JavaScript ES6+ Complete Learning Path

Table of Contents

- 1. Fundamentals
- 2. **DOM Manipulation**
- 3. ES6+ Core Features
- 4. Asynchronous JavaScript
- 5. OOP & Prototypes
- 6. Functional Programming
- 7. Advanced Concepts
- 8. Modern APIs & Tools
- 9. Performance Optimization
- 10. Testing & Debugging
- 11. Best Practices

1. Fundamentals

Variables: (let), (const) vs (var)

```
javascript

// ★ Avoid var - function scoped, hoisted, can be redeclared

var oldWay = 'avoid this';

// ✓ Use const for values that won't be reassigned

const API_URL = 'https://api.example.com';

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

// ✓ Use let for reassignable variables

let counter = 0;

let currentPage = 1;

// Block scoping demonstration

if (true) {

let blockScoped = 'only available here';

const alsoBlockScoped = 'same here';

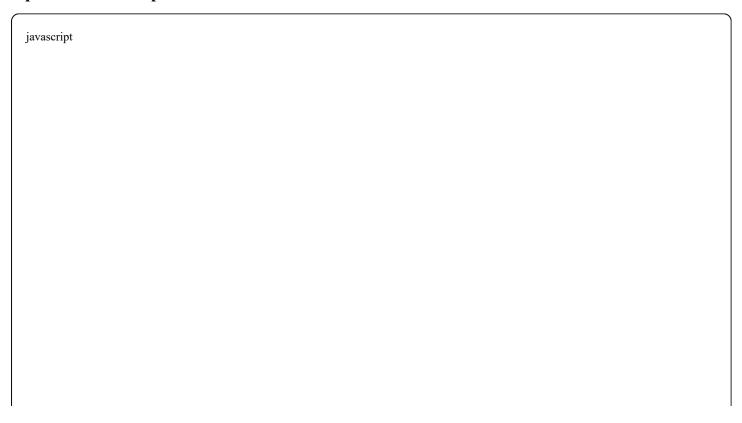
}

// console.log(blockScoped); // ReferenceError
```

Data Types & Type Checking

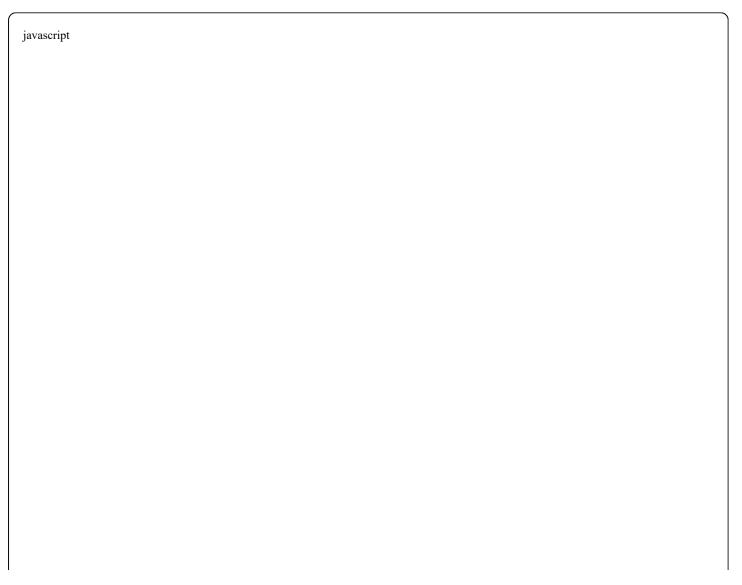
```
javascript
// Primitive types
const str = 'Hello World';
const num = 42;
const bool = true;
const undef = undefined;
const nul = null;
const sym = Symbol('unique');
const bigInt = 123n;
// Reference types
const obj = { key: 'value' };
const arr = [1, 2, 3];
const func = () \Rightarrow \{\};
// Modern type checking
const checkType = (value) => {
 if (Array.isArray(value)) return 'array';
 if (value === null) return 'null';
 return typeof value;
};
console.log(checkType([])); // 'array'
console.log(checkType(null)); // 'null'
```

Operators & Comparisons



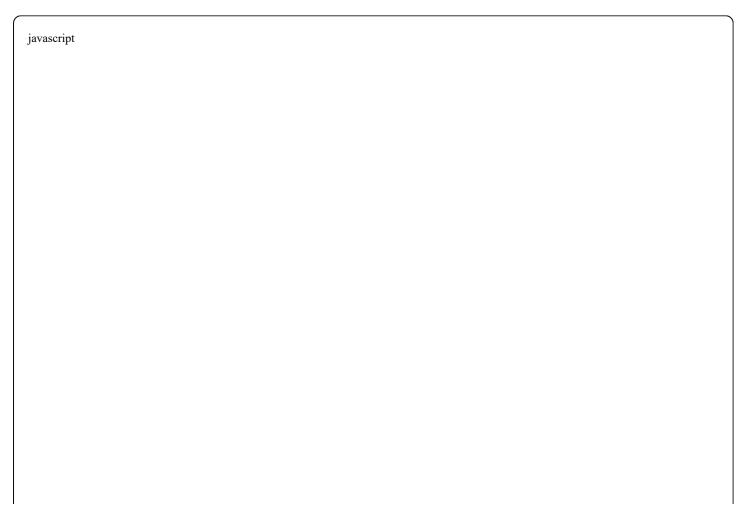
```
// Strict equality (always use)
const a = 5;
const b = '5';
console.log(a === b); // false
console.log(a == b); // true (avoid!)
// Nullish coalescing (??)
const username = null;
const displayName = username ?? 'Guest'; // 'Guest'
// Optional chaining (?.)
const user = { profile: { name: 'John' } };
const name = user?.profile?.name; // 'John'
const email = user?.profile?.email ?? 'No email'; // 'No email'
// Logical assignment operators
let config = {};
config.theme ??= 'dark'; // Only assign if null/undefined
config.debug |= false; // Assign if falsy
```

Modern Loops



```
const items = ['apple', 'banana', 'cherry'];
const prices = { apple: 1, banana: 2, cherry: 3 };
// for...of for arrays (gets values)
for (const item of items) {
 console.log(item); // apple, banana, cherry
// for...in for objects (gets keys)
for (const key in prices) {
 console.log(`${key}: $${prices[key]}`);
}
// forEach for arrays (functional approach)
items.forEach((item, index) => {
 console.log(`${index}: ${item}`);
});
// Modern array methods
const expensiveItems = Object.entries(prices)
 .filter(([\_, price]) \Rightarrow price > 1)
 .map(([name, price]) => ({ name, price }));
```

Functions & Arrow Functions



```
function declaration (hoisted)
function traditionalFunction(x, y) {
    return x + y;
}

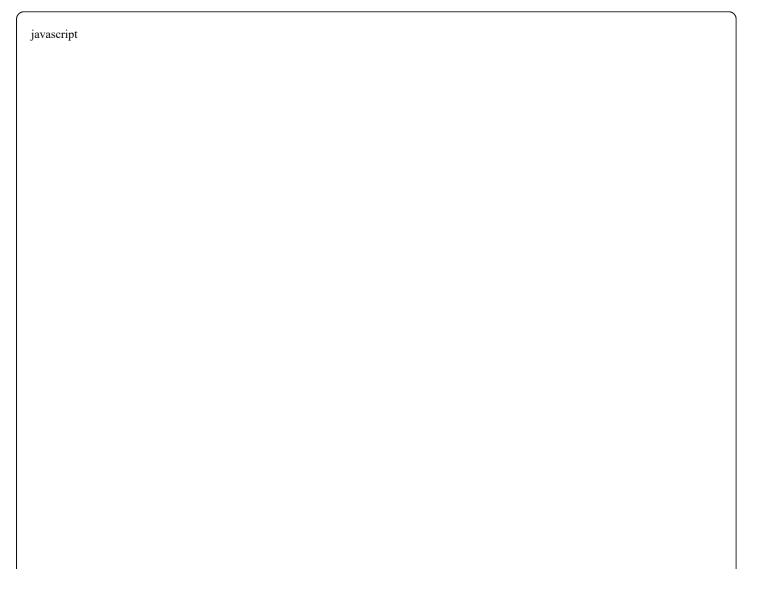
// Arrow function (not hoisted, lexical this)
const arrowFunction = (x, y) => x + y;

// Default parameters
const greet = (name = 'World', greeting = 'Hello') => {
    return `${greeting}, ${name}!';
};

// Rest parameters
const sum = (...numbers) => {
    return numbers.reduce((total, num) => total + num, 0);
};

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

Scope, Hoisting & Temporal Dead Zone



