66 JavaScript Coding Exercises

Source Code included!!!!

1. Change Background Color

Objective: Create a web page with a button that changes the background color of the page when clicked.

```
<!DOCTYPE html>
<html>
<head>
 <title>Change Background Color</title>
</head>
<body>
 <button id="changeColorBtn">Change Background Color</button>
 <script>
 document.getElementById('changeColorBtn').addEventListener('click',
function() {
 document.body.style.backgroundColor = 'lightblue';
 });
 </script>
</body>
</html>
```

2. Toggle Image Display

Objective: Implement a feature to toggle the visibility of an image when a button is clicked.

```
<!DOCTYPE html>
<html>
<head>
 <title>Toggle Image Display</title>
</head>
<body>
 <button id="toggleImageBtn">Toggle Image</button>
 <img id="toggleImage" src="example.jpg" alt="Image"
style="display:none;">
 <script>
 document.getElementById('toggleImageBtn').addEventListener('click',
function() {
 var img = document.getElementById('toggleImage');
 img.style.display = img.style.display === 'none' ? 'block' : 'none';
 });
 </script>
</body>
</html>
```

3. Form Validation

Objective: Create a form that validates an email input field and displays a message if the input is not a valid email address.

```
<!DOCTYPE html>
<html>
<head>
 <title>Form Validation</title>
</head>
<body>
 <form id="emailForm">
 Email: <input type="email" id="emailInput">
 <button type="submit">Submit</button>
 <p id="message"></p>
 </form>
 <script>
 document.getElementById('emailForm').addEventListener('submit',
function(event) {
 event.preventDefault();
 var email = document.getElementById('emailInput').value;
 var message = '';
 if (!email.includes('@')) {
 message = 'Please enter a valid email address.';
 } else {
 message = 'Email is valid!';
 }
 document.getElementById('message').innerText = message;
 });
 </script>
</body>
```

```
</html>
```

4. Fetch and Display Data

Objective: Fetch data from a public API and display the results in a list.

```
<!DOCTYPE html>
<html>
<head>
 <title>Fetch and Display Data</title>
</head>
<body>
 <ul id="dataList"></ul>
 <script>
 fetch('https://jsonplaceholder.typicode.com/posts')
 .then(response => response.json())
 .then(data => {
 const list = document.getElementById('dataList');
 data.forEach(item => {
 const li = document.createElement('li');
 li.textContent = item.title;
 list.appendChild(li);
 });
 });
 </script>
</body>
</html>
```

5. Create a To-Do List

Objective: Develop a simple to-do list where users can add items to a list and remove them once completed.

```
<!DOCTYPE html>
<html>
<head>
 <title>To-Do List</title>
 <style>
 .todo-item { cursor: pointer; }
 </style>
</head>
<body>
 <input type="text" id="todoInput">
 <button id="addTodoBtn">Add To-Do</button>
 <ul id="todoList"></ul>
 <script>
 document.getElementById('addTodoBtn').addEventListener('click',
function() {
 var input = document.getElementById('todoInput');
 var itemText = input.value;
 input.value = ''; // Clear input field
 var li = document.createElement('li');
 li.textContent = itemText;
 li.classList.add('todo-item');
 li.addEventListener('click', function() {
 this.parentNode.removeChild(this);
 });
 document.getElementById('todoList').appendChild(li);
 });
 </script>
</body>
</html>
```

6. Simple Calculator

Objective: Implement a simple calculator that can perform basic arithmetic operations like addition, subtraction, multiplication, and division.

```
<!DOCTYPE html>
<html>
<head>
 <title>Simple Calculator</title>
</head>
<body>
 <input type="text" id="number1">
 <select id="operation">
 <option value="add">+</option>
 <option value="subtract">-</option>
 <option value="multiply">*</option>
 <option value="divide">/</option>
 </select>
 <input type="text" id="number2">
 <button id="calculateBtn">Calculate</button>
 <p id="result"></p>
 <script>
 document.getElementById('calculateBtn').addEventListener('click',
function() {
 var num1 = parseFloat(document.getElementById('number1').value);
 var num2 = parseFloat(document.getElementById('number2').value);
 var operation = document.getElementById('operation').value;
 var result;
 switch(operation) {
 case 'add':
 result = num1 + num2;
 break;
 case 'subtract':
 result = num1 - num2;
 break;
 case 'multiply':
 result = num1 * num2;
 break;
 case 'divide':
 result = num1 / num2;
 break;
 }
 document.getElementById('result').innerText = 'Result: ' + result;
 });
 </script>
</body>
</html>
```

7. Countdown Timer

Objective: Create a countdown timer that counts down from a specified time and displays a message when the countdown reaches zero.

```
<!DOCTYPE html>
<html>
<head>
 <title>Countdown Timer</title>
</head>
<body>
 <div id="timer"></div>
 <script>
 var countdownDuration = 10; // 10 seconds for example
 var timerDisplay = document.getElementById('timer');
 var countdown = setInterval(function() {
 timerDisplay.innerText = countdownDuration + ' seconds remaining';
 countdownDuration--;
 if (countdownDuration < 0) {
 clearInterval(countdown);
 timerDisplay.innerText = 'Time is up!';
 }
 }, 1000);
 </script>
</body>
</html>
```

8. Interactive Quiz

Objective: Design an interactive quiz with multiple-choice questions. After submitting, show the correct answers and the user's score.

```
<!DOCTYPE html>
<html>
<head>
 <title>Interactive Quiz</title>
</head>
<body>
```

```
 <div id="quiz"></div>
 <button id="submitQuizBtn">Submit Answers</button>
 <div id="result"></div>
 <script>
 const questions = [
 { question: 'What is 2 + 2?', choices: ['2', '3', '4', '5'], correctAnswer: '4' },
 { question: 'What is the capital of France?', choices: ['Paris', 'London',
'Berlin', 'Madrid'], correctAnswer: 'Paris' }
 ];
 function displayQuestions() {
 const quizContainer = document.getElementById('quiz');
 questions.forEach((q, index) => {
 const questionDiv = document.createElement('div');
 questionDiv.innerHTML = `<p>${q.question}</p>`;
 q.choices.forEach(choice => {
 questionDiv.innerHTML += `<input type="radio" name="question${index}"
value="${choice}">${choice}<br>`;
 });
 quizContainer.appendChild(questionDiv);
 });
 }
 document.getElementById('submitQuizBtn').addEventListener('click',
function() {
 let score = 0;
 questions.forEach((q, index) => {
 const selectedAnswer =
document.querySelector(`input[name="question${index}"]:checked`)?.value
;
 if (selectedAnswer === q.correctAnswer) {
 score++;
 }
 });
 document.getElementById('result').innerHTML = `You scored ${score} out
of ${questions.length}`;
 });
 displayQuestions();
 </script>
</body>
```

```
</html>
```

