Chapter 12

JavaScript Performance Optimization and Future Trends

**Common Sources of JavaScript Performance Bottlenecks**

**A. Inefficient DOM Manipulation**

* **Example:**

for (let i = 0; i < 1000; i++) {

let div = document.createElement('div');

document.body.appendChild(div); // Updates DOM on each iteration

}

**B. Blocking the Main Thread**

* **Example:** A complex calculation freezes the UI.

function heavyComputation() {

for (let i = 0; i < 1e9; i++) {} // Blocks the main thread

}

heavyComputation();

**C. Unoptimized Loops and Iterations**

* **Example:**

const largeArray = new Array(100000).fill(1);

largeArray.forEach((item) => {

// Inefficient operation

console.log(item \* 2);

});

**D. Memory Leaks**

* **Example:** Forgetting to remove event listeners.

function addClickListener() {

const button = document.getElementById('btn');

button.addEventListener('click', () => console.log('Clicked!'));

}

addClickListener(); // Listener persists even after `button` is removed

**E. Inefficient Event Handling**

* **Example:** Attaching individual listeners to a list of items.

document.querySelectorAll('.item').forEach((el) => {

el.addEventListener('click', () => console.log('Item clicked!'));

});

**Real-World Example: Optimizing a Web Application**

**Optimized Code:**

// Lazy loading articles

const loadArticles = async (page) => {

const response = await fetch(`/api/articles?page=${page}`);

const articles = await response.json();

renderArticles(articles);

};

// Debounced search

const searchInput = document.getElementById('search');

const debounce = (func, delay) => {

let timeout;

return (...args) => {

clearTimeout(timeout);

timeout = setTimeout(() => func(...args), delay);

};

};

searchInput.addEventListener(

'input',

debounce((event) => {

const query = event.target.value;

fetch(`/api/search?q=${query}`).then((res) => res.json()).then(renderArticles);

}, 300)

);

**Optimizing Loops**

**Inefficient:**

for (let i = 0; i < arr.length; i++) {

const value = expensiveFunction(arr[i]); // Expensive operation inside loop

console.log(value);

}

**Optimized:**

const expensiveValue = expensiveFunction(); // Run once outside the loop

for (let i = 0; i < arr.length; i++) {

console.log(expensiveValue);

}

**Loop Unrolling**

**Inefficient:**

for (let i = 0; i < arr.length; i++) {

process(arr[i]);

}

**Optimized (Loop Unrolling):**

for (let i = 0; i < arr.length; i += 2) {

process(arr[i]);

if (i + 1 < arr.length) process(arr[i + 1]); // Handle even cases

}

**Inefficient (forEach):**

arr.forEach(item => {

process(item);

});

**Optimized (for loop):**

for (let i = 0; i < arr.length; i++) {

process(arr[i]);

}

**Using for...of for Arrays**

**Optimized (for...of loop):**

for (const item of arr) {

process(item);

}

**Optimizing Function Calls**

**Inefficient:**

for (let i = 0; i < arr.length; i++) {

const result = expensiveFunction(i); // Repeated calls with the same argument

console.log(result);

}

**Optimized:**

const results = [];

for (let i = 0; i < arr.length; i++) {

if (!results[i]) results[i] = expensiveFunction(i); // Cache the result

console.log(results[i]);

}

**Memoization**

**Example:**

const memo = {};

function expensiveFunction(n) {

if (memo[n]) {

return memo[n]; // Return cached result if already computed

}

const result = n \* 100; // Expensive computation

memo[n] = result; // Store the result for future calls

return result;

}

**Throttling and Debouncing**

**Example (Debounce):**

let timeout;

function debounce(func, delay) {

clearTimeout(timeout);

timeout = setTimeout(func, delay);

}

**Example (Throttle):**

let lastCall = 0;

function throttle(func, delay) {

const now = new Date().getTime();

if (now - lastCall >= delay) {

lastCall = now;

func();

}

}

**Example: Lazy Loading Images**

**Complete Example:**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Lazy Loading Images Example</title>

  <style>

    /\* Basic styling for the images and page \*/

    body {

      font-family: Arial, sans-serif;

      padding: 20px;

      margin: 0;

      height: 2000px; /\* Ensures scrolling \*/

      background-color: #f4f4f4;