```
// Temporal Dead Zone example
console.log(x); // ReferenceError: Cannot access 'x' before initialization
let x = 5;
// Function hoisting
console.log(hoistedFunc()); // Works! Returns "I'm hoisted"
function hoistedFunc() {
 return "I'm hoisted";
// Block scope example
function scopeExample() {
 if (true) {
  let blockVar = 'block scoped';
  const blockConst = 'also block scoped';
  var functionVar = 'function scoped';
 // console.log(blockVar); // ReferenceError
 console.log(functionVar); // Works - function scoped
}
```

2. DOM Manipulation

Selecting Elements

```
javascript

// Modern selectors

const element = document.querySelector('.my-class');

const elements = document.querySelectorAll('.item');

const byId = document.getElementById('myId');

// Check if element exists

const safeElement = document.querySelector('.maybe-exists');

if (safeElement) {

safeElement.textContent = 'Found it!';

}

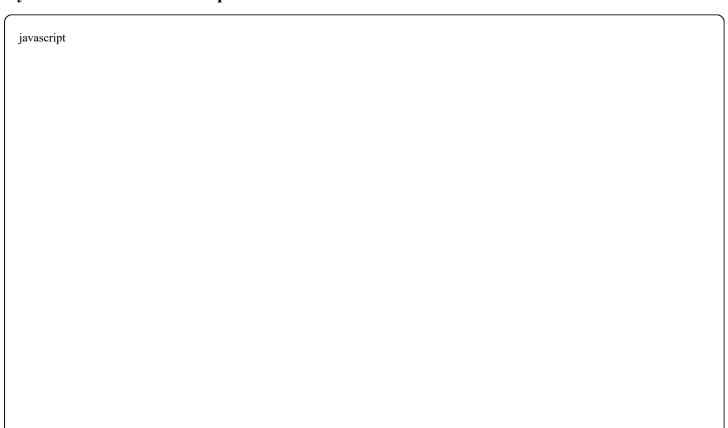
// Alternative null-safe approach

document.querySelector('.maybe-exists')?.classList.add('active');
```

Event Handling

```
javascript
// Modern event listeners
const button = document.querySelector('#submit-btn');
const form = document.querySelector('#user-form');
// Arrow functions maintain lexical this
button?.addEventListener('click', (e) => {
 e.preventDefault();
 console.log('Button clicked!');
});
// Event delegation for dynamic content
document.addEventListener('click', (e) => {
 if (e.target.matches('.delete-btn')) {
  e.target.closest('.item').remove();
});
// Custom events
const customEvent = new CustomEvent('userLoggedIn', {
 detail: { userId: 123, timestamp: Date.now() }
});
document.dispatchEvent(customEvent);
```

Dynamic HTML & DOM Updates



```
// Create elements efficiently
const createProductCard = (product) => {
 const card = document.createElement('div');
 card.className = 'product-card';
 card.innerHTML = `
  \frac{h3}{\$} {product.name}\frac{h3}{h3}
  $${product.price}
  <button data-id="${product.id}">Add to Cart</button>
 return card;
};
// Batch DOM updates
const products = [
 { id: 1, name: 'Laptop', price: 999 },
 { id: 2, name: 'Phone', price: 699 }
];
const container = document.querySelector('#products');
const fragment = document.createDocumentFragment();
products.forEach(product => {
 fragment.appendChild(createProductCard(product));
});
container.appendChild(fragment); // Single reflow
```

Form Handling

```
// Modern form handling
const form = document.querySelector('#registration-form');
form?.addEventListener('submit', async (e) => {
 e.preventDefault();
 // FormData API
 const formData = new FormData(form);
 const data = Object.fromEntries(formData);
 // Validation
 const errors = validateForm(data);
 if (errors.length > 0) {
  displayErrors(errors);
  return;
 try {
  await submitForm(data);
  form.reset();
  showSuccess('Registration successful!');
 } catch (error) {
  showError('Registration failed. Please try again.');
 }
});
const validateForm = (data) => {
 const errors = [];
 if (!data.email?.includes('@')) {
  errors.push('Valid email required');
 if (data.password?.length < 8) {
  errors.push('Password must be at least 8 characters');
 return errors;
};
```

AJAX: Fetch API vs XMLHttpRequest

```
// Modern Fetch API
const fetchUserData = async (userId) => {
 try {
  const response = await fetch(`/api/users/${userId}`, {
   method: 'GET',
   headers: {
    'Content-Type': 'application/json',
    'Authorization': `Bearer ${token}`
  });
  if (!response.ok) {
   throw new <a>Error</a>('HTTP error! status: ${response.status}');
  return await response.json();
 } catch (error) {
  console.error('Fetch error:', error);
  throw error;
};
// X Old XMLHttpRequest (avoid)
const xhr = new XMLHttpRequest();
xhr.open('GET', '/api/users/123');
xhr.onreadystatechange = function() {
 if (xhr.readyState === 4 && xhr.status === 200) {
  const data = JSON.parse(xhr.responseText);
  // Handle response
 }
};
xhr.send();
```

3. ES6+ Core Features

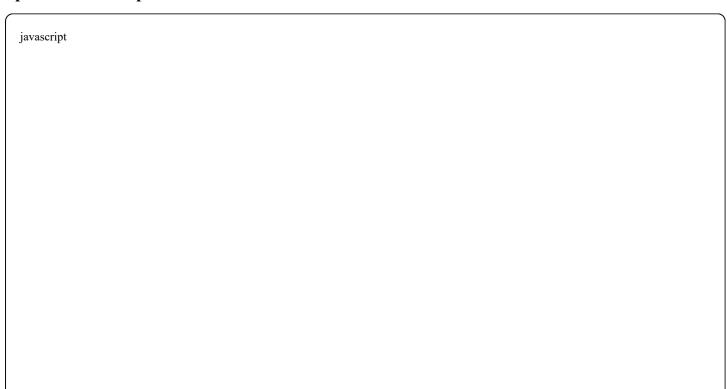
Arrow Functions

```
// Basic syntax
const add = (a, b) \Rightarrow a + b;
const square = x \Rightarrow x * x;
const greet = () => 'Hello!';
// Lexical this binding
class Timer {
 constructor() {
  this.seconds = 0;
 start() {
  // Arrow function preserves 'this'
  setInterval(() => {
    this.seconds++;
    console.log(this.seconds);
  }, 1000);
 }
}
// When NOT to use arrow functions
const obj = {
 name: 'John',
 // X Don't use arrow function for methods
 greetBad: () => `Hello, ${this.name}`, // 'this' is undefined
 // Use regular function for methods
 greetGood() {
  return `Hello, ${this.name}`;
};
```

Destructuring

```
// Array destructuring
const colors = ['red', 'green', 'blue'];
const [primary, secondary, ...rest] = colors;
console.log(primary); // 'red'
console.log(rest); // ['blue']
// Object destructuring
const user = { name: 'John', age: 30, city: 'NYC' };
const { name, age, city = 'Unknown' } = user;
// Nested destructuring
const response = {
 data: {
  users: [{ id: 1, name: 'Alice' }]
 }
};
const { data: { users: [firstUser] } } = response;
// Function parameter destructuring
const createUser = ({ name, age, email }) => {
 return { id: Date.now(), name, age, email };
};
// Swapping variables
let a = 1, b = 2;
[a, b] = [b, a];
```

Spread & Rest Operators



```
// Spread operator (...)
const arr1 = [1, 2, 3];
const arr2 = [4, 5, 6];
const combined = [...arr1, ...arr2]; // [1, 2, 3, 4, 5, 6]

// Object spread
const user = { name: 'John', age: 30 };
const updatedUser = { ...user, age: 31, city: 'NYC' };

// Rest parameters
const calculateTotal = (tax, ...prices) => {
    const subtotal = prices.reduce((sum, price) => sum + price, 0);
    return subtotal + (subtotal * tax);
};

