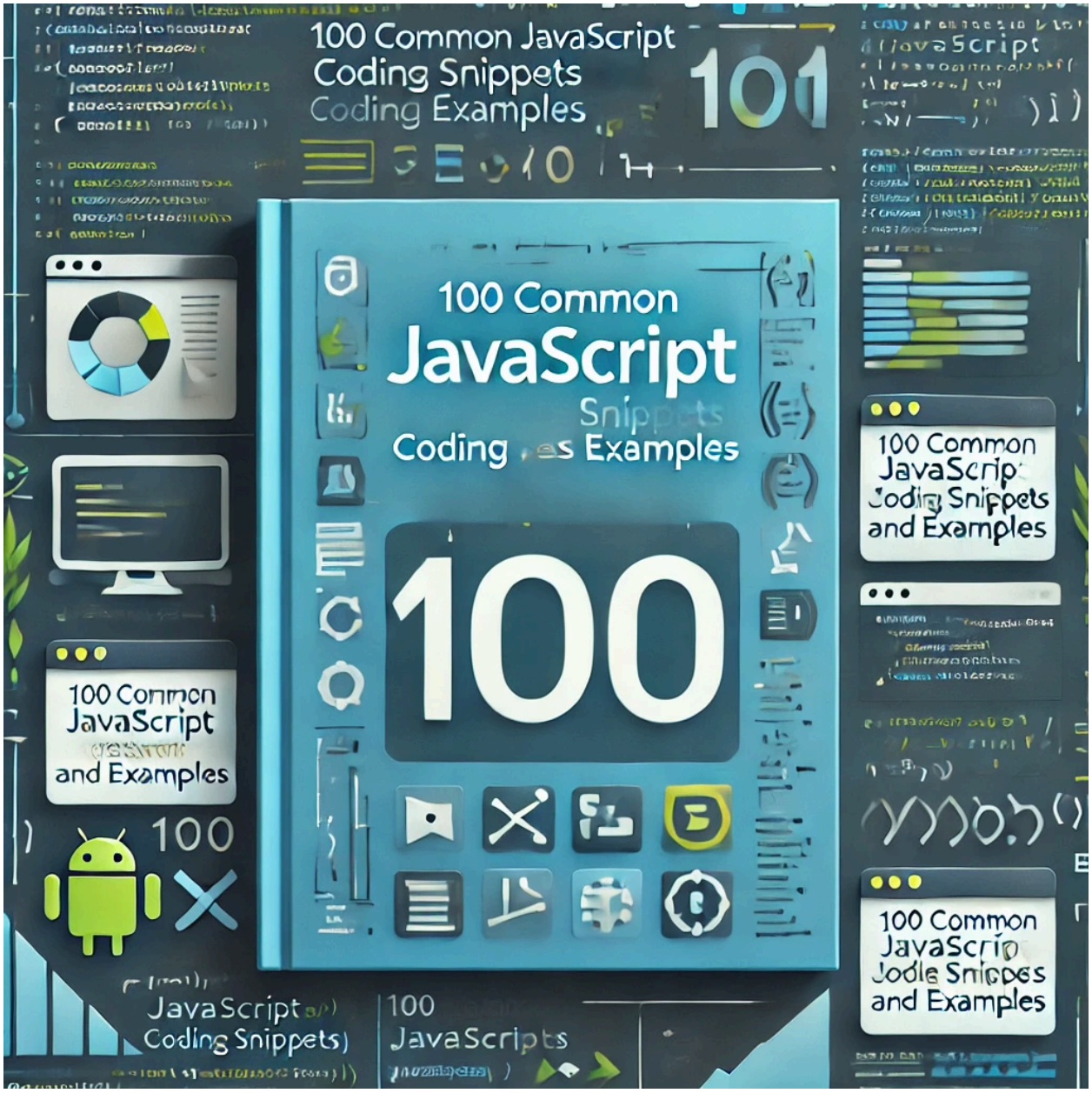
100+ Common JavaScript Code Snippets

In today's tech-driven world, mastering JavaScript is essential for developers looking to create dynamic, responsive, and efficient applications. "100 Common JavaScript Coding Snippets and Examples" is your comprehensive guide to essential JavaScript techniques, covering everything from core fundamentals to advanced programming concepts. This ebook is packed with practical exercises that will help you build a solid foundation and refine your skills through real-world examples.

Whether you're aiming to streamline data manipulation, optimize application performance, or understand the intricacies of asynchronous programming, these snippets offer a powerful toolkit. Each exercise is thoughtfully designed to illustrate a specific technique, breaking down complex topics like closures, async/await, DOM manipulation, and design patterns into manageable, hands-on examples. Through practice and application, you'll gain a deep understanding of JavaScript's capabilities, enabling you to write cleaner, faster, and more efficient code.

Ideal for beginners and seasoned developers alike, this collection serves as both a learning resource and a handy reference for tackling common JavaScript challenges. Get ready to level up your coding with this essential guide to JavaScript's most practical and powerful features.



Selecting DOM Elements

Use document.querySelector and document.querySelectorAll to select elements from the DOM.

```
// Select the first element with class 'my-class'
const element = document.querySelector('.my-class');
// Select all <div> elements
const elements = document.querySelectorAll('div');
```

Manipulating DOM Elements

Modify classes, styles, attributes, and content of DOM elements.

```
// Add a class
element.classList.add('active');
// Remove a class
element.classList.remove('active');
// Toggle a class
element.classList.toggle('hidden');
// Change styles
element.style.color = 'red';
// Change content
element.textContent = 'Hello, World!';
```

Adding Event Listeners

Attach events to elements to handle user interactions.

```
element.addEventListener('click', function(event) {
  console.log('Element clicked!');
});
```

Fetching Data from an API Using Fetch

Make HTTP requests to retrieve data from a server.