9. Dynamic Content Loading

Objective: Dynamically load content into a page section without reloading the page when a button is clicked.

```
<!DOCTYPE html>
<html>
<head>
 <title>Dynamic Content Loading</title>
</head>
<body>
 <button id="loadContentBtn">Load Content</button>
 <div id="content"></div>
 <script>
 document.getElementById('loadContentBtn').addEventListener('click',
function() {
 document.getElementById('content').innerHTML = '<p>New content
loaded!</p>';
 });
 </script>
</body>
</html>
```

10. Image Slider

Objective: Create an image slider that cycles through a series of images when the next and previous buttons are clicked.

```
<!DOCTYPE html>
<html>
<head>
 <title>Image Slider</title>
 <style>
 .slider { width: 500px; overflow: hidden; }
 .slider img { width: 500px; display: none; }
 </style>
```

```
</head>
<body>
 <div class="slider">
 <img src="image1.jpg" alt="Image 1" style="display:block;">
 <img src="image2.jpg" alt="Image 2">
 <img src="image3.jpg" alt="Image 3">
 </div>
 <button id="prevBtn">Previous</button>
 <button id="nextBtn">Next</button>
 <script>
 var images = document.querySelectorAll('.slider img');
 var currentImageIndex = 0;
 function showImage(index) {
 images.forEach((img, i) => {
 img.style.display = i === index ? 'block' : 'none';
 });
 }
 document.getElementById('nextBtn').addEventListener('click', function() {
 currentImageIndex = (currentImageIndex + 1) % images.length;
 showImage(currentImageIndex);
 });
 document.getElementById('prevBtn').addEventListener('click', function() {
 currentImageIndex = (currentImageIndex - 1 + images.length) %
images.length;
 showImage(currentImageIndex);
 });
 showImage(currentImageIndex);
 </script>
</body>
</html>
```

11. Modal Popup

Objective: Create a modal popup that appears when a button is clicked and can be closed with a close button.

```
<!DOCTYPE html>
<html>
```

```
<head>
 <title>Modal Popup</title>
 <style>
 .modal { display: none; position: fixed; z-index: 1; left: 0; top: 0; width: 100%; height: 100%; overflow: auto; background-color: rgba(0,0,0,0.4); }
 .modal-content { background-color: #fefefe; margin: 15% auto; padding: 20px; border: 1px solid #888; width: 80%; }
 </style>
</head>
<body>
 <button id="openModalBtn">Open Modal</button>
 <div id="myModal" class="modal">
 <div class="modal-content">
 <span id="closeModalBtn">&times;</span>
 <p>Some text in the Modal..</p>
 </div>
 </div>
 <script>
 var modal = document.getElementById('myModal');
 var btn = document.getElementById('openModalBtn');
 var span = document.getElementById('closeModalBtn');
 btn.onclick = function() {
 modal.style.display = "block";
 }
 span.onclick = function() {
 modal.style.display = "none";
 }
 window.onclick = function(event) {
 if (event.target == modal) {
 modal.style.display = "none";
 }
 }
 </script>
</body>
</html>
```

12. Dynamic Select Options

Objective: Dynamically populate a select dropdown with options from an array.

```
<!DOCTYPE html>
<html>
<head>
 <title>Dynamic Select Options</title>
</head>
<body>
 <select id="dynamicSelect"></select>
 <script>
 const options = ['Option 1', 'Option 2', 'Option 3'];
 const select = document.getElementById('dynamicSelect');
 options.forEach(option => {
 const opt = document.createElement('option');
 opt.value = option;
 opt.innerHTML = option;
 select.appendChild(opt);
 });
 </script>
</body>
</html>
```

13. Accordion

Objective: Implement an accordion where clicking on a section header expands or collapses the section content.

```
<!DOCTYPE html>
<html>
<head>
 <title>Accordion</title>
 <style>
 .accordion { background-color: #eee; color: #444; cursor: pointer; padding: 18px; width: 100%; border: none; text-align: left; outline: none; font-size: 15px; transition: 0.4s; }
 .active, .accordion:hover { background-color: #ccc; }
 .panel { padding: 0 18px; display: none; background-color: white; overflow: hidden; }
 </style>
</head>
<body>
 <button class="accordion">Section 1</button>
 <div class="panel">
 <p>Section 1 content...</p>
 </div>
 <button class="accordion">Section 2</button>
 <div class="panel">
 <p>Section 2 content...</p>
 </div>
 <script>
 var acc = document.getElementsByClassName("accordion");
 for (var i = 0; i < acc.length; i++) {
 acc[i].addEventListener("click", function() {
 this.classList.toggle("active");
 var panel = this.nextElementSibling;
 if (panel.style.display === "block") {
 panel.style.display = "none";
 } else {
 panel.style.display = "block";
 }
 });
 }
 </script>
</body>
</html>
```

14. Filter List

Objective: Implement a search feature that filters through a list of items and shows only those matching the query.

```
<!DOCTYPE html>
<html>
```

```
<head>
 <title>Filter List</title>
</head>
<body>
 <input type="text" id="filterInput" placeholder="Search for names..">
 <ul id="namesList">
 <li>John</li>
 <li>Jane</li>
 <li>Mike</li>
 <li>Mary</li>
 </ul>
 <script>
 document.getElementById('filterInput').addEventListener('keyup', function()
{
 var input = document.getElementById('filterInput').value.toUpperCase();
 var ul = document.getElementById("namesList");
 var li = ul.getElementsByTagName('li');
 for (var i = 0; i < li.length; i++) {
 var txtValue = li[i].textContent || li[i].innerText;
 if (txtValue.toUpperCase().indexOf(input) > -1) {
 li[i].style.display = "";
 } else {
 li[i].style.display = "none";
 }
 }
 });
 </script>
</body>
</html>
```