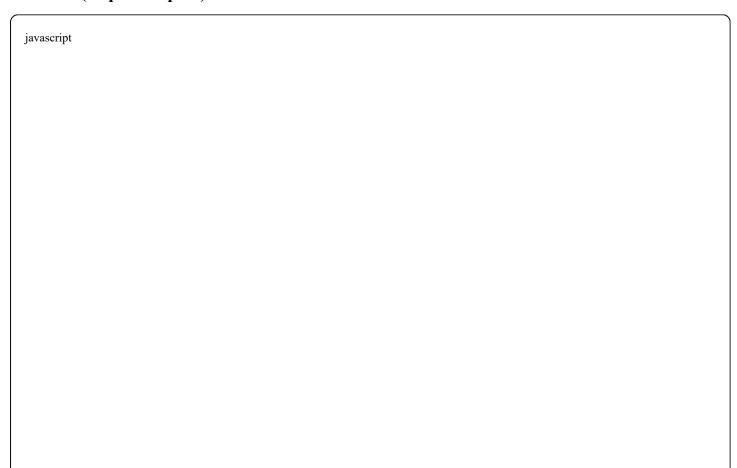
// Spread in function calls
const numbers = [1, 2, 3, 4, 5];
const max = Math.max(...numbers);
```

Template Literals



```
const name = 'John';
const age = 30;
// Basic template literal
const greeting = `Hello, ${name}! You are ${age} years old.`;
// Multi-line strings
const html = `
 <div class="user-card">
  <h2>${name}</h2>
  Age: ${age}
 </div>
// Tagged template literals
const highlight = (strings, ...expressions) => {
 return strings.reduce((result, string, i) => {
  const expression = expressions[i] ? '<mark>${expressions[i]}</mark>' : ";
  return result + string + expression;
 }, ");
};
const message = highlight`Hello ${name}, you have ${5} new messages!`;
```

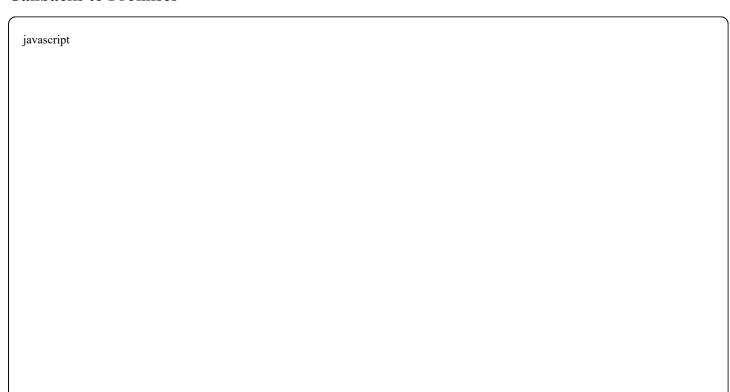
Modules (Import/Export)



```
// math.js - Named exports
export const PI = 3.14159;
export const add = (a, b) \Rightarrow a + b;
export const multiply = (a, b) \Rightarrow a * b;
// utils.js - Default export
const formatCurrency = (amount) => `$${amount.toFixed(2)}`;
export default formatCurrency;
// main.js - Importing
import formatCurrency from './utils.js'; // Default import
import { PI, add, multiply } from './math.js'; // Named imports
import * as math from './math.js'; // Namespace import
// Dynamic imports
const loadModule = async () => {
 const { default: formatCurrency } = await import('./utils.js');
 return formatCurrency(123.45);
};
// Re-exports
export { PI, add } from './math.js';
export { default as formatCurrency } from './utils.js';
```

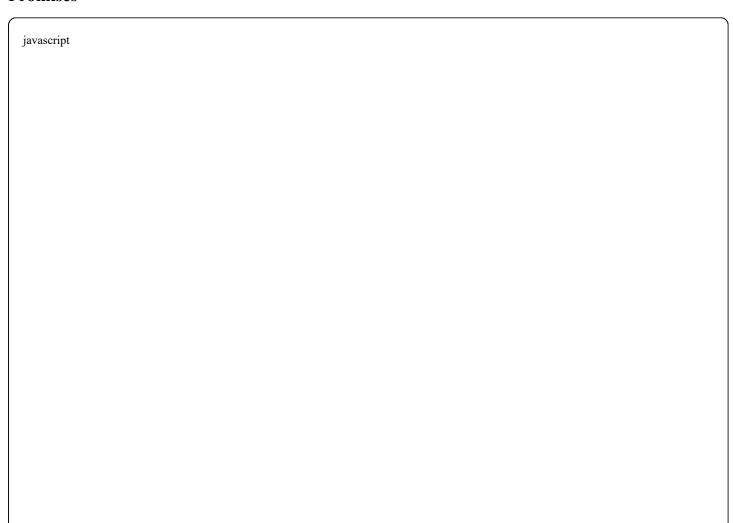
4. Asynchronous JavaScript

Callbacks to Promises



```
// 🗶 Callback hell
function fetchUserData(userId, callback) {
 setTimeout(() => {
  fetchUser(userId, (user) => {
   fetchUserPosts(user.id, (posts) => {
     fetchPostComments(posts[0].id, (comments) => {
      callback({ user, posts, comments });
    });
   });
  });
 }, 1000);
// Promise chain
const fetchUserData = (userId) => {
 return fetchUser(userId)
  .then(user => fetchUserPosts(user.id))
  .then(posts => fetchPostComments(posts[0].id))
  .then(comments => ({ user, posts, comments }));
};
```

Promises



```
// Creating promises
const delay = (ms) => new Promise(resolve => setTimeout(resolve, ms));
const fetchData = (url) => {
 return new Promise((resolve, reject) => {
  fetch(url)
    .then(response => {
     if (!response.ok) {
      reject(new Error(`HTTP ${response.status}`));
     return response.json();
    })
   .then(data => resolve(data))
   .catch(error => reject(error));
 });
};
// Promise.all vs Promise.allSettled
const urls = ['/api/users', '/api/posts', '/api/comments'];
// Promise.all - fails if any promise fails
const loadAllData = async () => {
 try {
  const [users, posts, comments] = await Promise.all(
   urls.map(url => fetch(url).then(r => r.json()))
  );
  return { users, posts, comments };
 } catch (error) {
  console.error('One or more requests failed:', error);
};
// Promise.allSettled - waits for all promises regardless of outcome
const loadAllDataSafely = async () => {
 const results = await Promise.allSettled(
  urls.map(url => fetch(url).then(r => r.json()))
 );
 const successful = results
  .filter(result => result.status === 'fulfilled')
  .map(result => result.value);
 const failed = results
  .filter(result => result.status === 'rejected')
  .map(result => result.reason);
```

```
return { successful, failed };
};
```

Async/Await

```
javascript
// Clean async/await syntax
const fetchUserProfile = async (userId) => {
 try {
  const user = await fetchUser(userId);
  const posts = await fetchUserPosts(user.id);
  const followers = await fetchUserFollowers(user.id);
  return {
   ...user,
   postsCount: posts.length,
   followersCount: followers.length
  };
 } catch (error) {
  console.error('Error fetching user profile:', error);
  throw new Error('Failed to load user profile');
 }
};
// Parallel vs Sequential execution
const sequentialFetch = async () => {
 const user = await fetchUser(1); // Wait
 const posts = await fetchPosts(1); // Then wait
 return { user, posts };
};
const parallelFetch = async () => {
 const [user, posts] = await Promise.all([
  fetchUser(1),
  fetchPosts(1)
 ]); // Execute simultaneously
 return { user, posts };
};
```

Error Handling

```
javascript
```

```
// Comprehensive error handling
class APIError extends Error {
 constructor(message, status, code) {
  super(message);
  this.name = 'APIError';
  this.status = status;
  this.code = code;
const apiRequest = async (url, options = {}) => {
 try {
  const response = await fetch(url, {
   ...options,
   headers: {
     'Content-Type': 'application/json',
     ...options.headers
   }
  });
  if (!response.ok) {
   const errorData = await response.json();
   throw new APIError(
     errorData.message | 'Request failed',
     response.status,
     errorData.code
   );
  return await response.json();
 } catch (error) {
  if (error instanceof APIError) {
   throw error;
  // Network or other errors
  throw new Error(`Network error: ${error.message}`);
};
// Using the error handling
const handleUserLogin = async (credentials) => {
  const user = await apiRequest('/api/login', {
   method: 'POST',
   body: JSON.stringify(credentials)
```

```
return { success: true, user };
} catch (error) {

if (error instanceof APIError && error: 'Invalid credentials' };

return { success: false, error: 'Invalid credentials' };
}

return { success: false, error: 'Login failed. Please try again.' };
};
```

Microtasks vs Macrotasks

```
javascript
// Understanding the event loop
console.log('1: Start');
setTimeout(() => console.log('2: Timeout'), 0); // Macrotask
Promise.resolve().then(() => console.log('3: Promise')); // Microtask
console.log('4: End');
// Output: 1: Start, 4: End, 3: Promise, 2: Timeout
// Practical example: Batching DOM updates
const batchDOMUpdates = () => {
 const element = document.querySelector('#status');
 // These will be batched together
 element.textContent = 'Loading...';
 element.classList.add('loading');
 // This will run after DOM updates
 Promise.resolve().then(() => {
  console.log('DOM updated');
 });
};
```

5. OOP & Prototypes

Constructor Functions vs Classes