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

Working with Arrays (map, filter, reduce)

Perform transformations and computations on array data.

```
const numbers = [1, 2, 3, 4, 5];
// Map: Multiply each number by 2
const doubled = numbers.map(num => num * 2);
// Filter: Get even numbers
const evens = numbers.filter(num => num % 2 === 0);
// Reduce: Sum all numbers
const sum = numbers.reduce((total, num) => total + num, 0);
```

Promises and Async/Await Usage

Handle asynchronous operations effectively.

```
// Using Promises
function fetchData() {
  return new Promise((resolve, reject) => {
    // Asynchronous operation
  });
}
// Using async/await
async function getData() {
  try {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error('Error:', error);
  }
}
```

Manipulating localStorage/sessionStorage

Store and retrieve data in the browser's storage.

```
// Set item
localStorage.setItem('username', 'JohnDoe');
// Get item
const user = localStorage.getItem('username');
// Remove item
localStorage.removeItem('username');
// Clear all items
localStorage.clear();
```

Form Validation

Validate user input before processing.

```
const form = document.querySelector('form');
form.addEventListener('submit', function(event) {
  const input = document.querySelector('input').value;
  if (input === '') {
    alert('Input cannot be empty');
```

```
    event.preventDefault();
  }
});
```

Debounce and Throttle Functions

Control the rate at which a function is executed.

```
// Debounce function
function debounce(func, delay) {
  let timeoutId;
  return function(...args) {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => func.apply(this, args), delay);
  };
}
// Throttle function
function throttle(func, limit) {
  let inThrottle;
  return function(...args) {
    if (!inThrottle) {
      func.apply(this, args);
      inThrottle = true;
      setTimeout(() => (inThrottle = false), limit);
    }
  };
}
```

Cloning Objects (Deep Copy)

Create a deep copy of an object to prevent mutations.

```
// Using JSON methods (note: does not copy functions or Date objects)
const original = { a: 1, b: { c: 2 } };
const copy = JSON.parse(JSON.stringify(original));
// Using structuredClone (modern browsers)
const deepCopy = structuredClone(original);
```

Working with Dates (Date Object)

Manage and manipulate date and time.

```
const now = new Date();
```

```
const specificDate = new Date('2023-10-05');
const day = now.getDate();
const month = now.getMonth(); // 0-based index
const year = now.getFullYear();
```

Regular Expressions

Perform pattern matching and text manipulation.

```
const regex = /hello/i;
const str = 'Hello World';
const result = regex.test(str); // true
const match = str.match(regex); // ['Hello']
```

Sorting Arrays

Sort arrays numerically or alphabetically.

```
const numbers = [3, 1, 4, 1, 5, 9];
// Ascending order
numbers.sort((a, b) => a - b);
// Descending order
numbers.sort((a, b) => b - a);
```

Generating Random Numbers

Generate random numbers for various purposes.

```
// Random number between 0 and 1
const randomNum = Math.random();
// Random integer between min and max
function getRandomInt(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

Parsing and Stringifying JSON

Convert between JSON strings and JavaScript objects.

```
const jsonString = '{"name":"John","age":30}';
const obj = JSON.parse(jsonString);
const newJsonString = JSON.stringify(obj);
```

Using setTimeout and setInterval

Schedule code execution after a delay or at intervals.

```
// setTimeout
setTimeout(() => {
  console.log('This runs after 1 second');
}, 1000);
// setInterval
const intervalId = setInterval(() => {
  console.log('This runs every 2 seconds');
}, 2000);
// To stop the interval
clearInterval(intervalId);
```

Error Handling with try/catch

Handle exceptions and errors gracefully.

```
try {
  // Code that may throw an error
  nonExistentFunction();
} catch (error) {
  console.error('An error occurred:', error.message);
} finally {
  console.log('This runs regardless of the try/catch result');
}
```

Using Arrow Functions

Write concise function expressions.

```
const add = (a, b) => a + b;
// Implicit return for single expressions
const square = x => x * x;
```

Destructuring Assignment

Extract data from arrays or objects into variables.

```
const obj = { x: 1, y: 2 };
const { x, y } = obj;
const arr = [1, 2, 3];
const [first, second, third] = arr;
```

Spread and Rest Operators

Expand or collect elements in arrays and function arguments.

```
// Spread operator
const arr1 = [1, 2];
const arr2 = [...arr1, 3, 4]; // [1, 2, 3, 4]
// Rest operator
function sum(...numbers) {
  return numbers.reduce((total, num) => total + num, 0);
}
```

Creating and Using Classes (ES6 Classes)

Implement object-oriented programming concepts.

```
class Person {
  constructor(name) {
    this.name = name;
  }
  greet() {
    console.log(`Hello, my name is ${this.name}`);
  }
}
const john = new Person('John');
john.greet(); // "Hello, my name is John"
```

Module Imports and Exports (ES6 Modules)

Organize code into reusable modules.

```
// In math.js
export function add(a, b) {
  return a + b;
}
// In main.js
import { add } from './math.js';
console.log(add(2, 3)); // 5
```

Prototypal Inheritance

Implement inheritance using prototypes.

```
function Animal(name) {
```

```
  this.name = name;
}
Animal.prototype.speak = function() {
  console.log(`${this.name} makes a noise.`);
};
function Dog(name) {
  Animal.call(this, name);
}
Dog.prototype = Object.create(Animal.prototype);
Dog.prototype.constructor = Dog;
Dog.prototype.speak = function() {
  console.log(`${this.name} barks.`);
};
const dog = new Dog('Rex');
dog.speak(); // "Rex barks."
```

Closures

Preserve data in the scope where a function was created.

```
function makeAdder(x) {
  return function(y) {
    return x + y;
  };
}
const add5 = makeAdder(5);
console.log(add5(2)); // 7
```

String Manipulation (split, replace, etc.)

Perform common operations on strings.