15. Sort Table

Objective: Create a table that can be sorted by column when the header is clicked.

```
<!DOCTYPE html>
<html>
<head>
```

```
 <title>Sort Table</title>
</head>
<body>
 <table id="sortTable">
 <thead>
 <tr>
 <th onclick="sortTable(0)">Name</th>
 <th onclick="sortTable(1)">Age</th>
 </tr>
 </thead>
 <tbody>
 <tr>
 <td>John</td>
 <td>25</td>
 </tr>
 <tr>
 <td>Jane</td>
 <td>20</td>
 </tr>
 <tr>
 <td>Mike</td>
 <td>30</td>
 </tr>
 </tbody>
 </table>
 <script>
 function sortTable(n) {
 var table, rows, switching, i, x, y, shouldSwitch, dir, switchcount = 0;
 table = document.getElementById("sortTable");
 switching = true;
 // Set the sorting direction to ascending:
 dir = "asc";
 while (switching) {
 switching = false;
 rows = table.rows;
 for (i = 1; i < (rows.length - 1); i++) {
 shouldSwitch = false;
 x = rows[i].getElementsByTagName("TD")[n];
 y = rows[i + 1].getElementsByTagName("TD")[n];
 if (dir == "asc") {
 if (x.innerHTML.toLowerCase() > y.innerHTML.toLowerCase()) {
 shouldSwitch= true;
 break;
 }
 } else if (dir == "desc") {
 if (x.innerHTML.toLowerCase() < y.innerHTML.toLowerCase()) {
 shouldSwitch = true;
 break;
 }
 }
 }
 if (shouldSwitch) {
 rows[i].parentNode.insertBefore(rows[i + 1], rows[i]);
 switching = true;
 switchcount ++;
 } else {
 if (switchcount == 0 && dir == "asc") {
 dir = "desc";
 switching = true;
 }
 }
 }
 }
 </script>
</body>
</html>
```

16. Sticky Navigation Bar

Objective: Create a navigation bar that becomes fixed to the top of the viewport when you scroll past it.

```
<!DOCTYPE html>
<html>
<head>
 <title>Sticky Navigation Bar</title>
 <style>
 .navbar { overflow: hidden; background-color: #333; }
 .navbar a { float: left; display: block; color: white; text-align: center;
padding: 14px 20px; text-decoration: none; }
 .sticky { position: fixed; top: 0; width: 100%; }
 </style>
</head>
<body>
 <div class="navbar" id="navbar">
 <a href="#home">Home</a>
 <a href="#news">News</a>
 <a href="#contact">Contact</a>
 </div>
 <script>
 window.onscroll = function() {myFunction()};
 var navbar = document.getElementById("navbar");
 var sticky = navbar.offsetTop;
 function myFunction() {
 if (window.pageYOffset >= sticky) {
 navbar.classList.add("sticky")
 } else {
 navbar.classList.remove("sticky");
 }
 }
 </script>
</body>
</html>
```

17. Slideshow

Objective: Develop a slideshow that automatically transitions between images every few seconds.

```
<!DOCTYPE html>
<html>
<head>
 <title>Slideshow</title>
 <style>
 .slideshow-container { max-width: 1000px; position: relative; margin: auto;
}
 .mySlides { display: none; }
 .fade { animation-name: fade; animation-duration: 1.5s; }
 @keyframes fade { from {opacity: .4} to {opacity: 1} }
 </style>
</head>
<body>
 <div class="slideshow-container">
 <div class="mySlides fade">
 <img src="img1.jpg" style="width:100%">
 </div>
 <div class="mySlides fade">
 <img src="img2.jpg" style="width:100%">
 </div>
 <div class="mySlides fade">
 <img src="img3.jpg" style="width:100%">
 </div>
 </div>
 <script>
 var slideIndex = 0;
 showSlides();
 function showSlides() {
 var i;
 var slides = document.getElementsByClassName("mySlides");
 for (i = 0; i < slides.length; i++) {
 slides[i].style.display = "none";
 }
 slideIndex++;
 if (slideIndex > slides.length) {slideIndex = 1}
 slides[slideIndex-1].style.display = "block";
 setTimeout(showSlides, 2000); // Change image every 2 seconds
 }
 </script>
</body>
</html>
```

18. Expand/Collapse Content

Objective: Create a section of content that can be expanded or collapsed with a button click.

```
<!DOCTYPE html>
<html>
<head>
 <title>Expand/Collapse Content</title>
</head>
<body>
 <button onclick="toggleContent()">Show More/Less</button>
 <div id="expandableContent" style="display:none;">
 <p>Here is some more content you can see after you click the button!</p>
 </div>
 <script>
 function toggleContent() {
 var content = document.getElementById("expandableContent");
 if (content.style.display === "none") {
 content.style.display = "block";
 } else {
 content.style.display = "none";
 }
 }
 </script>
</body>
</html>
```

19. Digital Clock

Objective: Display a live digital clock on the web page that updates every second.

```
<!DOCTYPE html>
<html>
<head>
 <title>Digital Clock</title>
</head>
```

```
<body>
 <div id="digitalClock"></div>
 <script>
 function showTime() {
 var date = new Date();
 var h = date.getHours(); // 0 - 23
 var m = date.getMinutes(); // 0 - 59
 var s = date.getSeconds(); // 0 - 59
 var session = h >= 12 ? 'PM' : 'AM';
 h = h % 12;
 h = h ? h : 12; // the hour '0' should be '12'
 m = m < 10 ? '0' + m : m;
 s = s < 10 ? '0' + s : s;
 var time = h + ":" + m + ":" + s + " " + session;
 document.getElementById("digitalClock").innerText = time;
 document.getElementById("digitalClock").textContent = time;
 setTimeout(showTime, 1000);
 }
 showTime();
 </script>
</body>
</html>
```

20. Typing Speed Tester

Objective: Create a simple typing speed tester that measures how quickly the user can type a provided sample text.

```
<!DOCTYPE html>
<html>
<head>
 <title>Typing Speed Tester</title>
</head>
<body>
 <p id="sampleText">Type this text as fast as you can!</p>
 <textarea id="typingArea" rows="10" cols="50"></textarea>
 <button id="startBtn">Start</button>
 <p id="result"></p>
 <script>
 document.getElementById('startBtn').addEventListener('click', function() {
 var startTime = new Date().getTime();
 var sampleText = document.getElementById('sampleText').innerText;
 var typingArea = document.getElementById('typingArea');
 typingArea.value = '';
 typingArea.focus();
 typingArea.oninput = function() {
 if(typingArea.value === sampleText) {
 var endTime = new Date().getTime();
 var elapsedTime = (endTime - startTime) / 1000; // in seconds
 document.getElementById('result').innerText = 'Your time: ' + elapsedTime
+ ' seconds';
 typingArea.oninput = null; // stop the test
 }
 };
 });
 </script>
</body>
</html>
```

These exercises are designed to progressively build a learner's JavaScript, HTML, and CSS skills through practical, hands-on coding tasks. They can be used as part of a course curriculum, self-study, or coding practice to reinforce web development concepts.