```
javascript
// X Old constructor function pattern
function User(name, email) {
 this.name = name;
 this.email = email;
}
User.prototype.greet = function() {
 return `Hello, I'm ${this.name}`;
};
// Modern class syntax
class User {
 constructor(name, email) {
  this.name = name;
  this.email = email;
 greet() {
  return `Hello, I'm ${this.name}`;
 // Static method
 static fromJSON(json) {
  const data = JSON.parse(json);
  return new User(data.name, data.email);
 // Getter/Setter
 get displayName() {
  return this.name.toUpperCase();
 set displayName(value) {
  this.name = value.toLowerCase();
}
```

Inheritance

```
class Animal {
 constructor(name, species) {
  this.name = name;
  this.species = species;
 speak() {
  return `${this.name} makes a sound`;
 // Protected method (convention)
 _sleep() {
  return `${this.name} is sleeping`;
}
class Dog extends Animal {
 constructor(name, breed) {
  super(name, 'Canine');
  this.breed = breed;
 speak() {
  return `${this.name} barks`;
 }
 fetch() {
  return `${this.name} fetches the ball`;
// Usage
const dog = new Dog('Rex', 'German Shepherd');
console.log(dog.speak()); // "Rex barks"
console.log(dog instanceof Dog); // true
console.log(dog instanceof Animal); // true
```

The (this) Keyword

```
class Calculator {
 constructor() {
  this.result = 0;
 }
 add(value) {
  this.result += value;
  return this; // Method chaining
 multiply(value) {
  this.result *= value;
  return this;
 }
 // Arrow function preserves 'this'
 delayed = (value) => {
  setTimeout(() => {
   this.result += value;
   console.log(this.result);
  }, 1000);
 }
}
// Method chaining
const calc = new Calculator();
calc.add(5).multiply(2).add(3); // result = 13
// Context binding
const obj = {
 name: 'Example',
 regularMethod() {
  console.log(this.name); // 'Example'
 },
 arrowMethod: () => {
  console.log(this.name); // undefined (lexical this)
 }
};
// Explicit binding
const boundMethod = obj.regularMethod.bind({ name: 'Bound' });
boundMethod(); // 'Bound'
```

Prototypal Inheritance

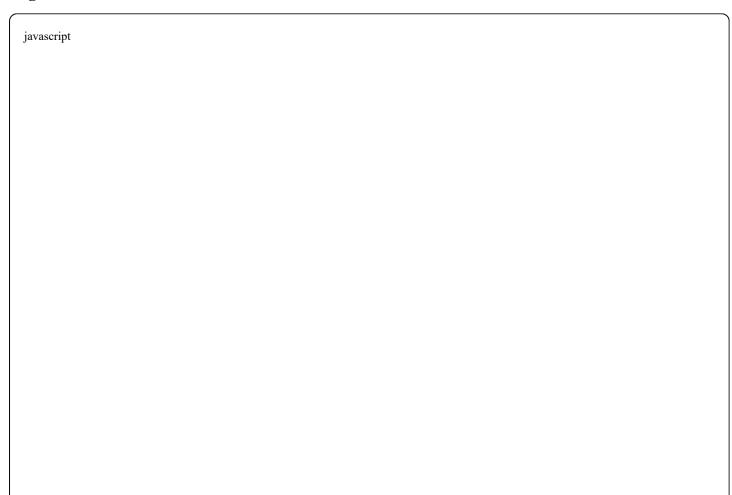
```
// Understanding the prototype chain
const animal = {
 species: 'Unknown',
 speak() {
  return `${this.name} makes a sound`;
};
const dog = Object.create(animal);
dog.name = 'Rex';
dog.bark = function() {
 return `${this.name} barks`;
};
console.log(dog.speak()); // "Rex makes a sound"
console.log(dog.__proto__ === animal); // true
// Prototype pollution prevention
const safeObject = Object.create(null); // No prototype
safeObject.name = 'Safe';
// safeObject.toString(); // TypeError: not a function
// Modern prototype manipulation
class ModernClass {
 static [Symbol.hasInstance](instance) {
  return instance.customProperty ==== 'special';
 }
}
const obj = { customProperty: 'special' };
console.log(obj instanceof ModernClass); // true
```

6. Functional Programming

Pure Functions

```
// Pure function - same input, same output, no side effects
const add = (a, b) => a + b;
const multiply = (a, b) \Rightarrow a * b;
// X Impure function - modifies external state
let counter = 0;
const impureIncrement = () => ++counter;
// V Pure alternative
const increment = (value) => value + 1;
// Pure function for calculations
const calculateTax = (price, taxRate) => {
 return price * taxRate;
};
const calculateTotal = (items, taxRate) => {
 const subtotal = items.reduce((sum, item) => sum + item.price, 0);
 const tax = calculateTax(subtotal, taxRate);
 return subtotal + tax;
};
```

Higher-Order Functions



```
// Functions that return functions
const createMultiplier = (multiplier) => {
 return (value) => value * multiplier;
};
const double = createMultiplier(2);
const triple = createMultiplier(3);
// Functions that accept functions
const withLogging = (fn) => {
 return (...args) => {
  console.log(`Calling ${fn.name} with:`, args);
  const result = fn(...args);
  console.log(`Result:`, result);
  return result;
 };
};
const loggedAdd = withLogging(add);
loggedAdd(2, 3); // Logs function call and result
// Practical example: Retry logic
const with Retry = (fn, maxAttempts = 3) => {
 return async (...args) => {
  for (let attempt = 1; attempt <= maxAttempts; attempt++) {
   try {
     return await fn(...args);
   } catch (error) {
     if (attempt === maxAttempts) throw error;
     console.log(`Attempt ${attempt} failed, retrying...');
 };
};
const fetchWithRetry = withRetry(fetch);
```

Immutability

```
// Immutable array operations
const original Array = [1, 2, 3];
// / Immutable operations
const newArray = [...originalArray, 4]; // Add
const filtered = originalArray.filter(x => x > 1); // Remove
const mapped = originalArray.map(x \Rightarrow x * 2); // Transform
// X Mutable operations (avoid)
// originalArray.push(4);
// originalArray.splice(0, 1);
// Immutable object updates
const originalUser = { name: 'John', age: 30, address: { city: 'NYC' } };
// Shallow copy
const updatedUser = { ...originalUser, age: 31 };
// Deep copy for nested objects
const deepCopy = structuredClone(originalUser);
deepCopy.address.city = 'LA';
// Immutable helper functions
const updateProperty = (obj, key, value) => ({ ...obj, [key]: value });
const updateNestedProperty = (obj, path, value) => {
 const keys = path.split('.');
 const [first, ...rest] = keys;
 if (rest.length === 0) {
  return { ...obj, [first]: value };
 return {
  ...obj,
  [first]: updateNestedProperty(obj[first], rest.join('.'), value)
 };
};
```

Map, Filter, Reduce

```
const products = [
 { id: 1, name: 'Laptop', price: 999, category: 'Electronics' },
 { id: 2, name: 'Book', price: 15, category: 'Education' },
 { id: 3, name: 'Phone', price: 699, category: 'Electronics' },
 { id: 4, name: 'Pen', price: 2, category: 'Stationery' }
];
// Map - transform each element
const productNames = products.map(product => product.name);
const discountedPrices = products.map(product => ({
 ...product,
 discountedPrice: product.price * 0.9
}));
// Filter - select elements matching criteria
const electronics = products.filter(product => product.category === 'Electronics');
const expensiveItems = products.filter(product => product.price > 100);
// Reduce - aggregate data
const totalValue = products.reduce((sum, product) => sum + product.price, 0);
const productsByCategory = products.reduce((acc, product) => {
 const category = product.category;
 if (!acc[category]) {
  acc[category] = [];
 acc[category].push(product);
 return acc;
}, {});
// Chaining operations
const expensiveElectronicsNames = products
 .filter(product => product.category ==== 'Electronics')
 .filter(product => product.price > 500)
 .map(product => product.name);
// Advanced reduce patterns
const stats = products.reduce((acc, product) => {
 acc.totalProducts++;
 acc.totalValue += product.price;
 acc.avgPrice = acc.totalValue / acc.totalProducts;
 if (product.price > acc.maxPrice) {
  acc.maxPrice = product.price;
  acc.mostExpensive = product.name;
```