```
const str = 'Hello World';
// Split string into array
const words = str.split(' '); // ['Hello', 'World']
// Replace substring
const newStr = str.replace('World', 'JavaScript'); // 'Hello JavaScript'
// Convert to uppercase
const upperStr = str.toUpperCase(); // 'HELLO WORLD'
// Trim whitespace
const trimmedStr = '  Hello World  '.trim(); // 'Hello World'
```

Implementing Custom Events

Create and dispatch custom events in the DOM.

```
// Create a new custom event
const event = new CustomEvent('myCustomEvent', { detail: { message: 'Hello World' }
});
// Listen for the event
document.addEventListener('myCustomEvent', function(e) {
  console.log(e.detail.message); // Outputs: Hello World
});
// Dispatch the event
document.dispatchEvent(event);
```

Using the Geolocation API

Access the user's geographical location (with permission).

```
if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(
    position => {
      console.log('Latitude:', position.coords.latitude);
      console.log('Longitude:', position.coords.longitude);
    },
    error => {
      console.error('Error getting location:', error);
    }
  );
} else {
  console.error('Geolocation is not supported by this browser.');
}
```

Drawing with the Canvas API

Create graphics using the <canvas> element.

```
const canvas = document.getElementById('myCanvas');
const ctx = canvas.getContext('2d');
// Draw a rectangle
ctx.fillStyle = '#FF0000';
ctx.fillRect(20, 20, 150, 100);
```

Drag and Drop Functionality

Enable draggable elements and handle drop events.

```
<!-- HTML -->
<div id="dragItem" draggable="true">Drag me</div>
<div id="dropZone">Drop here</div>
// JavaScript
const dragItem = document.getElementById('dragItem');
const dropZone = document.getElementById('dropZone');
dragItem.addEventListener('dragstart', function(e) {
  e.dataTransfer.setData('text/plain', 'This text may be dragged');
});
dropZone.addEventListener('dragover', function(e) {
  e.preventDefault();
});
dropZone.addEventListener('drop', function(e) {
  e.preventDefault();
  const data = e.dataTransfer.getData('text/plain');
  dropZone.textContent = data;
});
```

Handling Keyboard Events

Respond to keyboard inputs from the user.

```
document.addEventListener('keydown', function(event) {
  console.log(`Key pressed: ${event.key}`);
});
```

Using Web Workers

Run scripts in background threads.

```
// worker.js (Web Worker script)
self.addEventListener('message', function(e) {
  const result = e.data * 2;
  self.postMessage(result);
});
// Main script
const worker = new Worker('worker.js');
worker.postMessage(10);
worker.addEventListener('message', function(e) {
```

```
  console.log('Result from worker:', e.data); // Outputs: 20
});
```

Animating with requestAnimationFrame

Create smooth animations.

```
const box = document.getElementById('box');
let position = 0;
function animate() {
  position += 1;
  box.style.left = position + 'px';
  if (position < 300) {
    requestAnimationFrame(animate);
  }
}
requestAnimationFrame(animate);
```

Accessing and Setting Cookies

Manage cookies in the browser.

```
// Set a cookie
document.cookie = 'username=JohnDoe; expires=Fri, 31 Dec 2024 23:59:59 GMT;
path=/';
// Get cookies
const cookies = document.cookie;
console.log(cookies);
```

Using the Clipboard API

Copy text to the clipboard.

```
navigator.clipboard.writeText('Copy this text')
  .then(() => {
    console.log('Text copied to clipboard');
  })
  .catch(err => {
    console.error('Error copying text:', err);
  });
```

Implementing WebSockets

Establish a persistent connection with a server.

```
const socket = new WebSocket('wss://echo.websocket.org');
socket.addEventListener('open', function() {
  socket.send('Hello Server!');
});
socket.addEventListener('message', function(event) {
  console.log('Message from server:', event.data);
});
```

Parsing and Manipulating URLs

Work with URL components.

```
const url = new URL('https://example.com?param1=value1&param2=value2');
// Get query parameters
console.log(url.searchParams.get('param1')); // Outputs: value1
// Add a new parameter
url.searchParams.append('param3', 'value3');
console.log(url.toString()); // Outputs updated URL
```

Generators and Iterators

Create generator functions for custom iteration.

```
function* numberGenerator() {
  let i = 0;
  while (true) {
    yield i++;
  }
}
const gen = numberGenerator();
console.log(gen.next().value); // 0
console.log(gen.next().value); // 1
console.log(gen.next().value); // 2
```

Using Symbols

Create unique identifiers.

```
const sym1 = Symbol('identifier');
const sym2 = Symbol('identifier');
```

```
console.log(sym1 === sym2); // false
```

Working with Sets and Maps

Store unique values and key-value pairs.

```
// Set
const mySet = new Set([1, 2, 3, 3]);
mySet.add(4);
console.log(mySet.has(3)); // true
// Map
const myMap = new Map();
myMap.set('key1', 'value1');
console.log(myMap.get('key1')); // 'value1'
```

Using WeakMap and WeakSet

Hold weak references to objects.

```
const weakMap = new WeakMap();
let obj = {};
weakMap.set(obj, 'Some value');
// When 'obj' is garbage collected, the entry is removed from the WeakMap
obj = null;
```

Creating Proxies

Intercept and customize operations on objects.

```
const target = {};
const handler = {
  get: function(obj, prop) {
    return prop in obj ? obj[prop] : 'Property does not exist';
  },
};
const proxy = new Proxy(target, handler);
proxy.a = 1;
console.log(proxy.a); // 1
console.log(proxy.b); // 'Property does not exist'
```

Using the Reflect API

Perform object operations.

```
const obj = { x: 1 };
Reflect.set(obj, 'y', 2);
console.log(Reflect.get(obj, 'y')); // 2
```

Implementing a Simple Promise

Create a custom promise.

