has access to outer function's variables
    function innerFunction(y) {
        return x + y;
    }
    return innerFunction;
}

const addFive = outerFunction(5);
addFive(3); // 8
```

Arrays and Objects

Arrays

Array Creation

```
let fruits = ["apple", "banana", "orange"];
let numbers = new Array(1, 2, 3, 4, 5);
let empty = [];
```

Array Methods

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

// Adding/Removing elements
arr.push(6);        // Add to end
arr.pop();          // Remove from end
arr.unshift(0);     // Add to beginning
arr.shift();        // Remove from beginning
arr.splice(2, 1);   // Remove 1 element at index 2

// Searching
arr.indexOf(3);     // Find index of element
arr.includes(3);    // Check if element exists
arr.find(x => x > 3); // Find first element matching condition

// Transformation
arr.map(x => x * 2);        // Transform each element
arr.filter(x => x > 2);     // Filter elements
arr.reduce((sum, x) => sum + x, 0); // Reduce to single value

// Iteration
arr.forEach(x => console.log(x));
```

Array Destructuring

```
let [first, second, ...rest] = [1, 2, 3, 4, 5];
// first = 1, second = 2, rest = [3, 4, 5]
```

Objects

Object Creation

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

// Constructor function
function Person(name, age) {
    this.name = name;
```

```
    this.age = age;
}

let john = new Person("John", 30);
```

Object Methods

```
let person = {
    name: "John",
    age: 30,
    greet() {
        return `Hello, I'm ${this.name}`;
    }
};

// Accessing properties
person.name;        // Dot notation
person["age"];      // Bracket notation

// Adding/Modifying properties
person.email = "john@example.com";
person["phone"] = "123-456-7890";

// Object methods
Object.keys(person);    // Get all keys
Object.values(person);  // Get all values
Object.entries(person); // Get key-value pairs
```

Object Destructuring

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

// With renaming
let {name: fullName, age: years} = person;

// With default values
let {name, country = "USA"} = person;
```

Spread Operator

```
// Arrays
let arr1 = [1, 2, 3];
let arr2 = [...arr1, 4, 5]; // [1, 2, 3, 4, 5]

// Objects
```

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

Classes and Object-Oriented Programming

Class Declaration

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

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

    static getSpecies() {
        return "Homo sapiens";
    }
}

let john = new Person("John", 30);
john.greet(); // "Hello, I'm John"
Person.getSpecies(); // "Homo sapiens"
```

Inheritance

```
class Animal {
    constructor(name, type) {
        this.name = name;
        this.type = type;
    }

    makeSound() {
        return "Some generic sound";
    }
}

class Dog extends Animal {
    constructor(name, breed) {
        super(name, "mammal");
        this.breed = breed;
    }
```

```
    makeSound() {
        return "Woof!";
    }

    fetch() {
        return `${this.name} is fetching the ball`;
    }
}

let rex = new Dog("Rex", "Golden Retriever");
rex.makeSound(); // "Woof!"
rex.fetch(); // "Rex is fetching the ball"
```

Getters and Setters

```
class Temperature {
    constructor(celsius) {
        this._celsius = celsius;
    }

    get fahrenheit() {
        return (this._celsius * 9/5) + 32;
    }

    set fahrenheit(value) {
        this._celsius = (value - 32) * 5/9;
    }

    get celsius() {
        return this._celsius;
    }

    set celsius(value) {
        this._celsius = value;
    }
}

let temp = new Temperature(25);
console.log(temp.fahrenheit); // 77
temp.fahrenheit = 86;
console.log(temp.celsius); // 30
```

Private Fields (ES2022+)

```
class BankAccount {
    #balance = 0;
```

```
    constructor(initialBalance) {
        this.#balance = initialBalance;
    }

    deposit(amount) {
        this.#balance += amount;
    }

    withdraw(amount) {
        if (amount <= this.#balance) {
            this.#balance -= amount;
        }
    }

    getBalance() {
        return this.#balance;
    }
}
```

Asynchronous JavaScript

Callbacks