21. Local Storage Todo List

Objective: Enhance the To-Do List application by adding functionality to save tasks to local storage, allowing persistence between page reloads.

```
<!DOCTYPE html>
<html>
<head>
    <title>Local Storage Todo List</title>
</head>
<body>
    <input type="text" id="taskInput">
    <button onclick="addTask()">Add Task</button>
    <ul id="tasks"></ul>
    <script>
        document.addEventListener('DOMContentLoaded', loadTasks);
        function addTask() {
            const input = document.getElementById('taskInput');
            const task = input.value.trim();
            if (!task) return;
            saveTask(task);
            input.value = '';
            renderTask(task);
        }
        function saveTask(task) {
            const tasks = JSON.parse(localStorage.getItem('tasks')) || [];
            tasks.push(task);
            localStorage.setItem('tasks', JSON.stringify(tasks));
        }
        function renderTask(task) {
            const list = document.getElementById('tasks');
            const li = document.createElement('li');
            li.textContent = task;
            list.appendChild(li);
        }
        function loadTasks() {
            const tasks = JSON.parse(localStorage.getItem('tasks')) || [];
            tasks.forEach(renderTask);
        }
    </script>
</body>
</html>
```

22. Async Data Loading

Objective: Fetch data from an API asynchronously using async/await syntax and display it on the page.

```
<!DOCTYPE html>
<html>
<head>
    <title>Async Data Loading</title>
```

```
</head>
<body>
    <div id="data"></div>
    <script>
        async function loadData() {
            const response = await
fetch('https://jsonplaceholder.typicode.com/posts');
            const data = await response.json();
            const container = document.getElementById('data');
            container.innerHTML = data.map(post =>
`<p>${post.title}</p>`).join('');
        }
        loadData();
    </script>
</body>
</html>
```

23. Canvas Drawing

Objective: Create a simple drawing application using the HTML5 <canvas> element.

```
<!DOCTYPE html>
<html>
<head>
    <title>Canvas Drawing</title>
</head>
<body>
    <canvas id="drawingCanvas" width="400" height="400"
style="border:1px solid #000;"></canvas>
    <script>
        const canvas = document.getElementById('drawingCanvas');
        const ctx = canvas.getContext('2d');
        let painting = false;
        function startPosition(e) {
            painting = true;
            draw(e);
        }
        function endPosition() {
```

```
      painting = false;
      ctx.beginPath();
    }
    function draw(e) {
      if (!painting) return;
      ctx.lineWidth = 5;
      ctx.lineCap = 'round';
      ctx.lineTo(e.clientX - canvas.offsetLeft, e.clientY -
canvas.offsetTop);
      ctx.stroke();
      ctx.beginPath();
      ctx.moveTo(e.clientX - canvas.offsetLeft, e.clientY -
canvas.offsetTop);
    }
    canvas.addEventListener('mousedown', startPosition);
    canvas.addEventListener('mouseup', endPosition);
    canvas.addEventListener('mousemove', draw);
  </script>
</body>
</html>
```

24. Custom Video Player

Objective: Create a custom video player controls using HTML5 video APIs.

```
<!DOCTYPE html>
<html>
<head>
  <title>Custom Video Player</title>
</head>
<body>
  <video id="videoPlayer" width="320" height="240" controls>
    <source src="movie.mp4" type="video/mp4">
    Your browser does not support the video tag.
  </video>
  <button id="playPause">Play/Pause</button>
  <script>
    const video = document.getElementById('videoPlayer');
    const playPauseBtn = document.getElementById('playPause');
```

```
        playPauseBtn.addEventListener('click', function() {
            if (video.paused) {
                video.play();
                playPauseBtn.textContent = 'Pause';
            } else {
                video.pause();
                playPauseBtn.textContent = 'Play';
            }
        });
    </script>
</body>
</html>
```

25. Drag and Drop

Objective: Implement a basic drag and drop interface allowing users to drag an item from one list to another.

```
<!DOCTYPE html>
<html>
<head>
    <title>Drag and Drop</title>
    <style>
        .droppable { width: 200px; height: 100px; margin: 10px; background:
#f9f9f9; border: 2px dashed #cccccc; }
        .draggable { width: 100px; height: 50px; background: lightblue;
text-align: center; line-height: 50px; }
    </style>
</head>
<body>
    <div class="droppable" ondrop="drop(event)"
ondragover="allowDrop(event)"></div>
    <div class="draggable" draggable="true"
ondragstart="drag(event)">Drag me</div>
    <script>
        function allowDrop(ev) {
            ev.preventDefault();
        }
        function drag(ev) {
            ev.dataTransfer.setData("text", ev.target.id);
        }
        function drop(ev) {
            ev.preventDefault();
            var data = ev.dataTransfer.getData("text");
            ev.target.appendChild(document.getElementById(data));
        }
    </script>
</body>
</html>
```

26. Simple Pagination

Objective: Implement simple pagination for a list of items, displaying a fixed number of items per page.

```
<!DOCTYPE html>
<html>
<head>
    <title>Simple Pagination</title>
</head>
<body>
    <ul id="list">
        <!-- Items will be dynamically added here -->
    </ul>
    <button onclick="prevPage()">Prev</button>
    <button onclick="nextPage()">Next</button>
    <script>
        const items = Array.from({ length: 100 }).map((_, i) => `Item ${i + 1}`);
        let currentPage = 1;
        const itemsPerPage = 10;
        function renderList(page) {
            const start = (page - 1) * itemsPerPage;
            const end = start + itemsPerPage;
            const slicedItems = items.slice(start, end);
            document.getElementById('list').innerHTML = slicedItems.map(item
=> `<li>${item}</li>`).join('');
        }
        function prevPage() {
```

```
        if (currentPage > 1) {
            currentPage--;
            renderList(currentPage);
        }
    }
    function nextPage() {
        if (currentPage * itemsPerPage < items.length) {
            currentPage++;
            renderList(currentPage);
        }
    }
    renderList(currentPage);
  </script>
</body>
</html>
```