```
const myPromise = new Promise((resolve, reject) => {
  const success = true;
  if (success) {
    resolve('Operation successful');
  } else {
    reject('Operation failed');
  }
});
myPromise
  .then(message => console.log(message))
  .catch(error => console.error(error));
```

Creating Polyfills

Add support for features in older browsers.

```
// Polyfill for Array.includes
if (!Array.prototype.includes) {
  Array.prototype.includes = function(value) {
    return this.indexOf(value) !== -1;
  };
}
```

Feature Detection

Check if a feature is available before using it.

```
if ('geolocation' in navigator) {
  // Use geolocation
} else {
  console.error('Geolocation is not supported');
}
```

Using Template Literals

Create strings with embedded expressions.

```
const name = 'John';
const greeting = `Hello, ${name}!`;
console.log(greeting); // 'Hello, John!'
```

Tagged Template Literals

Manipulate template literals with functions.

```
function tag(strings, ...values) {
  return strings.raw[0];
}
const message = tag`Hello\nWorld`;
console.log(message); // 'Hello\nWorld'
```

Default Function Parameters

Assign default values to function parameters.

```
function greet(name = 'Guest') {
  console.log(`Hello, ${name}`);
}
greet(); // 'Hello, Guest'
greet('Alice'); // 'Hello, Alice'
```

Short-Circuit Evaluation

Use logical operators to set default values.

```
const value = null;
const result = value || 'Default Value';
console.log(result); // 'Default Value'
```

Nullish Coalescing Operator

Handle null or undefined values.

```
const value = null;
const result = value ?? 'Default Value';
console.log(result); // 'Default Value'
```

Optional Chaining

Safely access nested object properties.

```
const user = { profile: { name: 'John' } };
const username = user.profile?.name;
console.log(username); // 'John'
const age = user.profile?.age;
console.log(age); // undefined
```

Using the History API

Manipulate the browser's session history.

```
// Add a new history entry
history.pushState({ page: 1 }, 'Title', '?page=1');
// Listen for popstate event
window.addEventListener('popstate', function(event) {
  console.log('Location changed:', document.location.href);
});
```

Abortable Fetch Requests

Cancel fetch requests using AbortController.

```
const controller = new AbortController();
const signal = controller.signal;
fetch('https://api.example.com/data', { signal })
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(err => {
    if (err.name === 'AbortError') {
      console.log('Fetch aborted');
    } else {
      console.error('Fetch error:', err);
    }
  });
// Abort the fetch request
controller.abort();
```

Handling Files with the File API

Read files selected by the user.

```
<!-- HTML -->
<input type="file" id="fileInput" />
// JavaScript
const fileInput = document.getElementById('fileInput');
fileInput.addEventListener('change', function(event) {
  const file = event.target.files[0];
  const reader = new FileReader();
  reader.onload = function(e) {
    console.log('File content:', e.target.result);
  };
  reader.readAsText(file);
});
```

Observing DOM Changes with MutationObserver

Monitor changes to the DOM.

```
const targetNode = document.getElementById('content');
const config = { childList: true, subtree: true };
const callback = function(mutationsList) {
  for (const mutation of mutationsList) {
    if (mutation.type === 'childList') {
      console.log('A child node has been added or removed.');
    }
  }
};
const observer = new MutationObserver(callback);
observer.observe(targetNode, config);
// To stop observing
// observer.disconnect();
```

Lazy Loading with IntersectionObserver

Load content as it enters the viewport.

```
const images = document.querySelectorAll('img[data-src]');
const observer = new IntersectionObserver(entries => {
  entries.forEach(entry => {
    if (entry.isIntersecting) {
      const img = entry.target;
      img.src = img.dataset.src;
```

```
      observer.unobserve(img);
    }
  });
});
images.forEach(img => observer.observe(img));
```

Using ResizeObserver

Detect changes to the size of an element.

```
const box = document.getElementById('box');
const resizeObserver = new ResizeObserver(entries => {
  for (const entry of entries) {
    console.log('Element size changed:', entry.contentRect);
  }
});
resizeObserver.observe(box);
```

Creating Custom Elements (Web Components)

Define new HTML elements.

```
class MyElement extends HTMLElement {
  constructor() {
    super();
    this.innerHTML = '<p>Hello from custom element!</p>';
  }
}
customElements.define('my-element', MyElement);
// Usage in HTML:
// <my-element></my-element>
```

Using Shadow DOM

Encapsulate styles and markup.

```
class ShadowElement extends HTMLElement {
  constructor() {
    super();
    const shadow = this.attachShadow({ mode: 'open' });
    shadow.innerHTML = `<style>p { color: red; }</style><p>Shadow DOM Content</p>`;
  }
}
```

```
customElements.define('shadow-element', ShadowElement);
// Usage in HTML:
// <shadow-element></shadow-element>
```

Implementing the Singleton Pattern

Ensure a class has only one instance.

```
const Singleton = (function() {
  let instance;
  function createInstance() {
    return new Object('I am the instance');
  }
  return {
    getInstance: function() {
      if (!instance) {
        instance = createInstance();
      }
      return instance;
    },
  };
})();
const instance1 = Singleton.getInstance();
const instance2 = Singleton.getInstance();
console.log(instance1 === instance2); // true
```

Function Currying

Transform functions to take arguments one at a time.

```
function curry(f) {
  return function(a) {
    return function(b) {
      return f(a, b);
    };
  };
}
function add(a, b) {
  return a + b;
}
const curriedAdd = curry(add);
```

```
console.log(curriedAdd(2)(3)); // 5
```

Memoization

Cache function results for optimization.