```
function fetchData(callback) {
    setTimeout(() => {
        const data = "Some data";
        callback(data);
    }, 1000);
}

fetchData((data) => {
    console.log(data); // "Some data"
});
```

Promises

Creating Promises

```
const myPromise = new Promise((resolve, reject) => {
    const success = true;

    if (success) {
        resolve("Operation successful");
    } else {
        reject("Operation failed");
```

```
    }
});

// Using promises
myPromise
    .then(result => {
        console.log(result); // "Operation successful"
    })
    .catch(error => {
        console.error(error);
    });
```

Promise Methods

```
// Promise.all - wait for all promises
Promise.all([promise1, promise2, promise3])
    .then(results => {
        // All promises resolved
    })
    .catch(error => {
        // At least one promise rejected
    });

// Promise.race - first promise to settle
Promise.race([promise1, promise2, promise3])
    .then(result => {
        // First promise resolved
    });

// Promise.allSettled - wait for all promises to settle
Promise.allSettled([promise1, promise2, promise3])
    .then(results => {
        // All promises settled (resolved or rejected)
    });
```

Async/Await

```
async function fetchUserData() {
    try {
        const response = await fetch('https://api.example.com/user');
        const userData = await response.json();
        return userData;
    } catch (error) {
        console.error('Error fetching user data:', error);
        throw error;
    }
}
```

```
// Using async function
async function main() {
    try {
        const user = await fetchUserData();
        console.log(user);
    } catch (error) {
        console.error('Failed to load user:', error);
    }
}

main();
```

API Calls

Fetch API

Basic GET Request

```
fetch('https://api.example.com/data')
    .then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        return response.json();
    })
    .then(data => {
        console.log(data);
    })
    .catch(error => {
        console.error('Error:', error);
    });
```

POST Request

```
fetch('https://api.example.com/users', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
    },
    body: JSON.stringify({
        name: 'John Doe',
        email: 'john@example.com'
    })
```

```
})
.then(response => response.json())
.then(data => {
    console.log('Success:', data);
})
.catch(error => {
    console.error('Error:', error);
});
```

With Async/Await

```
async function apiCall() {
    try {
        const response = await fetch('https://api.example.com/data');

        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }

        const data = await response.json();
        return data;
    } catch (error) {
        console.error('API call failed:', error);
        throw error;
    }
}
```

Request Configuration

```
const requestOptions = {
    method: 'PUT',
    headers: {
        'Content-Type': 'application/json',
        'Authorization': 'Bearer token123',
        'X-Custom-Header': 'value'
    },
    body: JSON.stringify({
        key: 'value'
    })
};

fetch('https://api.example.com/resource', requestOptions)
    .then(response => response.json())
    .then(data => console.log(data));
```

Error Handling

Try-Catch-Finally

```
try {
    // Code that might throw an error
    let result = riskyOperation();
    console.log(result);
} catch (error) {
    // Handle the error
    console.error('An error occurred:', error.message);
} finally {
    // Code that always runs
    console.log('Cleanup operations');
}
```

Custom Errors

```
class ValidationError extends Error {
    constructor(message) {
        super(message);
        this.name = 'ValidationError';
    }
}

function validateEmail(email) {
    if (!email.includes('@')) {
        throw new ValidationError('Invalid email format');
    }
}

try {
    validateEmail('invalid-email');
} catch (error) {
    if (error instanceof ValidationError) {
        console.log('Validation failed:', error.message);
    } else {
        console.log('Unexpected error:', error);
    }
}
```

Error Handling with Promises

```
fetch('https://api.example.com/data')
    .then(response => {
        if (!response.ok) {
            throw new Error(`HTTP ${response.status}: ${response.statusText}`);
```