```
return acc;
}, {

totalProducts: 0,

totalValue: 0,

avgPrice: 0,

maxPrice: 0,

mostExpensive: null
});
```

7. Advanced Concepts

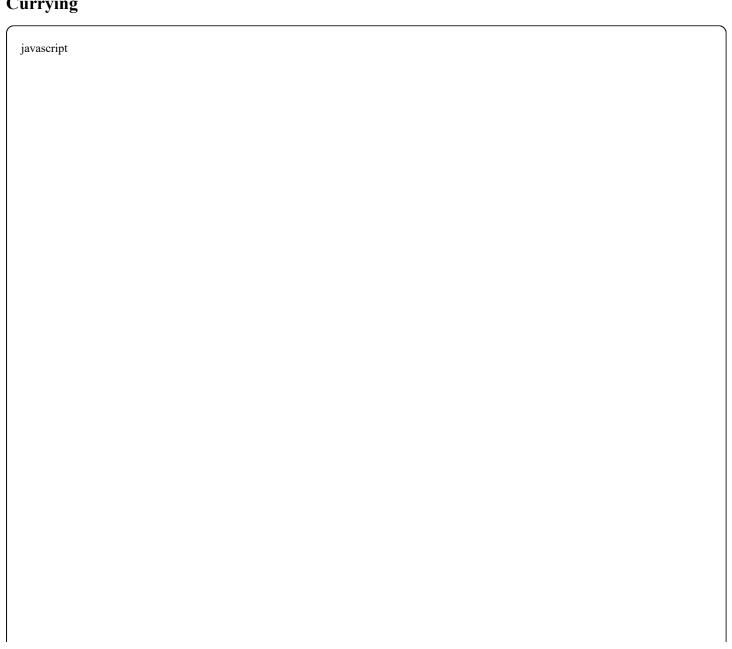
Closures

javascript		

```
// Basic closure
const createCounter = () => {
 let count = 0;
 return {
  increment: () => ++count,
  decrement: () => --count,
  getValue: () => count
 };
};
const counter = createCounter();
console.log(counter.getValue()); // 0
counter.increment();
console.log(counter.getValue()); // 1
// Practical example: Module pattern
const UserModule = (() => {
 const users = [];
 let currentUser = null;
 return {
  addUser(user) {
   users.push({ ...user, id: Date.now() });
  },
  login(email, password) {
   const user = users.find(u => u.email === email && u.password === password);
   if (user) {
    currentUser = user;
     return true;
   return false;
  },
  getCurrentUser() {
   return currentUser ? { ...currentUser } : null;
  },
  getUserCount() {
   return users.length;
  }
 };
})();
// Memory management with closures
```

```
const createTimer = () => {
 let timeoutId;
 return {
  start(callback, delay) {
   this.stop(); // Clear any existing timer
   timeoutId = setTimeout(callback, delay);
  },
  stop() {
   if (timeoutId) {
     clearTimeout(timeoutId);
     timeoutId = null;
  }
 };
};
```

Currying



```
// Basic currying
const add = (a) => (b) => a + b;
const add5 = add(5);
console.log(add5(3)); // 8
// Curry utility function
const curry = (fn) => \{
 return function curried(...args) {
  if (args.length >= fn.length) {
   return fn.apply(this, args);
  }
  return (...nextArgs) => curried.apply(this, [...args, ...nextArgs]);
 };
};
// Practical currying example
const multiply = (a, b, c) \Rightarrow a * b * c;
const curriedMultiply = curry(multiply);
const double = curriedMultiply(2);
const doubleAndTriple = double(3);
console.log(doubleAndTriple(4)); // 24
// Real-world example: API request builder
const apiRequest = (baseUrl) => (endpoint) => (method) => (data) => {
 return fetch(`${baseUrl}${endpoint}`, {
  method,
  headers: { 'Content-Type': 'application/json' },
  body: data? JSON.stringify(data): undefined
 });
};
const apiCall = apiRequest('https://api.example.com');
const usersAPI = apiCall('/users');
const getUsers = usersAPI('GET');
const createUser = usersAPI('POST');
// Usage
getUsers().then(response => response.json());
createUser({ name: 'John', email: 'john@example.com' });
```

Immediately Invoked Function Expressions (IIFEs)

```
// Basic IIFE
(function() {
 console.log('IIFE executed immediately');
})();
// IIFE with parameters
(function(global, undefined) {
 // Safe reference to global object
 // undefined is truly undefined
})(window);
// Modern IIFE with arrow functions
(() => {
 const privateVar = 'not accessible outside';
 // Initialize app
 const initApp = () => {
  console.log('App initialized');
 };
 initApp();
})();
// Practical example: Configuration
const AppConfig = (() => {
 const config = {
  apiUrl: process.env.NODE_ENV === 'production'
   ? 'https://api.production.com'
   : 'https://api.dev.com',
  version: '1.0.0',
  features: {
   darkMode: true,
   analytics: process.env.NODE_ENV === 'production'
 };
 return {
  get(key) {
   return config[key];
  },
  getFeature(feature) {
   return config.features[feature];
```

}; })();		
Veb Workers		
javascript		

```
// main.js - Main thread
const worker = new Worker('worker.js');
// Send data to worker
const processLargeDataset = (data) => {
 worker.postMessage({
  type: 'PROCESS DATA',
  data: data
 });
};
// Listen for worker responses
worker.addEventListener('message', (e) => {
 const { type, result, error } = e.data;
 switch (type) {
  case 'PROCESSING_COMPLETE':
   console.log('Processing complete:', result);
   updateUI(result);
   break;
  case 'PROCESSING ERROR':
   console.error('Worker error:', error);
   showError('Processing failed');
   break:
  case 'PROGRESS UPDATE':
   updateProgressBar(result.progress);
   break;
});
// worker.js - Worker thread
self.addEventListener('message', (e) => {
 const { type, data } = e.data;
 switch (type) {
  case 'PROCESS DATA':
   try {
    const result = processData(data);
    self.postMessage({
      type: 'PROCESSING_COMPLETE',
      result
    });
   } catch (error) {
    self.postMessage({
```

```
type: 'PROCESSING_ERROR',
      error: error.message
     });
    }
   break;
});
const processData = (data) => {
 const processed = [];
 const total = data.length;
 for (let i = 0; i < total; i++) {
  // Simulate heavy computation
  const item = heavyComputation(data[i]);
  processed.push(item);
  // Report progress
  if (i % 100 === 0) {
   self.postMessage({
    type: 'PROGRESS_UPDATE',
    result: { progress: (i / total) * 100 }
   });
 return processed;
```

Memory Management & Garbage Collection

```
// Memory leak prevention
class EventEmitter {
 constructor() {
  this.listeners = new Map();
 on(event, callback) {
  if (!this.listeners.has(event)) {
   this.listeners.set(event, new Set());
  this.listeners.get(event).add(callback);
 off(event, callback) {
  const eventListeners = this.listeners.get(event);
  if (eventListeners) {
   eventListeners.delete(callback);
   if (eventListeners.size === 0) {
     this.listeners.delete(event);
 // Prevent memory leaks
 destroy() {
  this.listeners.clear();
// WeakMap for private data
const privateData = new WeakMap();
class User {
 constructor(name, ssn) {
  this.name = name;
  // Store sensitive data in WeakMap
  privateData.set(this, { ssn });
 getSSN() {
  return privateData.get(this).ssn;
// When user instance is garbage collected,
// WeakMap entry is automatically removed
```

```
// Memory-efficient DOM handling
const elementCleanup = new WeakMap();
const attachBehavior = (element, behavior) => {
 const cleanup = () => {
  // Cleanup logic
  element.removeEventListener('click', behavior);
 };
 element.addEventListener('click', behavior);
 elementCleanup.set(element, cleanup);
};
// Automatic cleanup when element is removed from DOM
const observer = new MutationObserver((mutations) => {
 mutations.forEach((mutation) => {
  mutation.removedNodes.forEach((node) => {
   if (node.nodeType === Node.ELEMENT NODE) {
    const cleanup = elementCleanup.get(node);
    if (cleanup) {
     cleanup();
     elementCleanup.delete(node);
   }
  });
 });
});
```

8. Modern APIs & Tools

LocalStorage & SessionStorage

javascript