```
function memoize(fn) {
  const cache = {};
  return function(...args) {
    const key = JSON.stringify(args);
    if (cache[key]) {
      return cache[key];
    } else {
      const result = fn(...args);
      cache[key] = result;
      return result;
    }
  };
}
const factorial = memoize(function(n) {
  if (n <= 1) return 1;
  return n * factorial(n - 1);
});
console.log(factorial(5)); // 120
```

Event Delegation

Handle events efficiently for multiple elements.

```
document.getElementById('parent').addEventListener('click', function(event) {
  if (event.target && event.target.matches('button.className')) {
    console.log('Button clicked:', event.target);
  }
});
```

Using the Content Security Policy (CSP)

Control resources the user agent is allowed to load.

```
<!-- In the HTML head -->
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'self';">
```

Understanding Cross-Origin Resource Sharing (CORS)

Make cross-origin requests in compliance with CORS.

```
fetch('https://api.example.com/data', {
  mode: 'cors',
})
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

Using IndexedDB

Store large amounts of data locally.

```
const request = indexedDB.open('MyDatabase', 1);
request.onupgradeneeded = function(event) {
  const db = event.target.result;
  const objectStore = db.createObjectStore('customers', { keyPath: 'id' });
  objectStore.createIndex('name', 'name', { unique: false });
};
request.onsuccess = function(event) {
  const db = event.target.result;
  const transaction = db.transaction(['customers'], 'readwrite');
  const objectStore = transaction.objectStore('customers');
  objectStore.add({ id: 1, name: 'John Doe' });
};
```

Using Promise.all

Handle multiple promises concurrently.

```
const promise1 = Promise.resolve(3);
const promise2 = 42;
const promise3 = new Promise((resolve) => setTimeout(resolve, 100, 'foo'));
Promise.all([promise1, promise2, promise3]).then(values => {
  console.log(values); // [3, 42, 'foo']
});
```

Async Iterators

Iterate over data asynchronously.

```
async function* asyncGenerator() {
  for (let i = 0; i < 3; i++) {
    await new Promise(resolve => setTimeout(resolve, 1000));
    yield i;
```

```
  }
}
(async function() {
  for await (const num of asyncGenerator()) {
    console.log(num); // 0, 1, 2 (with delays)
  }
})();
```

Async Generators

Combine async functions with generators.

```
async function* fetchGenerator(urls) {
  for (const url of urls) {
    const response = await fetch(url);
    const data = await response.json();
    yield data;
  }
}
const urls = ['https://api.example.com/data1', 'https://api.example.com/data2'];
(async function() {
  for await (const data of fetchGenerator(urls)) {
    console.log(data);
  }
})();
```

Working with URLSearchParams

Manipulate query parameters.

```
const params = new URLSearchParams('foo=1&bar=2');
// Get a parameter
console.log(params.get('foo')); // '1'
// Set a parameter
params.set('baz', 3);
// Iterate over parameters
for (const [key, value] of params) {
  console.log(`${key} = ${value}`);
}
```

Using FormData

Construct form data for XMLHttpRequest or fetch.

```
const formData = new FormData();
formData.append('username', 'JohnDoe');
formData.append('file', fileInput.files[0]);
fetch('https://example.com/upload', {
  method: 'POST',
  body: formData,
})
  .then(response => response.json())
  .then(result => console.log('Success:', result))
  .catch(error => console.error('Error:', error));
```

Working with Blobs and FileReader

Read binary data.

```
const blob = new Blob(['Hello, world!'], { type: 'text/plain' });
const reader = new FileReader();
reader.onload = function() {
  console.log(reader.result); // 'Hello, world!'
};
reader.readAsText(blob);
```

Typed Arrays

Handle binary data.

```
const buffer = new ArrayBuffer(16);
const view = new Uint32Array(buffer);
view[0] = 123456;
console.log(view[0]); // 123456
```

Implementing Recursion

Create functions that call themselves.

```
function factorial(n) {
  if (n <= 1) return 1;
  return n * factorial(n - 1);
}
console.log(factorial(5)); // 120
```

Understanding Tail Call Optimization

Optimize recursive functions.

```
function factorial(n, acc = 1) {
  if (n <= 1) return acc;
  return factorial(n - 1, n * acc);
}
console.log(factorial(5)); // 120
```

Using Promise.race

Handle the first settled promise among multiple promises.

```
const promise1 = new Promise(resolve => setTimeout(resolve, 500, 'First'));
const promise2 = new Promise(resolve => setTimeout(resolve, 300, 'Second'));
const promise3 = new Promise(resolve => setTimeout(resolve, 100, 'Third'));
Promise.race([promise1, promise2, promise3]).then(value => {
  console.log(value); // Outputs: 'Third'
});
```

Debouncing User Input

Delay processing of user input until the user stops typing.

```
const input = document.getElementById('search');
const debounce = (func, delay) => {
  let timeout;
  return function(...args) {
    clearTimeout(timeout);
    timeout = setTimeout(() => func.apply(this, args), delay);
  };
};
input.addEventListener('input', debounce(function(event) {
  console.log('User typed:', event.target.value);
}, 500));
```

Copying Text to Clipboard

Programmatically copy text to the clipboard.

```
function copyText(text) {
  const textarea = document.createElement('textarea');
  textarea.value = text;
```

```
  document.body.appendChild(textarea);
  textarea.select();
  document.execCommand('copy');
  document.body.removeChild(textarea);
  console.log('Text copied to clipboard');
}
copyText('Hello, World!');
```

Formatting Numbers with Commas

Format numbers with commas as thousand separators.

```
function formatNumber(num) {
  return num.toString().replace(/\B(?=(\d{3})+(?!\d))/g, ',');
}
console.log(formatNumber(1234567)); // Outputs: '1,234,567'
```

Generating UUIDs

Create universally unique identifiers.

```
function generateUUID() {
  return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
    const r = (Math.random() * 16) | 0;
    const v = c === 'x' ? r : (r & 0x3) | 0x8;
    return v.toString(16);
  });
}
console.log(generateUUID());
```