```
        }
        return response.json();
    })
    .then(data => {
        console.log(data);
    })
    .catch(error => {
        console.error('Request failed:', error.message);
    });
```

Modern JavaScript Features

Template Literals

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

// Multi-line strings
const message = `
    Hello, ${name}!
    You are ${age} years old.
    Next year you'll be ${age + 1}.
`;

// Tagged template literals
function highlight(strings, ...values) {
    return strings.reduce((result, string, i) => {
        return result + string + (values[i] ? `<mark>${values[i]}</mark>` : '');
    }, '');
}

const highlighted = highlight`Hello ${name}, you are ${age} years old`;
```

Modules

Exporting

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

Importing

```
// main.js
import multiply, { PI, add } from './math.js';
import * as math from './math.js';

console.log(PI); // 3.14159
console.log(add(2, 3)); // 5
console.log(multiply(4, 5)); // 20
```

Optional Chaining

```
const user = {
    name: "John",
    address: {
        street: "123 Main St",
        city: "New York"
    }
};

// Safe property access
console.log(user?.address?.street); // "123 Main St"
console.log(user?.phone?.number); // undefined (no error)

// Safe method calling
user?.someMethod?.(); // Won't throw error if method doesn't exist
```

Nullish Coalescing

```
const userInput = null;
const defaultValue = "default";

// Using nullish coalescing operator
const value = userInput ?? defaultValue; // "default"

// Difference from logical OR
const emptyString = "";
console.log(emptyString || "default"); // "default"
console.log(emptyString ?? "default"); // ""
```

Best Practices

Code Style

1. **Use meaningful variable names**

```
// Bad
let d = new Date();
let u = users.filter(u => u.a);

// Good
let currentDate = new Date();
let activeUsers = users.filter(user => user.isActive);
```

2. **Use const by default, let when reassignment is needed**

```
// Good
const apiUrl = 'https://api.example.com';
const users = [];

let currentUser = null; // Will be reassigned later
```

3. **Use template literals for string interpolation**

```
// Bad
const message = 'Hello, ' + name + '! You have ' + count + ' messages.';

// Good
const message = `Hello, ${name}! You have ${count} messages.`;
```

Performance Tips

1. **Use efficient array methods**

```
// Find first matching element
const user = users.find(user => user.id === targetId);

// Check if any element matches
const hasAdmin = users.some(user => user.role === 'admin');

// Avoid creating unnecessary arrays
users.forEach(user => processUser(user)); // Better than users.map() when not using result
```

2. **Avoid memory leaks**

```
// Remove event listeners when no longer needed
const button = document.getElementById('myButton');
```

```
const handler = () => console.log('clicked');

button.addEventListener('click', handler);
// Later...
button.removeEventListener('click', handler);
```

Security Considerations

1. **Validate user input**

```
function sanitizeInput(input) {
    if (typeof input !== 'string') {
        throw new Error('Input must be a string');
    }
    return input.trim().replace(/[<>]/g, '');
}
```

2. **Use HTTPS for API calls**

```
const apiUrl = 'https://api.example.com'; // Always use HTTPS
```

3. **Avoid eval() and similar functions**

```
// Bad
eval(userInput);

// Good
JSON.parse(userInput); // For JSON data
```

Conclusion

This documentation covers the fundamental concepts of JavaScript programming. Master these concepts and you'll have a solid foundation for building web applications, server-side applications with Node.js, and modern JavaScript frameworks.

Remember to:

- Practice regularly with coding exercises
- Read other developers' code to learn different approaches
- Stay updated with new JavaScript features
- Use developer tools for debugging
- Write clean, readable code with proper error handling

For more advanced topics, explore:

- Web APIs (DOM manipulation, LocalStorage, etc.)
- JavaScript frameworks (React, Vue, Angular)
- Node.js for server-side development
- Testing frameworks (Jest, Mocha)
- Build tools (Webpack, Vite)

Happy coding!