```
// Storage utility class
class StorageManager {
 static set(key, value, useSession = false) {
  const storage = useSession ? sessionStorage : localStorage;
  try {
   storage.setItem(key, JSON.stringify(value));
   return true;
  } catch (error) {
   console.error('Storage error:', error);
   return false;
 static get(key, useSession = false) {
  const storage = useSession ? sessionStorage : localStorage;
  try {
   const item = storage.getItem(key);
   return item ? JSON.parse(item) : null;
  } catch (error) {
   console.error('Storage retrieval error:', error);
   return null;
 }
 static remove(key, useSession = false) {
  const storage = useSession ? sessionStorage : localStorage;
  storage.removeItem(key);
 }
 static clear(useSession = false) {
  const storage = useSession ? sessionStorage : localStorage;
  storage.clear();
 // Storage with expiration
 static setWithExpiry(key, value, ttl) {
  const now = new Date();
  const item = {
   value: value,
   expiry: now.getTime() + ttl
  };
  this.set(key, item);
 static getWithExpiry(key) {
  const item = this.get(key);
```

```
if (!item) return null;

const now = new Date();
if (now.getTime() > item.expiry) {
    this.remove(key);
    return null;
}

return item.value;
}

// Usage examples

StorageManaget.set('user', { name: 'John', preferences: { theme: 'dark' } });
const user = StorageManager.get('user');

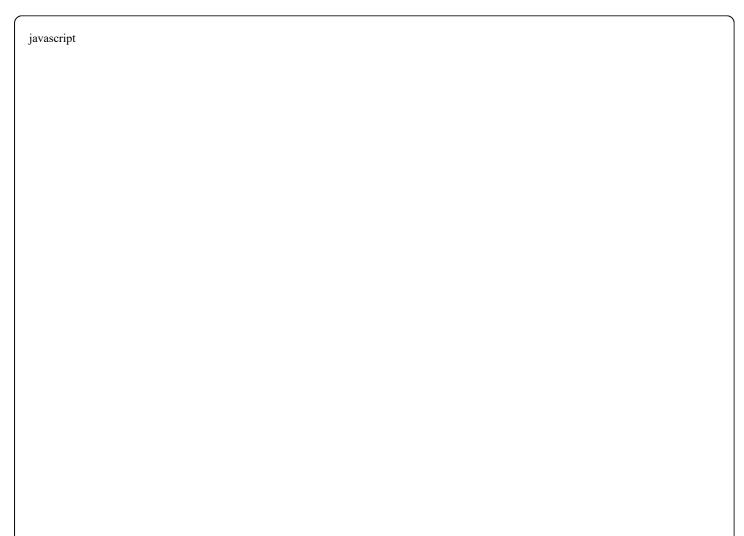
// Session-only storage

StorageManager.set('tempData', { sessionId: '123' }, true);

// Storage with expiration (1 hour)

StorageManager.setWithExpiry('apiCache', data, 60 * 60 * 1000);
```

IndexedDB



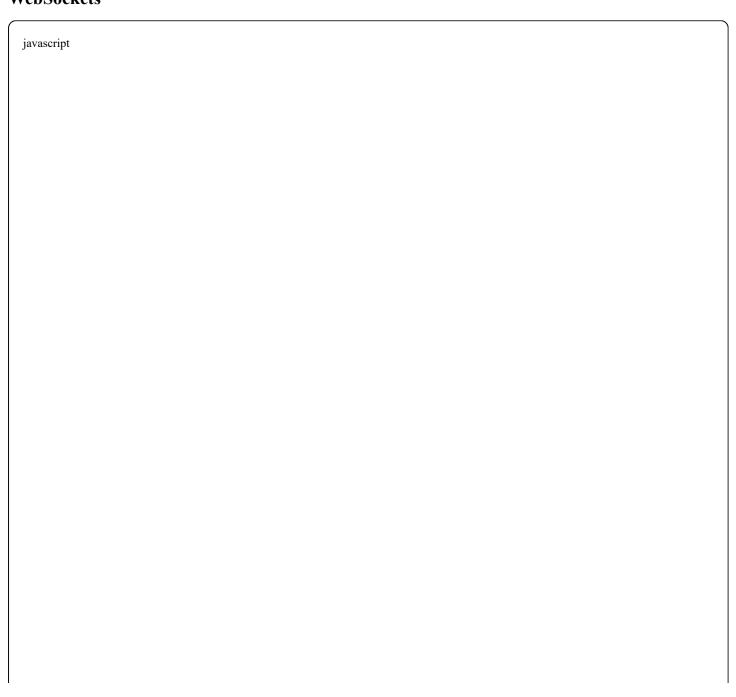
```
// IndexedDB wrapper
class IndexedDBManager {
 constructor(dbName, version = 1) {
  this.dbName = dbName;
  this.version = version;
  this.db = null;
 async init(stores = []) {
  return new Promise((resolve, reject) => {
   const request = indexedDB.open(this.dbName, this.version);
   request.onerror = () => reject(request.error);
   request.onsuccess = () \Rightarrow \{
     this.db = request.result;
    resolve(this.db);
   };
   request.onupgradeneeded = (event) => {
     const db = event.target.result;
     stores.forEach(({ name, keyPath, indexes = [] }) => {
      if (!db.objectStoreNames.contains(name)) {
       const store = db.createObjectStore(name, { keyPath });
       indexes.forEach(({ name: indexName, keyPath: indexKeyPath, unique = false }) => {
        store.createIndex(indexName, indexKeyPath, { unique });
       });
     });
   };
  });
 async add(storeName, data) {
  const transaction = this.db.transaction([storeName], 'readwrite');
  const store = transaction.objectStore(storeName);
  return new Promise((resolve, reject) => {
   const request = store.add(data);
   request.onsuccess = () => resolve(request.result);
   request.onerror = () => reject(request.error);
  });
 }
 async get(storeName, key) {
```

```
const transaction = this.db.transaction([storeName], 'readonly');
  const store = transaction.objectStore(storeName);
  return new Promise((resolve, reject) => {
   const request = store.get(key);
   request.onsuccess = () => resolve(request.result);
   request.onerror = () => reject(request.error);
  });
 }
 async getAll(storeName) {
  const transaction = this.db.transaction([storeName], 'readonly');
  const store = transaction.objectStore(storeName);
  return new Promise((resolve, reject) => {
   const request = store.getAll();
   request.onsuccess = () => resolve(request.result);
   request.onerror = () => reject(request.error);
  });
 }
 async update(storeName, data) {
  const transaction = this.db.transaction([storeName], 'readwrite');
  const store = transaction.objectStore(storeName);
  return new Promise((resolve, reject) => {
   const request = store.put(data);
   request.onsuccess = () => resolve(request.result);
   request.onerror = () => reject(request.error);
  });
 async delete(storeName, key) {
  const transaction = this.db.transaction([storeName], 'readwrite');
  const store = transaction.objectStore(storeName);
  return new Promise((resolve, reject) => {
   const request = store.delete(key);
   request.onsuccess = () => resolve(request.result);
   request.onerror = () => reject(request.error);
  });
// Usage
const dbManager = new IndexedDBManager('MyApp', 1);
```

```
await dbManager.init([
{
    name: 'users',
    keyPath: 'id',
    indexes: [
    { name: 'email', keyPath: 'email', unique: true },
    { name: 'name', keyPath: 'name' }
    ]
}
]);

// CRUD operations
await dbManager.add('users', { id: 1, name: 'John', email: 'john@example.com' });
const user = await dbManager.get('users', 1);
await dbManager.update('users', { id: 1, name: 'John Doe', email: 'john@example.com' });
```

WebSockets



```
// WebSocket manager with reconnection
class WebSocketManager {
 constructor(url, options = {}) {
  this.url = url;
  this.options = {
   reconnectInterval: 5000,
   maxReconnectAttempts: 5,
   ...options
  };
  this.ws = null;
  this.reconnectAttempts = 0;
  this.listeners = new Map();
 connect() {
  try {
   this.ws = new WebSocket(this.url);
   this.ws.onopen = (event) => {
     console.log('WebSocket connected');
     this.reconnectAttempts = 0;
     this.emit('connect', event);
    };
   this.ws.onmessage = (event) => {
     try {
      const data = JSON.parse(event.data);
      this.emit('message', data);
      // Handle specific message types
      if (data.type) {
       this.emit(data.type, data);
     } catch (error) {
      console.error('Error parsing WebSocket message:', error);
    };
   this.ws.onclose = (event) => {
     console.log('WebSocket disconnected');
     this.emit('disconnect', event);
     if (!event.wasClean && this.shouldReconnect()) {
      this.reconnect();
    };
```