Detecting Mobile Devices

Check if the user is on a mobile device.

```
function isMobile() {
  return /Mobi|Android/i.test(navigator.userAgent);
}
console.log('Is mobile:', isMobile());
```

Implementing a Simple Router

Manage client-side routing.

```
function router() {
  const routes = {
    '/': () => console.log('Home'),
```

```
    '/about': () => console.log('About'),
    '/contact': () => console.log('Contact'),
  };
  const path = window.location.pathname;
  const route = routes[path] || (() => console.log('404 Not Found'));
  route();
}
window.addEventListener('popstate', router);
router();
```

Image Lazy Loading

Defer loading of images until they are needed.

```
<!-- HTML -->
<img data-src="image.jpg" alt="Lazy Loaded Image">
const images = document.querySelectorAll('img[data-src]');
const lazyLoad = () => {
  images.forEach(img => {
    const rect = img.getBoundingClientRect();
    if (rect.top < window.innerHeight) {
      img.src = img.dataset.src;
      img.removeAttribute('data-src');
    }
  });
};
document.addEventListener('scroll', lazyLoad);
window.addEventListener('load', lazyLoad);
```

Flattening Nested Arrays

Convert nested arrays into a single-level array.

```
const nestedArray = [1, [2, [3, [4]], 5]];
// Using flat method
const flatArray = nestedArray.flat(Infinity);
console.log(flatArray); // [1, 2, 3, 4, 5]
// Using recursion
function flatten(arr) {
  return arr.reduce((acc, val) => Array.isArray(val) ? acc.concat(flatten(val)) :
acc.concat(val), []);
```

```
}
console.log(flatten(nestedArray)); // [1, 2, 3, 4, 5]
```

Deep Cloning Objects

Clone objects including nested properties.

```
function deepClone(obj) {
  if (obj === null || typeof obj !== 'object') return obj;
  if (obj instanceof Date) return new Date(obj.getTime());
  if (obj instanceof Array) return obj.map(deepClone);
  const clonedObj = {};
  for (let key in obj) {
    if (obj.hasOwnProperty(key)) {
      clonedObj[key] = deepClone(obj[key]);
    }
  }
  return clonedObj;
}
const original = { a: 1, b: { c: 2 } };
const copy = deepClone(original);
```

Parsing Query Parameters

Get query parameters from a URL.

```
function getQueryParams(url) {
  const params = {};
  const parser = new URL(url);
  for (const [key, value] of parser.searchParams.entries()) {
    params[key] = value;
  }
  return params;
}
const params = getQueryParams('https://example.com?page=2&sort=desc');
console.log(params); // { page: '2', sort: 'desc' }
```

Validating Email Addresses

Check if a string is a valid email.

```
function isValidEmail(email) {
  const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
```

```
  return regex.test(email);
}
console.log(isValidEmail('test@example.com')); // true
```

Shuffling an Array

Randomize the order of elements in an array.

```
function shuffle(array) {
  let currentIndex = array.length, temporaryValue, randomIndex;
  while (0 !== currentIndex) {
    randomIndex = Math.floor(Math.random() * currentIndex);
    currentIndex -= 1;
    temporaryValue = array[currentIndex];
    array[currentIndex] = array[randomIndex];
    array[randomIndex] = temporaryValue;
  }
  return array;
}
const arr = [1, 2, 3, 4, 5];
console.log(shuffle(arr));
```

Detecting Dark Mode Preference

Check if the user prefers dark mode.

```
const prefersDarkMode = window.matchMedia &&
window.matchMedia('(prefers-color-scheme: dark)').matches;
if (prefersDarkMode) {
  console.log('User prefers dark mode');
} else {
  console.log('User prefers light mode');
}
```

Calculating the Fibonacci Sequence

Generate Fibonacci numbers.

```
function fibonacci(n) {
  if (n <= 1) return n;
  return fibonacci(n - 1) + fibonacci(n - 2);
}
console.log(fibonacci(6)); // Outputs: 8
```

Implementing a Simple Map

Use an object as a simple key-value store.

```
const map = {};
// Set value
map['key'] = 'value';
// Get value
console.log(map['key']); // 'value'
// Check existence
console.log('key' in map); // true
```

Formatting Dates

Convert dates to readable strings.

```
const date = new Date();
// Locale string
console.log(date.toLocaleDateString());
// Custom format
function formatDate(date) {
  const day = date.getDate().toString().padStart(2, '0');
  const month = (date.getMonth() + 1).toString().padStart(2, '0');
  const year = date.getFullYear();
  return `${day}/${month}/${year}`;
}
console.log(formatDate(date)); // e.g., '05/10/2023'
```

Implementing Bubble Sort

Sort an array using the bubble sort algorithm.

```
function bubbleSort(arr) {
  let n = arr.length;
  for (let i = 0; i < n - 1; i++) {
    for (let j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
      }
    }
  }
  return arr;
}
```

```
console.log(bubbleSort([5, 3, 8, 4, 2])); // [2, 3, 4, 5, 8]
```

Understanding Event Bubbling and Capturing

Explore how events propagate in the DOM.

```
// HTML structure
// <div id="parent">
//   <div id="child">Click me</div>
// </div>
const parent = document.getElementById('parent');
const child = document.getElementById('child');
parent.addEventListener('click', () => console.log('Parent clicked'), false);
child.addEventListener('click', (e) => {
  console.log('Child clicked');
  e.stopPropagation(); // Stops bubbling
}, false);
```