    }

    img {

      width: 100%;

      max-width: 600px;

      margin: 20px 0;

      display: block;

      height: auto;

    }

    .lazy {

      background-color: #e0e0e0;

      display: block;

      height: 200px; /\* Placeholder height \*/

      width: 100%;

    }

  </style>

</head>

<body>

  <h1>Lazy Loading Images</h1>

  <!-- Lazy-loaded images with data-src for actual image sources -->

  <img data-src="https://picsum.photos/600/400?text=Image+1" alt="Lazy Loaded Image 1" class="lazy">

  <img data-src="https://picsum.photos/600/400?text=Image+2" alt="Lazy Loaded Image 2" class="lazy">

  <img data-src="https://picsum.photos/600/400?text=Image+3" alt="Lazy Loaded Image 3" class="lazy">

  <img data-src="https://picsum.photos/600/400?text=Image+4" alt="Lazy Loaded Image 4" class="lazy">

  <img data-src="https://picsum.photos/600/400?text=Image+5" alt="Lazy Loaded Image 5" class="lazy">

  <img data-src="https://picsum.photos/600/400?text=Image+6" alt="Lazy Loaded Image 6" class="lazy">

  <script>

    // Select all the lazy-loaded images

    const lazyImages = document.querySelectorAll('img.lazy');

    // Create an IntersectionObserver instance

    const imageObserver = new IntersectionObserver((entries, observer) => {

      entries.forEach(entry => {

        if (entry.isIntersecting) {

          const image = entry.target;

          image.src = image.getAttribute('data-src'); // Replace placeholder with actual image source

          image.classList.remove('lazy');  // Remove the 'lazy' class (optional)

          observer.unobserve(image);  // Stop observing the image after it has loaded

        }

      });

    }, {

      threshold: 0.1 // Trigger when 10% of the image is visible

    });

    // Observe each lazy-loaded image

    lazyImages.forEach(image => {

      imageObserver.observe(image);  // Observe each image to track its visibility

    });

  </script>

</body>

</html>

**Example: Debouncing Search Input**

**HTML (debounce.html):**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Search with Debounce</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      margin: 20px;

      padding: 0;

    }

    #search {

      width: 100%;

      max-width: 400px;

      padding: 10px;

      font-size: 16px;

      margin-bottom: 20px;

      border: 1px solid #ccc;

      border-radius: 4px;

    }

    #results {

      margin-top: 10px;

    }

    #results p {

      background-color: #f4f4f4;

      padding: 10px;

      border: 1px solid #ddd;

      border-radius: 4px;

      margin: 5px 0;

    }

  </style>

</head>

<body>

  <h1>Search Example</h1>

  <input type="text" id="search" placeholder="Search...">

  <div id="results"></div>

  <script>

    const searchInput = document.getElementById('search');

    const resultsDiv = document.getElementById('results');

    // Function to fetch search results

    const fetchSearchResults = async (query) => {

      try {

        const response = await fetch(`https://jsonplaceholder.typicode.com/users?q=${query}`);

        if (!response.ok) throw new Error('Network response was not ok');

        const data = await response.json();

        renderResults(data);

      } catch (error) {

        console.error('Error fetching search results:', error);

        resultsDiv.innerHTML = '<p style="color: red;">Failed to fetch results. Please try again later.</p>';

      }

    };

    // Function to render search results

    const renderResults = (data) => {

      if (data.length === 0) {

        resultsDiv.innerHTML = '<p>No results found.</p>';

        return;

      }

      resultsDiv.innerHTML = data

        .map(item => `<p>${item.name} (${item.email})</p>`)

        .join('');

    };

    // Debounce function

    const debounce = (func, delay) => {

      let timeout;

      return (...args) => {

        clearTimeout(timeout);

        timeout = setTimeout(() => func(...args), delay);

      };

    };

    // Attach debounced event listener

    searchInput.addEventListener('input', debounce((event) => {

      const query = event.target.value.trim();

      if (query) {

        fetchSearchResults(query);