```
this.ws.onerror = (error) => {
    console.error('WebSocket error:', error);
    this.emit('error', error);
  };
 } catch (error) {
  console.error('Failed to create WebSocket:', error);
  this.emit('error', error);
disconnect() {
 if (this.ws) {
  this.ws.close();
  this.ws = null;
send(data) {
 if (this.ws && this.ws.readyState === WebSocket.OPEN) {
  this.ws.send(JSON.stringify(data));
 } else {
  console.error('WebSocket is not connected');
on(event, callback) {
 if (!this.listeners.has(event)) {
  this.listeners.set(event, new Set());
 this.listeners.get(event).add(callback);
off(event, callback) {
 const eventListeners = this.listeners.get(event);
 if (eventListeners) {
  eventListeners.delete(callback);
emit(event, data) {
 const eventListeners = this.listeners.get(event);
 if (eventListeners) {
  eventListeners.forEach(callback => callback(data));
```

```
shouldReconnect() {
  return this.reconnectAttempts < this.options.maxReconnectAttempts;
 reconnect() {
  this.reconnectAttempts++;
  console.log(`Attempting to reconnect... (${this.reconnectAttempts})');
  setTimeout(() => {
   this.connect();
  }, this.options.reconnectInterval);
// Usage
const wsManager = new WebSocketManager('wss://api.example.com/ws');
wsManager.on('connect', () => {
console.log('Connected to server');
});
wsManager.on('message', (data) => {
 console.log('Received:', data);
});
wsManager.on('user_joined', (data) => {
 console.log(`User ${data.username} joined the chat`);
});
wsManager.connect();
// Send messages
wsManager.send({
 type: 'chat message',
 message: 'Hello, world!',
 timestamp: Date.now()
});
```

Service Workers (PWA)

javascript			

```
// sw.js - Service Worker
const CACHE NAME = 'my-app-v1';
const urlsToCache = [
 '/',
 '/styles/main.css',
 '/scripts/main.js',
 '/images/logo.png'
];
// Install event
self.addEventListener('install', (event) => {
 event.waitUntil(
  caches.open(CACHE_NAME)
   .then((cache) \Rightarrow {
     return cache.addAll(urlsToCache);
   })
 );
});
// Fetch event - Cache-first strategy
self.addEventListener('fetch', (event) => {
 event.respondWith(
  caches.match(event.request)
   .then((response) \Longrightarrow {
    // Return cached version or fetch from network
     return response || fetch(event.request);
   })
 );
});
// Activate event - Clean up old caches
self.addEventListener('activate', (event) => {
 event.waitUntil(
  caches.keys().then((cacheNames) => {
   return Promise.all(
     cacheNames.map((cacheName) => {
      if (cacheName !== CACHE NAME) {
       return caches.delete(cacheName);
     })
   );
  })
 );
});
// main.js - Register Service Worker
```

```
if ('serviceWorker' in navigator) {
 window.addEventListener('load', () => {
  navigator.serviceWorker.register('/sw.js')
   .then((registration) => {
     console.log('SW registered: ', registration);
   })
   .catch((registrationError) => {
     console.log('SW registration failed: ', registrationError);
    });
 });
}
// Push notifications
self.addEventListener('push', (event) => {
 const data = event.data ? event.data.json() : {};
 const options = {
  body: data.body,
  icon: '/images/icon-192x192.png',
  badge: '/images/badge-72x72.png',
  vibrate: [200, 100, 200],
  data: data.url
 };
 event.waitUntil(
  self.registration.showNotification(data.title, options)
 );
});
// Handle notification clicks
self.addEventListener('notificationclick', (event) => {
 event.notification.close();
 event.waitUntil(
  clients.openWindow(event.notification.data)
 );
});
```

Web Components

javascript

```
// Custom Element
class UserCard extends HTMLElement {
 constructor() {
  super();
  // Create shadow DOM
  this.attachShadow({ mode: 'open' });
  // Create template
  this.shadowRoot.innerHTML = `
   <style>
    :host {
      display: block;
      border: 1px solid #ccc;
      border-radius: 8px;
      padding: 16px;
      margin: 8px;
    .avatar {
      width: 50px;
      height: 50px;
      border-radius: 50%;
      object-fit: cover;
    .name {
      font-size: 1.2em;
      font-weight: bold;
      margin: 8px 0;
    .email {
      color: #666;
   </style>
   <div class="user-card">
    <img class="avatar" src="" alt="Avatar">
    <div class="name"></div>
    <div class="email"></div>
    <slot name="actions"></slot>
   </div>
```

```
connectedCallback() {
  this.render();
 static get observedAttributes() {
  return ['name', 'email', 'avatar'];
 attributeChangedCallback(name, oldValue, newValue) {
  if (oldValue !== newValue) {
   this.render();
 render() {
  const name = this.getAttribute('name') || ";
  const email = this.getAttribute('email') || ";
  const avatar = this.getAttribute('avatar') || '/images/default-avatar.png';
  this.shadowRoot.querySelector('.name').textContent = name;
  this.shadowRoot.querySelector('.email').textContent = email;
  this.shadowRoot.querySelector('.avatar').src = avatar;
 // Custom methods
 highlight() {
  this.style.backgroundColor = '#ffffcc';
  setTimeout(() => {
   this.style.backgroundColor = ";
  }, 1000);
 }
}
// Register the custom element
customElements.define('user-card', UserCard);
// Usage in HTML
<user-card
 name="John Doe"
 email="john@example.com"
 avatar="/images/john.jpg">
 <div slot="actions">
  <button>Follow</button>
  <button>Message</button>
 </div>
</user-card>
```

```
// Extend built-in elements
class EnhancedButton extends HTMLButtonElement {
   constructor() {
      super();
      this.addEventListener('click', this.handleClick);
   }

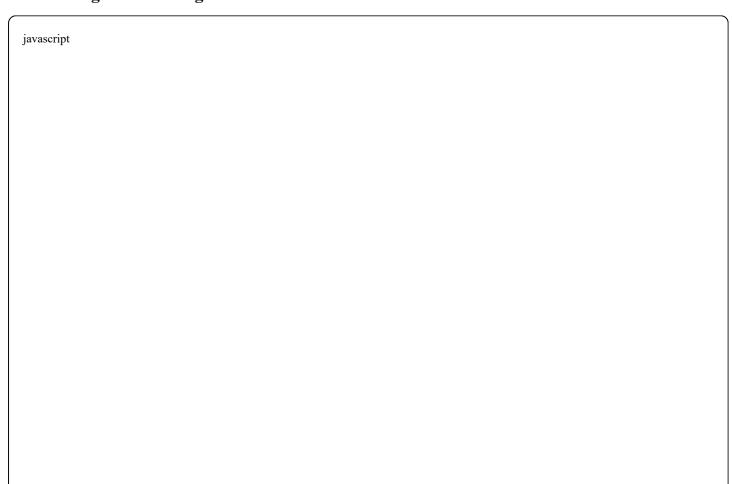
   handleClick() {
      this.classList.add('clicked');
      setTimeout(() => {
        this.classList.remove('clicked');
      }, 200);
   }
}

customElements.define('enhanced-button', EnhancedButton, { extends: 'button' });

// Usage: <button is="enhanced-button">Click me</button>
```

9. Performance Optimization

Debouncing & Throttling



```
// Debounce - Execute after delay, reset delay on new calls
const debounce = (func, delay) => {
 let timeoutId;
 return function debounced(...args) {
  clearTimeout(timeoutId);
  timeoutId = setTimeout(() => {
   func.apply(this, args);
  }, delay);
 };
};
// Throttle - Execute at most once per delay period
const throttle = (func, delay) => {
 let timeoutId;
 let lastExecTime = 0;
 return function throttled(...args) {
  const currentTime = Date.now();
  if (currentTime - lastExecTime > delay) {
   func.apply(this, args);
   lastExecTime = currentTime;
  } else if (!timeoutId) {
   timeoutId = setTimeout(() => {
     func.apply(this, args);
     lastExecTime = Date.now();
     timeoutId = null;
    }, delay - (currentTime - lastExecTime));
 };
};
// Practical examples
const searchInput = document.querySelector('#search');
const scrollContainer = document.querySelector('#container');
// Debounced search - wait for user to stop typing
const debouncedSearch = debounce(async (query) => {
 if (query.length > 2) {
  const results = await searchAPI(query);
  displayResults(results);
 }
}, 300);
```

```
searchInput.addEventListener('input', (e) => {
    debouncedSearch(e.target.value);
});

// Throttled scroll - limit scroll event handling
const throttledScroll = throttle(() => {
    const scrollTop = scrollContainer.scrollTop;
    const scrollHeight = scrollContainer.scrollHeight;
    const clientHeight = scrollContainer.clientHeight;