Throttling Function Execution

Limit the rate at which a function can fire.

```
function throttle(func, limit) {
  let lastFunc;
  let lastRan;
  return function(...args) {
    const context = this;
    if (!lastRan) {
      func.apply(context, args);
      lastRan = Date.now();
    } else {
      clearTimeout(lastFunc);
      lastFunc = setTimeout(function() {
        if (Date.now() - lastRan >= limit) {
          func.apply(context, args);
          lastRan = Date.now();
        }
      }, limit - (Date.now() - lastRan));
    }
  };
}
```

```
window.addEventListener('resize', throttle(() => {
  console.log('Window resized');
}, 1000));
```

Implementing the Observer Pattern

Allow objects to notify other objects about changes.

```
class Subject {
  constructor() {
    this.observers = [];
  }
  subscribe(observer) {
    this.observers.push(observer);
  }
  unsubscribe(observer) {
    this.observers = this.observers.filter(obs => obs !== observer);
  }
  notify(data) {
    this.observers.forEach(observer => observer.update(data));
  }
}
class Observer {
  update(data) {
    console.log('Observer received data:', data);
  }
}
const subject = new Subject();
const observer1 = new Observer();
const observer2 = new Observer();
subject.subscribe(observer1);
subject.subscribe(observer2);
subject.notify('Hello Observers!');
```

Simple Publish/Subscribe Implementation

Implement the pub/sub pattern.

```
const pubsub = {
  events: {},
  subscribe: function(event, listener) {
```

```
    if (!this.events[event]) {
      this.events[event] = [];
    }
    this.events[event].push(listener);
  },
  publish: function(event, data) {
    if (this.events[event]) {
      this.events[event].forEach(listener => listener(data));
    }
  },
};
pubsub.subscribe('message', data => console.log('Received:', data));
pubsub.publish('message', 'Hello, Pub/Sub!');
```

Creating a Countdown Timer

Implement a simple countdown timer.

```
function startTimer(duration, display) {
  let timer = duration, minutes, seconds;
  const interval = setInterval(() => {
    minutes = Math.floor(timer / 60);
    seconds = timer % 60;
    display.textContent = `${minutes}:${seconds < 10 ? '0' : ''}${seconds}`;
    if (--timer < 0) {
      clearInterval(interval);
      display.textContent = 'Time\'s up!';
    }
  }, 1000);
}
const display = document.getElementById('timer');
startTimer(120, display); // Starts a 2-minute timer
```

Implementing Binary Search

Efficiently search a sorted array.

```
function binarySearch(arr, target) {
  let left = 0;
  let right = arr.length - 1;
  while (left <= right) {
```

```
    const mid = Math.floor((left + right) / 2);
    if (arr[mid] === target) return mid;
    else if (arr[mid] < target) left = mid + 1;
    else right = mid - 1;
  }
  return -1;
}
const arr = [1, 3, 5, 7, 9];
console.log(binarySearch(arr, 5)); // Outputs: 2
```

Implementing a Stack

Create a simple stack data structure.

```
class Stack {
  constructor() {
    this.items = [];
  }
  push(element) {
    this.items.push(element);
  }
  pop() {
    if (this.items.length === 0) return 'Underflow';
    return this.items.pop();
  }
  peek() {
    return this.items[this.items.length - 1];
  }
  isEmpty() {
    return this.items.length === 0;
  }
}
const stack = new Stack();
stack.push(1);
stack.push(2);
console.log(stack.pop()); // Outputs: 2
```

Implementing a Queue

Create a simple queue data structure.

```
class Queue {
  constructor() {
    this.items = [];
  }
  enqueue(element) {
    this.items.push(element);
  }
  dequeue() {
    if (this.isEmpty()) return 'Underflow';
    return this.items.shift();
  }
  front() {
    return this.items[0];
  }
  isEmpty() {
    return this.items.length === 0;
  }
}
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
console.log(queue.dequeue()); // Outputs: 1
```

Using Bitwise Operators

Perform operations at the bit level.

```
// Check if a number is even
function isEven(n) {
  return (n & 1) === 0;
}
console.log(isEven(4)); // true
```

Using Object.freeze

Make an object immutable.

```
const obj = { a: 1, b: 2 };
Object.freeze(obj);
obj.a = 3; // This will have no effect
console.log(obj.a); // Outputs: 1
```

Using Object.seal

Prevent adding or removing properties.

```
const obj = { a: 1, b: 2 };
Object.seal(obj);
obj.c = 3; // Cannot add new property
delete obj.a; // Cannot delete property
obj.b = 4; // Can modify existing properties
console.log(obj); // { a: 1, b: 4 }
```

Detecting User's Time Zone

Get the user's time zone.

```
const timeZone = Intl.DateTimeFormat().resolvedOptions().timeZone;
console.log('User\'s time zone:', timeZone);
```

Unit Testing with Jasmine

Write a simple unit test using Jasmine.

```
// Function to test
function add(a, b) {
  return a + b;
}
// Jasmine test
describe('Addition', function() {
```

```
  it('should add two numbers', function() {
    expect(add(1, 2)).toBe(3);
  });
});
```

Singleton Pattern with Classes

Ensure a class has only one instance.

```
class Singleton {
  constructor() {
    if (Singleton.instance) {
      return Singleton.instance;
    }
    this.data = 'Some data';
    Singleton.instance = this;
  }
}
const instance1 = new Singleton();
const instance2 = new Singleton();
console.log(instance1 === instance2); // true
```

Inheritance with Classes

Use `extends` for inheritance.

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  speak() {
    console.log(`${this.name} makes a noise.`);
  }
}
class Dog extends Animal {
  speak() {
    console.log(`${this.name} barks.`);
```

```
  }
}
const dog = new Dog('Rex');
dog.speak(); // 'Rex barks.'
```

Using new.target

Detect if a function was called with `new`.

```
function Person(name) {
  if (!new.target) {
    throw 'Person() must be called with new';
  }
  this.name = name;
}
// Correct usage
const person = new Person('John');
// Incorrect usage
// Person('John'); // Throws error
```

Simple Middleware Pattern

Chain functions together.