27. Infinite Scrolling

Objective: Implement an infinite scrolling feature that loads more content as the user scrolls down the page.

```
<!DOCTYPE html>
<html>
<head>
  <title>Infinite Scrolling</title>
</head>
<body>
  <div id="content"></div>
  <script>
    let count = 0;
    function loadMoreContent() {
        for (let i = 0; i < 10; i++) {
            const div = document.createElement('div');
            div.textContent = `Item ${++count}`;
            document.getElementById('content').appendChild(div);
        }
    }
    window.addEventListener('scroll', () => {
      if (window.scrollY + window.innerHeight >=
document.documentElement.scrollHeight) {
        loadMoreContent();
      }
    });
    loadMoreContent(); // initial load
  </script>
</body>
</html>
```

28. Web Storage API

Objective: Use the Web Storage API to store user preferences that persist across sessions.

```
<!DOCTYPE html>
<html>
<head>
  <title>Web Storage API</title>
</head>
<body>
  <label for="color">Choose a background color:</label>
  <select id="color">
    <option value="white">White</option>
    <option value="lightblue">Light Blue</option>
    <option value="lightgreen">Light Green</option>
  </select>
  <script>
    document.getElementById('color').addEventListener('change',
function() {
      const selectedColor = this.value;
      document.body.style.backgroundColor = selectedColor;
      localStorage.setItem('bgColor', selectedColor);
    });
    document.addEventListener('DOMContentLoaded', () => {
      const savedColor = localStorage.getItem('bgColor');
      if (savedColor) {
        document.body.style.backgroundColor = savedColor;
        document.getElementById('color').value = savedColor;
```

```
            }
        });
    </script>
</body>
</html>
```

29. Fetch and Display JSON

Objective: Fetch JSON data from an external file and display it in a formatted manner on the webpage.

```
<!DOCTYPE html>
<html>
<head>
    <title>Fetch and Display JSON</title>
</head>
<body>
    <div id="userData"></div>
    <script>
        async function fetchUserData() {
            const response = await fetch('users.json');
            const users = await response.json();
            const formattedUsers = users.map(user => `<p>Name:
${user.name}, Age: ${user.age}</p>`).join('');
            document.getElementById('userData').innerHTML =
formattedUsers;
        }
        fetchUserData();
    </script>
</body>
</html>
```

30. Real-time Search Filter

Objective: Implement a real-time search filter that dynamically filters through a list of items as the user types in a search box.

```
<!DOCTYPE html>
<html>
<head>
```

```
  <title>Real-time Search Filter</title>
</head>
<body>
  <input type="text" id="searchBox" placeholder="Search for names...">
  <ul id="nameList">
    <li>Emma</li>
    <li>Noah</li>
    <li>Olivia</li>
    <li>Liam</li>
    <li>Ava</li>
    <!-- More names -->
  </ul>
  <script>
    document.getElementById('searchBox').addEventListener('keyup',
function() {
      const searchValue = this.value.toLowerCase();
      const listItems = document.querySelectorAll('#nameList li');
      listItems.forEach(item => {
        const text = item.textContent.toLowerCase();
        const isVisible = text.includes(searchValue);
        item.style.display = isVisible ? '' : 'none';
      });
    });
  </script>
</body>
</html>
```

These exercises introduce more advanced web development concepts and encourage the exploration of APIs, storage mechanisms, and the canvas API for graphics. They're suitable for learners looking to expand their JavaScript knowledge through practical, real-world applications.

User

create 10 more

ChatGPT

Here are 10 additional JavaScript exercises that delve into more advanced topics such as animations with CSS and JavaScript, working with maps, graphs, and more complex event handling. These exercises aim to provide a deeper understanding of JavaScript and its interaction with HTML and CSS to create dynamic and interactive web applications.

31. Animating a Shape Using JavaScript

Objective: Use JavaScript to animate a shape moving across the screen.

```
<!DOCTYPE html>
<html>
<head>
    <title>Shape Animation</title>
    <style>
        #animate { width: 50px; height: 50px; background-color: red; position: relative; }
    </style>
</head>
<body>
<div id="animate"></div>
<script>
function moveShape() {
  var elem = document.getElementById("animate");
  var pos = 0;
  var id = setInterval(frame, 5);
  function frame() {
    if (pos == 350) {
      clearInterval(id);
    } else {
      pos++;
      elem.style.top = pos + 'px';
      elem.style.left = pos + 'px';
    }
  }
}
moveShape();
</script>
</body>
</html>
```

32. Creating a Collapsible Sidebar Menu

Objective: Create a collapsible sidebar menu that expands and collapses when clicked.

```
<!DOCTYPE html>
<html>
<head>
  <title>Collapsible Sidebar</title>
  <style>
    #sidebar { width: 250px; height: 100vh; background: #333; transition:
width 0.5s ease; }
    #sidebar.collapsed { width: 50px; }
    #sidebarToggle { cursor: pointer; padding: 10px; color: white; }
  </style>
</head>
<body>
<div id="sidebar" class="collapsed">
  <div id="sidebarToggle">Toggle</div>
</div>
<script>
document.getElementById('sidebarToggle').onclick = function() {
  var sidebar = document.getElementById('sidebar');
  if (sidebar.classList.contains('collapsed')) {
    sidebar.classList.remove('collapsed');
  } else {
    sidebar.classList.add('collapsed');
  }
};
</script>
</body>
</html>
```

33. Dynamic Graph Drawing

Objective: Dynamically draw a graph based on user input values using the <canvas> element.

```
<!DOCTYPE html>
<html>
<head>
  <title>Dynamic Graph</title>
</head>
<body>
```

```
<canvas id="myCanvas" width="400" height="400" style="border:1px solid #000;">
Your browser does not support the canvas element.
</canvas>
<script>
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
var startPoint = { x: 50, y: 350 };
var points = [{x: 100, y: 300}, {x: 150, y: 200}, {x: 200, y: 100}, {x: 250, y: 50}];
// Draw the graph
ctx.moveTo(startPoint.x, startPoint.y);
points.forEach(function(point) {
    ctx.lineTo(point.x, point.y);
    ctx.stroke();
});
</script>
</body>
</html>
```

34. Using Web Workers for Background Tasks

Objective: Utilize Web Workers to perform a time-consuming calculation in the background without blocking the UI.