      } else {

        resultsDiv.innerHTML = ''; // Clear results if input is empty

      }

    }, 500)); // Wait for 500ms after the last keystroke before firing the function

  </script>

</body>

</html>

**Differences Between Lazy Loading and Debouncing**

**HTML (lazy.html):**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Lazy Loading and Debounced Search</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      padding: 20px;

    }

    #search {

      width: 100%;

      padding: 10px;

      margin-bottom: 20px;

      font-size: 16px;

    }

    #articles {

      display: grid;

      gap: 20px;

      grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));

    }

    .article {

      border: 1px solid #ccc;

      padding: 15px;

      border-radius: 5px;

      background: #f9f9f9;

    }

    .loading {

      text-align: center;

      margin-top: 20px;

    }

  </style>

</head>

<body>

  <h1>Lazy Loading and Debounced Search</h1>

  <input type="text" id="search" placeholder="Search articles..." />

  <div id="articles"></div>

  <div class="loading" id="loading">Loading...</div>

  <script src="app.js"></script>

</body>

</html>

**JavaScript(app.js):**

// Global variables

let currentPage = 1; // Track current page for lazy loading

let isLoading = false; // Prevent multiple simultaneous loads

// Fetch and render articles

const loadArticles = async (page) => {

  try {

    isLoading = true; // Set loading state

    document.getElementById('loading').style.display = 'block'; // Show loading indicator

    const response = await fetch(`https://jsonplaceholder.typicode.com/posts?\_page=${page}&\_limit=10`);

    if (!response.ok) throw new Error('Failed to fetch articles');

    const articles = await response.json();

    renderArticles(articles);

  } catch (error) {

    console.error('Error loading articles:', error);

  } finally {

    isLoading = false; // Reset loading state

    document.getElementById('loading').style.display = 'none'; // Hide loading indicator

  }

};

// Render articles to the DOM

const renderArticles = (articles) => {

  const articlesContainer = document.getElementById('articles');

  articles.forEach((article) => {

    const articleElement = document.createElement('div');

    articleElement.className = 'article';

    articleElement.innerHTML = `

      <h2>${article.title}</h2>

      <p>${article.body}</p>

    `;

    articlesContainer.appendChild(articleElement);

  });

};

// Debounce function to limit frequent calls

const debounce = (func, delay) => {

  let timeout;

  return (...args) => {

    clearTimeout(timeout);

    timeout = setTimeout(() => func(...args), delay);

  };

};

// Search articles

const searchInput = document.getElementById('search');

searchInput.addEventListener(

  'input',

  debounce(async (event) => {

    const query = event.target.value.trim();

    const articlesContainer = document.getElementById('articles');

    articlesContainer.innerHTML = ''; // Clear existing articles

    if (query) {

      try {

        const response = await fetch(`https://jsonplaceholder.typicode.com/posts?q=${query}`);

        if (!response.ok) throw new Error('Failed to fetch search results');

        const articles = await response.json();

        renderArticles(articles);

      } catch (error) {

        console.error('Error searching articles:', error);

      }

    } else {

      // Reload initial articles if search is cleared

      currentPage = 1;

      articlesContainer.innerHTML = ''; // Clear articles

      loadArticles(currentPage);

    }

  }, 300)

);

// Infinite scroll for lazy loading

window.addEventListener('scroll', () => {

  const { scrollTop, scrollHeight, clientHeight } = document.documentElement;

  if (!isLoading && scrollTop + clientHeight >= scrollHeight - 5) {

    currentPage++;

    loadArticles(currentPage);

  }

});

// Initial load

loadArticles(currentPage);

**Reference Counting**:

**Example:**

let a = { name: "Alice" }; // Reference count = 1

let b = a; // Reference count = 2

a = null; // Reference count = 1

b = null; // Reference count = 0 (garbage collected)

**Closures**: Variables in closures can persist in memory longer than necessary.

function createClosure() {

let data = "Leak";

return () => console.log(data);

}

let closure = createClosure();

**Excessive Object Retention**

Holding large objects in memory when they are no longer needed.