// Load more content when near bottom
if (scrollTop + clientHeight >= scrollHeight - 100) {
    loadMoreContent();
}
}, 100);
scrollContainer.addEventListener('scroll', throttledScroll);
```



```
// Image lazy loading with Intersection Observer
class LazyImageLoader {
 constructor(options = {}) {
  this.options = {
   root: null,
   rootMargin: '50px',
   threshold: 0.1,
   ...options
  };
  this.observer = new IntersectionObserver(
   this.handleIntersection.bind(this),
   this.options
  );
  this.init();
 init() {
  const lazyImages = document.querySelectorAll('img[data-src]');
  lazyImages.forEach(img => this.observer.observe(img));
 handleIntersection(entries) {
  entries.forEach(entry => {
   if (entry.isIntersecting) {
     this.loadImage(entry.target);
     this.observer.unobserve(entry.target);
   }
  });
 loadImage(img) {
  const src = img.dataset.src;
  const srcset = img.dataset.srcset;
  // Create new image to preload
  const imageLoader = new Image();
  imageLoader.onload = () => {
   img.src = src;
   if (srcset) img.srcset = srcset;
   img.classList.add('loaded');
  };
  imageLoader.onerror = () => {
```

```
img.classList.add('error');
  };
  imageLoader.src = src;
// Initialize lazy loading
new LazyImageLoader();
// Dynamic import for code splitting
const loadModule = async (moduleName) => {
 try {
  const module = await import(`./modules/${moduleName}.js`);
  return module.default;
 } catch (error) {
  console.error(`Failed to load module: ${moduleName}`, error);
  throw error;
 }
};
// Route-based code splitting
const router = {
 async navigate(route) {
  try {
   const RouteComponent = await loadModule(route);
   const component = new RouteComponent();
   component.render();
  } catch (error) {
   this.showError('Failed to load page');
};
// Lazy component loading
class ComponentLoader {
 static async loadComponent(componentName, container) {
  // Show loading state
  container.innerHTML = '<div class="loading">Loading...</div>';
  try {
   const Component = await import(`./components/${componentName}.js`);
   const instance = new Component.default();
   container.innerHTML = ";
   container.appendChild(instance.element);
  } catch (error) {
   container.innerHTML = '<div class="error">Failed to load component</div>';
```

}			
}			
}			

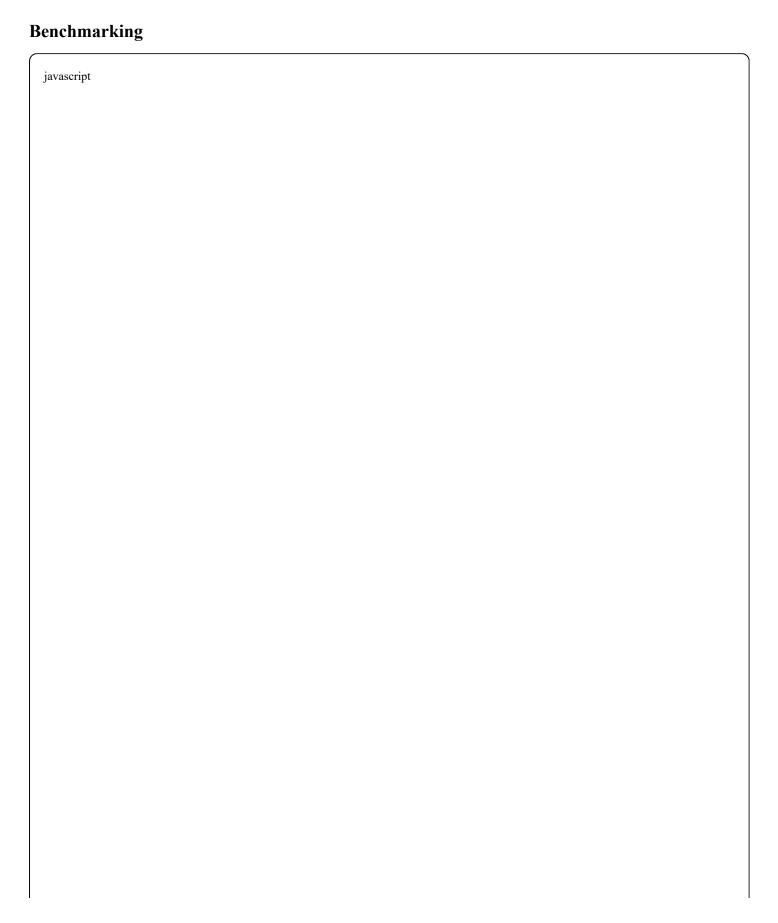
V8 Engine Optimization

javascript	

```
// V8 optimization tips
// 1. Use monomorphic functions (same parameter types)
// Good - consistent types
const addNumbers = (a, b) \Rightarrow \{
 return a + b; // Always receives numbers
};
// X Bad - polymorphic
const addAnything = (a, b) \Rightarrow \{
 return a + b; // Could receive different types
};
// 2. Initialize objects consistently
// Good - same hidden class
class Point {
 constructor(x, y) {
  this.x = x;
  this.y = y;
// X Bad - different hidden classes
const point1 = {};
point 1.x = 10;
point1.y = 20;
const point2 = \{\};
point2.y = 30; // Different property order
point2.x = 40;
// 3. Avoid deleting properties
// Good - set to null instead
const user = { name: 'John', email: 'john@example.com' };
user.email = null; // Instead of delete user.email
// 4. Use typed arrays for numerical data
// Good for performance
const coordinates = new Float32Array(1000);
for (let i = 0; i < coordinates.length; i += 2) {
 coordinates[i] = Math.random() * 100; //x
 coordinates[i + 1] = Math.random() * 100; // y
// 5. Optimize array operations
   Pre-allocate arrays when size is known
```

```
const createArray = (size) => {
 const arr = new Array(size);
 for (let i = 0; i < size; i++) {
  arr[i] = i * i;
 return arr;
};
// 6. Use efficient loops
const processArray = (arr) => {
 const length = arr.length; // Cache length
 // ✓ Fastest loop for simple operations
 for (let i = 0; i < length; i++) {
  arr[i] = arr[i] * 2;
 }
};
// 7. Avoid function calls in hot paths
// Good - inline calculations
const fastDistance = (x1, y1, x2, y2) \Rightarrow \{
 const dx = x2 - x1;
 const dy = y2 - y1;
return dx * dx + dy * dy; // Skip Math.sqrt if not needed
};
// 8. Use object pooling for frequent allocations
class ObjectPool {
 constructor(createFn, resetFn, initialSize = 10) {
  this.createFn = createFn;
  this.resetFn = resetFn;
  this.pool = [];
  // Pre-populate pool
  for (let i = 0; i < initialSize; i+++) {
   this.pool.push(this.createFn());
 }
 get() {
  return this.pool.length > 0 ? this.pool.pop() : this.createFn();
 release(obj) {
  this.resetFn(obj);
  this.pool.push(obj);
```

```
// Usage
const pointPool = new ObjectPool(
   () => ({ x: 0, y: 0 }),
   (point) => { point.x = 0; point.y = 0; }
);
```



```
// Performance measurement utilities
class PerformanceProfiler {
 static measure(name, fn) {
  const start = performance.now();
  const result = fn();
  const end = performance.now();
  console.log(`${name}: ${(end - start).toFixed(2)}ms`);
  return result;
 }
 static async measureAsync(name, asyncFn) {
  const start = performance.now();
  const result = await asyncFn();
  const end = performance.now();
  console.log(`${name}: ${(end - start).toFixed(2)}ms`);
  return result;
 }
 static benchmark(name, fn, iterations = 1000) {
  const times = [];
  // Warm up
  for (let i = 0; i < 10; i++) {
   fn();
  }
  // Measure
  for (let i = 0; i < iterations; i++) {
   const start = performance.now();
   fn();
   const end = performance.now();
   times.push(end - start);
  const avg = times.reduce((sum, time) => sum + time, 0) / times.length;
  const min = Math.min(...times);
  const max = Math.max(...times);
  console.log(`${name} Benchmark (${iterations} iterations):`);
  console.log(` Average: ${avg.toFixed(4)}ms`);
  console.log(` Min: ${min.toFixed(4)}ms`);
  console.log(` Max: ${max.toFixed(4)}ms`);
  return { avg, min, max, times };
```

```
}
}

// Memory usage tracking
class MemoryProfiler {
  static getMemoryUsage() {
   if (performance.memory) {
     return {
     used: performance.memory.usedJSHeapSize,
     total: performance
```