Create a file named worker.js:

```
// worker.js
self.onmessage = function(e) {
  let result = 0;
  // Example of a time-consuming task
  for (let i = 0; i < e.data; i++) {
    result += i;
  }
  self.postMessage(result);
};
```

And the main HTML file:

```
<!DOCTYPE html>
<html>
<head>
```

```
  <title>Web Worker Example</title>
</head>
<body>
<button onclick="startWorker()">Start Worker</button>
<div id="result"></div>
<script>
var worker;
function startWorker() {
 if(typeof(Worker) !== "undefined") {
  if(typeof(worker) == "undefined") {
   worker = new Worker("worker.js");
   worker.postMessage(1000000000); // Send data to our worker.
  }
  worker.onmessage = function(event) {
   document.getElementById("result").innerHTML = "Result: " +
event.data;
  };
 } else {
  document.getElementById("result").innerHTML = "Sorry, your browser
does not support Web Workers...";
 }
}
</script>
</body>
</html>
```

35. Implementing a Simple Game Loop

Objective: Create a simple game loop that updates and renders a moving object on the canvas.

```
<!DOCTYPE html>
<html>
<head>
  <title>Simple Game Loop</title>
</head>
<body>
<canvas id="gameCanvas" width="480" height="320"></canvas>
<script>
```

```
var canvas = document.getElementById("gameCanvas");
var ctx = canvas.getContext("2d");
var x = canvas.width/2;
var y = canvas.height - 30;
var dx = 2;
var dy = -2;
function drawBall() {
    ctx.beginPath();
    ctx.arc(x, y, 10, 0, Math.PI*2);
    ctx.fillStyle = "#0095DD";
    ctx.fill();
    ctx.closePath();
}
function update() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    drawBall();
    x += dx;
    y += dy;
    if(x + dx > canvas.width-10 || x + dx < 10) {
        dx = -dx;
    }
    if(y + dy > canvas.height-10 || y + dy < 10) {
        dy = -dy;
    }
    requestAnimationFrame(update);
}
update();
</script>
</body>
</html>
```

36. Creating a Simple Map with Leaflet.js

Objective: Integrate Leaflet.js to display a simple map with a marker denoting a specific location.

First, include the Leaflet CSS and JS in your HTML:

```
<link rel="stylesheet" href="https://unpkg.com/leaflet/dist/leaflet.css" />
<script src="https://unpkg.com/leaflet/dist/leaflet.js"></script>
```

Then, add the map container and script to initialize the map:

```
<div id="mapid" style="width: 600px; height: 400px;"></div>
<script>
var mymap = L.map('mapid').setView([51.505, -0.09], 13);
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
   attribution: '© OpenStreetMap contributors'
}).addTo(mymap);
L.marker([51.5, -0.09]).addTo(mymap)
   .bindPopup('A pretty CSS3 popup.<br> Easily customizable.')
   .openPopup();
</script>
```

37. Creating a Custom Audio Player

Objective: Build a custom audio player with play, pause, and volume controls.

```
<!DOCTYPE html>
<html>
<head>
   <title>Custom Audio Player</title>
</head>
<body>
<audio id="audioPlayer" src="your-audio-file.mp3"></audio>
<button onclick="playAudio()">Play</button>
<button onclick="pauseAudio()">Pause</button>
<input type="range" id="volumeControl" min="0" max="1" step="0.1"
onchange="changeVolume(this.value)" />
<script>
function playAudio() {
   document.getElementById("audioPlayer").play();
}
function pauseAudio() {
   document.getElementById("audioPlayer").pause();
}
function changeVolume(volume) {
   document.getElementById("audioPlayer").volume = volume;
}
</script>
```

```
</body>
</html>
```

38. Interactive Color Picker

Objective: Create an interactive color picker that changes the background color of a div in real-time.

```
<!DOCTYPE html>
<html>
<head>
   <title>Interactive Color Picker</title>
</head>
<body>
<input type="color" id="colorPicker">
<div id="colorfulDiv" style="width: 200px; height: 200px; border: 1px solid
#000;"></div>
<script>
document.getElementById('colorPicker').addEventListener('input', function()
{
   document.getElementById('colorfulDiv').style.backgroundColor =
this.value;
});
</script>
</body>
</html>
```

39. Simple Weather App

Objective: Fetch weather data from an API and display it. (Use a mock API or a service like OpenWeatherMap; remember to include your own API key if required).

```
<!DOCTYPE html>
<html>
<head>
   <title>Simple Weather App</title>
</head>
<body>
<div id="weather"></div>
```

```
<script>
async function fetchWeather() {
   const response = await
fetch('https://api.openweathermap.org/data/2.5/weather?q=London&appid=
your_api_key&units=metric');
   const data = await response.json();
   document.getElementById('weather').innerHTML = `Temperature in
London: ${data.main.temp}°C`;
}
fetchWeather();
</script>
</body>
</html>
```

40. Mouse Movement Effect

Objective: Create an effect that responds to the user's mouse movement over a specific element.

```
<!DOCTYPE html>
<html>
<head>
   <title>Mouse Movement Effect</title>
</head>
<body>
<div id="effectDiv" style="width: 400px; height: 200px; position: relative;
background: #ccc;">
</div>
<script>
document.getElementById('effectDiv').addEventListener('mousemove',
function(e) {
   var rect = this.getBoundingClientRect();
   var x = e.clientX - rect.left; // x position within the element.
   var y = e.clientY - rect.top;  // y position within the element.
   this.style.background = `radial-gradient(circle at ${x}px ${y}px, #eee,
#333)`;
});
</script>
</body>
```

```
</html>
```

These exercises cover a wide range of advanced web development concepts, from animations and custom UI components to working with external APIs and interactive effects. They are designed to challenge and expand your understanding of JavaScript, CSS, and HTML in creating more dynamic and interactive web experiences.