**Best Practices for Memory Management**

**Avoid Global Variables**: Use local variables wherever possible.

// BAD

var globalVar = "I'm global!";

// GOOD

function example() {

let localVar = "I'm local!";

}

**Nullify References**: Explicitly set variables to null when they are no longer needed.

let obj = { name: "Bob" };

obj = null; // Marks the object for garbage collection

**Remove Event Listeners**: Clean up event listeners to avoid memory leaks.

button.addEventListener('click', handleClick);

button.removeEventListener('click', handleClick);

**Use WeakMap and WeakSet**:

* + These are designed to handle objects that may not need to persist.
  + They allow garbage collection of their keys if there are no other references.

let weakMap = new WeakMap();

let obj = {};

weakMap.set(obj, "Value");

obj = null; // Garbage collected

**Examples of Memory Management**

**Example 1: Avoiding Closures Retaining Unnecessary Data**

function createLargeObject() {

let largeObject = new Array(1000000).fill("data");

return () => console.log("Closure active");

}

let closure = createLargeObject();

// `largeObject` will not be garbage collected until `closure` is cleared

closure = null; // Allows garbage collection

**Example 2: Managing Event Listeners**

let button = document.getElementById("myButton");

function handleClick() {

console.log("Button clicked");

}

button.addEventListener("click", handleClick);

// Later

button.removeEventListener("click", handleClick);

**Batch DOM Manipulations**

Minimize the number of times you modify the DOM by batching changes.

const fragment = document.createDocumentFragment();

for (let i = 0; i < 100; i++) {

const div = document.createElement('div');

div.textContent = `Item ${i}`;

fragment.appendChild(div);

}

document.body.appendChild(fragment); // Single reflow

**Use class Instead of Inline Styles**

Apply multiple style changes at once using CSS classes instead of updating inline styles individually.

// Better: Single DOM update

element.classList.add('new-style');

**Minimize Layout Thrashing**

Avoid repeatedly querying layout properties like offsetWidth or getBoundingClientRect() between DOM updates.

// Bad: Causes multiple reflows

for (let i = 0; i < elements.length; i++) {

elements[i].style.width = `${elements[i].offsetWidth}px`;

}

// Good: Cache layout properties

const width = container.offsetWidth;

elements.forEach(el => el.style.width = `${width}px`);

**Use CSS Transforms and Animations**

Prefer transform and opacity for animations, as they do not trigger reflows or repaints.

/\* Better animation \*/

.animate {

transform: translateX(100px);

opacity: 0.5;

}

**Defer or Throttle Resizing**

Use resize event listeners sparingly and throttle them for better performance.

let timeout;

window.addEventListener('resize', () => {

clearTimeout(timeout);

timeout = setTimeout(() => {

console.log('Resized');

}, 200);

});

**Use requestAnimationFrame for Animations**

Synchronize DOM changes with the browser's repaint cycle for smoother animations.

function update() {

element.style.transform = `translateX(${distance}px)`;

requestAnimationFrame(update);

}

requestAnimationFrame(update);

**Example: Poor vs. Optimized Code**

**Unoptimized Code**

const list = document.getElementById('list');

for (let i = 0; i < 100; i++) {

const item = document.createElement('li');

item.textContent = `Item ${i}`;

list.appendChild(item); // Triggers reflow on each append

}

**Optimized Code**

const list = document.getElementById('list');

const fragment = document.createDocumentFragment();

for (let i = 0; i < 100; i++) {

const item = document.createElement('li');

item.textContent = `Item ${i}`;

fragment.appendChild(item); // No immediate reflow

}

list.appendChild(fragment); // Single reflow

**Create a Web Worker**

You need a separate JavaScript file that will run as the worker thread. For example:

**worker.js**:

// worker.js

onmessage = function(event) {

const data = event.data; // Receive data from the main thread

let result = 0;

for (let i = 0; i < data; i++) {

result += i; // Perform a computation

}

postMessage(result); // Send the result back to the main thread

};

**Initialize the Worker in the Main Thread**

Create a new Worker instance and specify the worker script file.