```
function middleware1(next) {
  console.log('Middleware 1 Start');
  next();
  console.log('Middleware 1 End');
}
function middleware2(next) {
  console.log('Middleware 2 Start');
  next();
  console.log('Middleware 2 End');
}
function finalHandler() {
  console.log('Final Handler');
}
```

```
function compose(middlewares) {
  return middlewares.reduceRight(
    (next, mw) => () => mw(next),
    finalHandler
  );
}
const composed = compose([middleware1, middleware2]);
composed();
```

Promise-Based Sleep Function

Pause execution in async functions.

```
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}
async function demo() {
  console.log('Wait for 1 second');
  await sleep(1000);
  console.log('1 second passed');
}
demo();
```

Using Function.prototype.bind

Create a new function with `this` bound to a specific value.

```
const person = {
  name: 'Alice',
  greet: function(greeting) {
    console.log(`${greeting}, I am ${this.name}`);
  },
};
const greet = person.greet.bind({ name: 'Bob' });
greet('Hello'); // 'Hello, I am Bob'
```

Function Composition

Combine multiple functions.

```
const compose = (...functions) => args => functions.reduceRight((arg, fn) => fn(arg), args);
const add = x => x + 1;
const multiply = x => x * 2;
const addThenMultiply = compose(multiply, add);
console.log(addThenMultiply(5)); // Outputs: 12
```

Fetch API with Credentials

Send cookies with fetch requests.

```
fetch('https://example.com/data', {
  credentials: 'include', // or 'same-origin'
})
  .then(response => response.json())
  .then(data => console.log(data));
```

Detecting Online/Offline Status

Respond to network status changes.

```
window.addEventListener('online', () => console.log('Back online'));
window.addEventListener('offline', () => console.log('You are offline'));
```

Handling Asynchronous Iterations

Use `for...of` with async functions.

```
async function processArray(array) {
  for (const item of array) {
    await doSomethingAsync(item);
  }
}
```

```
async function doSomethingAsync(item) {
  return new Promise(resolve => setTimeout(() => {
    console.log('Processed:', item);
    resolve();
  }, 1000));
}
processArray([1, 2, 3]);
```

Implementing Merge Sort

Sort an array using merge sort.

```
function mergeSort(arr) {
  if (arr.length <= 1) return arr;
  const mid = Math.floor(arr.length / 2);
  const left = mergeSort(arr.slice(0, mid));
  const right = mergeSort(arr.slice(mid));
  return merge(left, right);
}
function merge(left, right) {
  const result = [];
  while (left.length && right.length) {
    if (left[0] < right[0]) result.push(left.shift());
    else result.push(right.shift());
  }
  return result.concat(left, right);
}
console.log(mergeSort([5, 3, 8, 4, 2])); // [2, 3, 4, 5, 8]
```

Implementing Quick Sort

Sort an array using quick sort.

```
function quickSort(arr) {
  if (arr.length <= 1) return arr;
  const pivot = arr[arr.length - 1];
  const left = [];
```

```
  const right = [];
  for (let i = 0; i < arr.length - 1; i++) {
    if (arr[i] < pivot) left.push(arr[i]);
    else right.push(arr[i]);
  }
  return [...quickSort(left), pivot, ...quickSort(right)];
}
console.log(quickSort([5, 3, 8, 4, 2])); // [2, 3, 4, 5, 8]
```

Custom Promise.allSettled Polyfill

Handle multiple promises regardless of their outcome.

```
if (!Promise.allSettled) {
  Promise.allSettled = function(promises) {
    return Promise.all(promises.map(promise =>
      promise.then(
        value => ({ status: 'fulfilled', value }),
        reason => ({ status: 'rejected', reason })
      )
    ));
  };
}
const promises = [
  Promise.resolve(1),
  Promise.reject('Error'),
  Promise.resolve(3),
];
Promise.allSettled(promises).then(results => console.log(results));
```

Validating Palindromes

Check if a string is a palindrome.

```
function isPalindrome(str) {
  const cleanStr = str.replace(/[^A-Za-z0-9]/g, '').toLowerCase();
  return cleanStr === cleanStr.split('').reverse().join('');
```

```
}
console.log(isPalindrome('A man, a plan, a canal, Panama')); // true
```

Converting Callbacks to Promises

Wrap a callback-based function into a promise.

```
function readFilePromise(filePath) {
  return new Promise((resolve, reject) => {
    fs.readFile(filePath, 'utf8', (err, data) => {
      if (err) reject(err);
      else resolve(data);
    });
  });
}
readFilePromise('file.txt')
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

Custom Iterators

Define custom iteration behavior.

```
const myIterable = {
  [Symbol.iterator]: function* () {
    yield 1;
    yield 2;
    yield 3;
  },
};
for (const value of myIterable) {
  console.log(value); // 1, 2, 3
}
```

124. Detecting Browser Language

Get the user's preferred language.

```
const language = navigator.language || navigator.userLanguage;
```

```
console.log('Preferred language:', language);
```

Creating a Reusable Modal

Implement a simple modal dialog.

```
<!-- HTML -->
<div id="modal" class="modal">
  <div class="modal-content">
    <span id="closeBtn" class="close">&times;</span>
    <p>Modal content here...</p>
  </div>
</div>
/* CSS */
.modal { display: none; position: fixed; /* ... */ }
.modal-content { /* ... */ }
.close { /* ... */ }
// JavaScript
const modal = document.getElementById('modal');
const closeBtn = document.getElementById('closeBtn');
function showModal() {
  modal.style.display = 'block';
}
closeBtn.addEventListener('click', () => {
  modal.style.display = 'none';
});
window.addEventListener('click', (event) => {
  if (event.target === modal) {
    modal.style.display = 'none';
  }
});
// To open the modal
showModal();
```