User

create 10 more

ChatGPT

Continuing with advanced and diverse JavaScript exercises, these tasks aim to further enhance your skills in areas like API integration, complex animations, data visualization, and more sophisticated event handling techniques. Each exercise is designed to challenge and improve your understanding of modern web development practices.

41. Real-Time Currency Converter

Objective: Build a real-time currency converter that fetches exchange rates from an API and updates the conversion as the user types.

```
<!DOCTYPE html>
<html>
<head>
   <title>Real-Time Currency Converter</title>
</head>
<body>
Amount: <input type="number" id="amount" placeholder="Enter amount">
<select id="fromCurrency">
   <option value="USD">USD</option>
   <option value="EUR">EUR</option>
   <!-- Add more currencies as needed -->
</select>
to
<select id="toCurrency">
   <option value="EUR">EUR</option>
   <option value="USD">USD</option>
   <!-- Add more currencies as needed -->
</select>
<button onclick="convertCurrency()">Convert</button>
```

```
<div id="result"></div>
<script>
async function fetchExchangeRate(from, to) {
   const response = await
fetch(`https://api.exchangerate-api.com/v4/latest/${from}`);
   const data = await response.json();
   return data.rates[to];
}
async function convertCurrency() {
   const amount = document.getElementById('amount').value;
   const fromCurrency = document.getElementById('fromCurrency').value;
   const toCurrency = document.getElementById('toCurrency').value;
   const rate = await fetchExchangeRate(fromCurrency, toCurrency);
   document.getElementById('result').innerText = `${amount}
${fromCurrency} = ${amount * rate} ${toCurrency}`;
}
</script>
</body>
</html>
```

42. Dynamic Pie Chart with Chart.js

Objective: Use Chart.js to dynamically generate a pie chart based on user input data.

First, include Chart.js in your HTML:

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
```

Then, add the HTML and JavaScript to generate the chart:

```
<!DOCTYPE html>
<html>
<head>
   <title>Dynamic Pie Chart</title>
</head>
<body>
<canvas id="myChart" width="400" height="400"></canvas>
<script>
var ctx = document.getElementById('myChart').getContext('2d');
var myChart = new Chart(ctx, {
   type: 'pie',
  data: {
    labels: ['Red', 'Blue', 'Yellow'],
    datasets: [{
      label: '# of Votes',
      data: [12, 19, 3],
      backgroundColor: [
        'rgba(255, 99, 132, 0.2)',
        'rgba(54, 162, 235, 0.2)',
        'rgba(255, 206, 86, 0.2)'
      ],
      borderColor: [
        'rgba(255, 99, 132, 1)',
        'rgba(54, 162, 235, 1)',
        'rgba(255, 206, 86, 1)'
      ],
      borderWidth: 1
    }]
  },
  options: {
    scales: {
      y: {
        beginAtZero: true
      }
    }
  }
});
</script>
</body>
</html>
```

43. Interactive SVG Map

Objective: Create an interactive SVG map where regions change color upon mouse hover and display information on click.

```
<!DOCTYPE html>
<html>
<head>
  <title>Interactive SVG Map</title>
```

```
</head>
<body>
<svg width="100" height="100">
  <rect id="rect1" width="100" height="100" style="fill:blue;" />
</svg>
<script>
document.getElementById('rect1').addEventListener('mouseover', function()
{
    this.style.fill = 'red';
});
document.getElementById('rect1').addEventListener('mouseout', function()
{
    this.style.fill = 'blue';
});
document.getElementById('rect1').addEventListener('click', function() {
    alert('Region information displayed here.');
});
</script>
</body>
</html>
```

44. Parallax Scrolling Effect

Objective: Implement a parallax scrolling effect where the background moves at a different speed than the foreground content.

```
<!DOCTYPE html>
<html>
<head>
    <title>Parallax Effect</title>
    <style>
        .parallax { background-image: url('your-image-url.jpg'); height: 500px;
background-attachment: fixed; background-position: center;
background-repeat: no-repeat; background-size: cover; }
        .content { height: 1500px; background-color: rgba(255,255,255,0.8); }
    </style>
</head>
<body>
<div class="parallax"></div>
```

```
<div class="content">
   <h1>Scroll down to see the effect</h1>
</div>
</body>
</html>
```

45. Scroll-triggered Animation

Objective: Create an animation that triggers when the user scrolls to a specific section of the page.

```
<!DOCTYPE html>
<html>
<head>
   <title>Scroll-triggered Animation</title>
   <style>
      #animateMe { opacity: 0; transition: opacity 2s; }
      .visible { opacity: 1; }
   </style>
</head>
<body>
<div style="height: 1500px;">
   Scroll to see the animation.
</div>
<div id="animateMe">
   This text will fade in when you reach this point.
</div>
<script>
window.addEventListener('scroll', function() {
   var element = document.getElementById('animateMe');
   var position = element.getBoundingClientRect();
   if(position.top < window.innerHeight && position.bottom >= 0) {
      element.classList.add('visible');
   }
});
</script>
</body>
</html>
```

46. WebSocket Chat Application

Objective: Create a simple chat application using WebSockets for real-time communication between clients and server.

You'll need a WebSocket server for this; here's an example using JavaScript for the client side:

```
<!DOCTYPE html>
<html>
<head>
   <title>WebSocket Chat</title>
</head>
<body>
<input type="text" id="messageInput" placeholder="Type a message...">
<button onclick="sendMessage()">Send</button>
<div id="chat"></div>
<script>
var ws = new WebSocket('ws://your-websocket-server-url');
ws.onmessage = function(event) {
   var messages = document.getElementById('chat');
   messages.innerHTML += '<div>' + event.data + '</div>';
};
function sendMessage() {
   var input = document.getElementById('messageInput');
   ws.send(input.value);
   input.value = '';
}
</script>
</body>
</html>
```

47. Drag-and-Drop File Upload

Objective: Implement a drag-and-drop interface for users to upload files.