**main.js**:

const worker = new Worker('worker.js'); // Initialize the worker

// Send data to the worker

worker.postMessage(1000000); // Sending 1 million as data to compute

// Listen for messages from the worker

worker.onmessage = function(event) {

console.log('Result from worker:', event.data); // Output the result

};

// Handle errors

worker.onerror = function(event) {

console.error('Error in worker:', event.message);

};

**Clean Up the Worker**

When the worker is no longer needed, terminate it to free up resources:

worker.terminate(); // Stop the worker

**Complete Example**

**HTML:web-worker.html**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Web Workers Example</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      text-align: center;

      padding: 20px;

    }

    canvas {

      border: 1px solid #ccc;

      margin-top: 20px;

    }

  </style>

</head>

<body>

  <h1>Web Workers Image Processing</h1>

  <p>Upload an image to see the grayscale effect using a Web Worker.</p>

  <input type="file" id="fileInput" accept="image/\*">

  <canvas id="canvas"></canvas>

  <script src="main.js"></script>

</body>

</html>

**JavaScript: main.js**

const fileInput = document.getElementById('fileInput');

const canvas = document.getElementById('canvas');

const ctx = canvas.getContext('2d');

// When the user selects an image

fileInput.addEventListener('change', (event) => {

  const file = event.target.files[0];

  if (file) {

    const reader = new FileReader();

    // Load the image

    reader.onload = (e) => {

      const img = new Image();

      img.src = e.target.result;

      img.onload = () => {

        canvas.width = img.width;

        canvas.height = img.height;

        ctx.drawImage(img, 0, 0);

        // Get image data

        const imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);

        // Use Web Worker to process the image

        const worker = new Worker('worker.js');

        worker.postMessage(imageData);

        // Handle the processed image from the worker

        worker.onmessage = (e) => {

          ctx.putImageData(e.data, 0, 0); // Display the processed image

        };

        // Handle errors

        worker.onerror = (e) => {

          console.error('Worker error:', e.message);

        };

      };

    };

    reader.readAsDataURL(file); // Read the file as a data URL

  } else {

    alert('Please select an image file.');

  }

});

**JavaScript: worker.js**

const fileInput = document.getElementById('fileInput');

const canvas = document.getElementById('canvas');

const ctx = canvas.getContext('2d');

// When the user selects an image

fileInput.addEventListener('change', (event) => {

  const file = event.target.files[0];

  if (file) {

    const reader = new FileReader();

    // Load the image

    reader.onload = (e) => {

      const img = new Image();

      img.src = e.target.result;

      img.onload = () => {

        canvas.width = img.width;

        canvas.height = img.height;

        ctx.drawImage(img, 0, 0);

        // Get image data

        const imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);

        // Use Web Worker to process the image

        const worker = new Worker('worker.js');

        worker.postMessage(imageData);

        // Handle the processed image from the worker

        worker.onmessage = (e) => {

          ctx.putImageData(e.data, 0, 0); // Display the processed image

        };

        // Handle errors

        worker.onerror = (e) => {

          console.error('Worker error:', e.message);

        };

      };

    };

    reader.readAsDataURL(file); // Read the file as a data URL

  } else {

    alert('Please select an image file.');

  }

});

**Example: Interactive Graphics on Canvas**

Below is an example of an interactive canvas where the user can draw using the mouse, using specific color, can erase/clear the drawing and can save the drawing:

**HTML**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Interactive Drawing Canvas</title>

  <style>

    body {

      display: flex;

      flex-direction: column;

      align-items: center;

      justify-content: center;

      font-family: Arial, sans-serif;

      margin: 20px;

    }

    canvas {

      border: 2px solid #000;

      background-color: #fff;

      cursor: crosshair;

    }

    .toolbar {

      margin-bottom: 10px;

    }

    button {

      margin: 5px;

      padding: 5px 10px;

      font-size: 14px;

      cursor: pointer;

    }

  </style>

</head>

<body>

  <h1>🎨 Interactive Drawing Canvas</h1>

  <div class="toolbar">

    <label>Color: <input type="color" id="colorPicker"></label>

    <label>Line Width: <input type="range" id="lineWidth" min="1" max="10" value="2"></label>

    <button id="eraseBtn">Eraser</button>

    <button id="clearBtn">Clear</button>

    <button id="saveBtn">Save</button>

  </div>

  <canvas id="interactiveCanvas" width="800" height="400"></canvas>

  <script src="in\_script.js"></script>

</body>

</html>

**JavaScript**

const canvas = document.getElementById('interactiveCanvas');

const ctx = canvas.getContext('2d');

// Toolbar elements

const colorPicker = document.getElementById('colorPicker');

const lineWidthInput = document.getElementById('lineWidth');

const eraseBtn = document.getElementById('eraseBtn');

const clearBtn = document.getElementById('clearBtn');

const saveBtn = document.getElementById('saveBtn');

// Initial state

let isDrawing = false;

let lastX = 0;

let lastY = 0;

let currentColor = '#000000';

let currentLineWidth = 2;

let isErasing = false;

// Function to start drawing

function startDrawing(e) {

  isDrawing = true;

  [lastX, lastY] = [e.offsetX, e.offsetY];

}

// Function to draw on the canvas

function draw(e) {

  if (!isDrawing) return;

  ctx.beginPath();

  ctx.moveTo(lastX, lastY);

  ctx.lineTo(e.offsetX, e.offsetY);

  ctx.strokeStyle = isErasing ? '#FFFFFF' : currentColor;

  ctx.lineWidth = currentLineWidth;

  ctx.lineCap = 'round';

  ctx.stroke();

  [lastX, lastY] = [e.offsetX, e.offsetY];

}

// Function to stop drawing

function stopDrawing() {

  isDrawing = false;

  ctx.beginPath();

}

// Event listeners for mouse events

canvas.addEventListener('mousedown', startDrawing);

canvas.addEventListener('mousemove', draw);

canvas.addEventListener('mouseup', stopDrawing);

canvas.addEventListener('mouseout', stopDrawing);

// Touch support (for mobile)

canvas.addEventListener('touchstart', (e) => {

  const touch = e.touches[0];

  const rect = canvas.getBoundingClientRect();

  lastX = touch.clientX - rect.left;

  lastY = touch.clientY - rect.top;

  isDrawing = true;

});

canvas.addEventListener('touchmove', (e) => {

  if (!isDrawing) return;

  e.preventDefault();

  const touch = e.touches[0];

  const rect = canvas.getBoundingClientRect();

  draw({ offsetX: touch.clientX - rect.left, offsetY: touch.clientY - rect.top });

});

canvas.addEventListener('touchend', stopDrawing);

// Change color

colorPicker.addEventListener('input', (e) => {

  currentColor = e.target.value;

  isErasing = false;

});

// Change line width

lineWidthInput.addEventListener('input', (e) => {

  currentLineWidth = e.target.value;

});

// Eraser mode

eraseBtn.addEventListener('click', () => {

  isErasing = true;

});

// Clear canvas

clearBtn.addEventListener('click', () => {

  ctx.clearRect(0, 0, canvas.width, canvas.height);

});

// Save drawing

saveBtn.addEventListener('click', () => {

  const dataURL = canvas.toDataURL('image/png');

  const link = document.createElement('a');

  link.href = dataURL;

  link.download = 'drawing.png';

  link.click();

});

**Example: Fetching JSON Data (already discussed in few examples).**

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.error('Error:', error));

**Example: WebSockets**

const socket = new WebSocket('wss://example.com/socket');

socket.onopen = () => console.log('Connected to WebSocket');

socket.onmessage = (event) => console.log('Received:', event.data);

socket.onclose = () => console.log('Disconnected');

**Example: Storing & Retrieving Data**

localStorage.setItem('username', 'JohnDoe');

console.log(localStorage.getItem('username')); // Output: JohnDoe

**Example:**

sessionStorage.setItem('sessionID', '123456');

console.log(sessionStorage.getItem('sessionID'));

**IndexedDB API**

**Example:**

let db;

const request = indexedDB.open("MyDatabase", 1);

request.onsuccess = (event) => {

db = event.target.result;

console.log("Database opened successfully");

};

**Example: Drawing a Red Rectangle**

const canvas = document.getElementById('myCanvas');

const ctx = canvas.getContext('2d');

ctx.fillStyle = 'red';

ctx.fillRect(50, 50, 100, 100);

**Example: Accessing Camera & Microphone**

navigator.mediaDevices.getUserMedia({ video: true, audio: true })

.then(stream => document.querySelector('video').srcObject = stream)

.catch(error => console.error('Error:', error));

**Example: Creating an Audio Context**

const audioContext = new AudioContext();

console.log("Web Audio API Initialized");

**Example: Fetching User Location**

navigator.geolocation.getCurrentPosition(position => {

console.log(`Latitude: ${position.coords.latitude}, Longitude: ${position.coords.longitude}`);

});

**Example: Checking Battery Level**

navigator.getBattery().then(battery => {

console.log(`Battery Level: ${battery.level \* 100}%`);

});

**Example: Vibrating for 500ms**

navigator.vibrate(500);

**Example: Running a Background Task**

const worker = new Worker('worker.js');

worker.postMessage("Start Task");

worker.onmessage = (event) => console.log("Message from Worker:", event.data);

**Example: Lazy Loading Images**

const observer = new IntersectionObserver(entries => {

entries.forEach(entry => {

if (entry.isIntersecting) {

const img = entry.target;

img.src = img.dataset.src;

observer.unobserve(img);

}

});

});

document.querySelectorAll('img.lazy').forEach(img => observer.observe(img));

**Example: Storing User Credentials**

navigator.credentials.create({

password: {

id: "user@example.com",

password: "securepassword"

}

});

**Example: Checking Camera Permission**

navigator.permissions.query({ name: 'camera' }).then(permission => {

console.log(permission.state);

});

**Payment Request API**

* Enables **in-browser** payments without redirecting.

**Example:**

const paymentRequest = new PaymentRequest([{

supportedMethods: 'basic-card',

}], { total: { label: 'Total', amount: { currency: 'USD', value: '10.00' } } });

paymentRequest.show();

**Push Notifications API**

* Allows sending notifications to users.

**Clipboard API**

* Enables copying text to the clipboard.

**Example: Copying Text**

navigator.clipboard.writeText("Hello World").then(() => {

console.log("Text copied to clipboard!");

});

**Example: Running WebAssembly in JavaScript**

Below is an example of loading a WebAssembly module in JavaScript:

**Step 1: Write C code (add.c)**

// add.c - Simple function to add two numbers

int add(int a, int b) {

return a + b;

}

**Step 2: Compile to WebAssembly**

Using Emscripten (a tool for compiling C/C++ to WASM), compile the code:

emcc add.c -o add.wasm -s EXPORTED\_FUNCTIONS='["\_add"]' -s MODULARIZE

**Step 3: Load WebAssembly in JavaScript**

fetch('add.wasm')

.then(response => response.arrayBuffer())

.then(bytes => WebAssembly.instantiate(bytes))

.then(({ instance }) => {

console.log(instance.exports.add(5, 3)); // Output: 8

});

**Key ESNext Features:**

**Optional Chaining (?.)**

Allows accessing deeply nested object properties without checking if each level exists.

**Before ESNext:**

let user = {};

console.log(user.profile && user.profile.name); // undefined (avoiding errors)

**With Optional Chaining:**

console.log(user?.profile?.name); // undefined (without error)

**Nullish Coalescing Operator (??)**

Provides a **default value** when a variable is null or undefined.

**Before ESNext:**

let username = null;

let displayName = username || "Guest"; // Problem: 0 or '' also default to "Guest"

**With ??:**

let displayName = username ?? "Guest"; // Correct behavior

**Top-Level await**

Allows await to be used in modules without wrapping in an async function.

**Before ESNext:**

async function fetchData() {

let response = await fetch("https://jsonplaceholder.typicode.com/posts/1");

let data = await response.json();

console.log(data);

}

fetchData();

**With Top-Level await:**

let response = await fetch("https://jsonplaceholder.typicode.com/posts/1");

let data = await response.json();

console.log(data);

**Records & Tuples (Proposal)**

Introduces **immutable** data structures in JavaScript.

**Example:**

const user = #{"name": "Alice", "age": 25}; // Record

const numbers = #[1, 2, 3]; // Tuple

**Pattern Matching (Proposal)**

A feature similar to switch-case, but more powerful.

**Example:**

const value = { type: "circle", radius: 10 };

match (value) {

{ type: "circle", radius } => console.log(`Circle with radius ${radius}`),

{ type: "square", side } => console.log(`Square with side ${side}`),

\_ => console.log("Unknown shape")

}