```
<!DOCTYPE html>
<html>
<head>
   <title>Drag and Drop File Upload</title>
   <style>
```

```
        #dropArea { border: 2px dashed #ccc; width: 300px; height: 200px; text-align: center; line-height: 200px; }
    </style>
</head>
<body>
<div id="dropArea">Drop files here</div>
<script>
var dropArea = document.getElementById('dropArea');
dropArea.addEventListener('dragover', function(e) {
    e.preventDefault();
    dropArea.style.border = "2px solid #000";
});
dropArea.addEventListener('dragleave', function(e) {
    dropArea.style.border = "2px dashed #ccc";
});
dropArea.addEventListener('drop', function(e) {
    e.preventDefault();
    dropArea.style.border = "2px dashed #ccc";
    // Process file here
    console.log(e.dataTransfer.files);
});
</script>
</body>
</html>
```

48. 3D Model Viewer with Three.js

Objective: Integrate Three.js to create a simple 3D model viewer.

First, include Three.js:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>
```

Then, set up the viewer:

```
<!DOCTYPE html>
<html>
<head>
    <title>3D Model Viewer</title>
</head>
```

```
<body>
<div id="viewer" style="width: 600px; height: 400px;"></div>
<script>
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera(75, 600/400, 0.1, 1000);
var renderer = new THREE.WebGLRenderer();
renderer.setSize(600, 400);
document.getElementById('viewer').appendChild(renderer.domElement);
var geometry = new THREE.BoxGeometry();
var material = new THREE.MeshBasicMaterial({color: 0x00ff00});
var cube = new THREE.Mesh(geometry, material);
scene.add(cube);
camera.position.z = 5;
function animate() {
  requestAnimationFrame(animate);
  cube.rotation.x += 0.01;
  cube.rotation.y += 0.01;
  renderer.render(scene, camera);
}
animate();
</script>
</body>
</html>
```

49. Audio Visualization

Objective: Create a simple audio visualization using the Web Audio API.

```
<!DOCTYPE html>
<html>
<head>
  <title>Audio Visualization</title>
</head>
<body>
<audio src="your-audio-file.mp3" controls></audio>
<canvas id="canvas" width="800" height="600"></canvas>
<script>
var context = new AudioContext();
var analyser = context.createAnalyser();
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');
var audio = document.querySelector('audio');
audio.addEventListener('play', function() {
    var audioSrc = context.createMediaElementSource(audio);
    audioSrc.connect(analyser);
    analyser.connect(context.destination);
    draw();
});
function draw() {
    requestAnimationFrame(draw);
    var freqData = new Uint8Array(analyser.frequencyBinCount);
    analyser.getByteFrequencyData(freqData);
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    ctx.fillStyle = '#00ff00';
    for (var i = 0; i < freqData.length; i++) {
        var barHeight = freqData[i];
        ctx.fillRect(i * 3, canvas.height - barHeight, 2, barHeight);
    }
}
</script>
</body>
</html>
```

50. Dynamic Content Tabs

Objective: Create a tabbed interface where clicking on a tab header displays the content associated with that tab.

```
<!DOCTYPE html>
<html>
<head>
    <title>Dynamic Content Tabs</title>
    <style>
        .tab-content { display: none; }
        .active-tab { display: block; }
    </style>
</head>
<body>
```

```
<div class="tab">
  <button class="tablinks" onclick="openTab(event, 'Tab1')">Tab 1</button>
  <button class="tablinks" onclick="openTab(event, 'Tab2')">Tab 2</button>
  <button class="tablinks" onclick="openTab(event, 'Tab3')">Tab 3</button>
</div>
<div id="Tab1" class="tab-content">
  <h3>Tab 1</h3>
  <p>Content for Tab 1.</p>
</div>
<div id="Tab2" class="tab-content">
  <h3>Tab 2</h3>
  <p>Content for Tab 2.</p>
</div>
<div id="Tab3" class="tab-content">
  <h3>Tab 3</h3>
  <p>Content for Tab 3.</p>
</div>
<script>
function openTab(evt, tabName) {
  var i, tabcontent, tablinks;
  tabcontent = document.getElementsByClassName("tab-content");
  for (i = 0; i < tabcontent.length; i++) {
    tabcontent[i].style.display = "none";
  }
  tablinks = document.getElementsByClassName("tablinks");
  for (i = 0; i < tablinks.length; i++) {
    tablinks[i].className = tablinks[i].className.replace(" active", "");
  }
  document.getElementById(tabName).style.display = "block";
  evt.currentTarget.className += " active";
}
</script>
</body>
</html>
```

These exercises delve into more specialized areas of web development, providing a broad overview of what's possible with modern JavaScript, HTML, and CSS. They're aimed at those looking to push their skills further, incorporating real-time data handling, 3D rendering, audio processing, and more interactive elements into web applications.

User

create 10 more

ChatGPT

Continuing to expand the horizons of web development skills, here are 10 more exercises that delve into various advanced JavaScript functionalities, including service workers, progressive web app (PWA) features, advanced animations, and more complex API integrations. These exercises are designed to challenge and refine your understanding of creating sophisticated web applications.

51. Service Worker Caching

Objective: Implement a service worker to cache important assets of a web page, enabling offline access.

First, create your service worker file, sw.js:

```
const CACHE_NAME = 'v1';
const urlsToCache = [
  '/',
  '/style.css',
  '/script.js',
  // Add other assets and pages as needed
];
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        console.log('Opened cache');
        return cache.addAll(urlsToCache);
      })
  );
});
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then(response => {
        if (response) {
          return response;
        }
        return fetch(event.request);
      })
  );
});
```

Then, register the service worker in your main JavaScript file:

```
<script>
  if ('serviceWorker' in navigator) {
    window.addEventListener('load', () => {
      navigator.serviceWorker.register('/sw.js').then(registration => {
        console.log('ServiceWorker registration successful with scope: ',
registration.scope);
      }, err => {
        console.log('ServiceWorker registration failed: ', err);
      });
    });
  }
</script>
```

52. Progressive Web App (PWA)

Objective: Convert a simple website into a PWA by adding a web app manifest and configuring it to be installable.

Create a manifest file manifest.json:

json

```
{
  "short_name": "PWA Demo",
  "name": "Progressive Web Application Demonstration",
  "icons": [
    {
      "src": "icon-192x192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "icon-512x512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": "/index.html",
  "background_color": "#ffffff",
  "display": "standalone",
  "scope": "/",
  "theme_color": "#000000"
}
```

Include the manifest in your HTML:

```
<link rel="manifest" href="/manifest.json">
```

53. Web Speech API for Voice Commands

Objective: Use the Web Speech API to add voice command functionality that changes the background color of the page.

```
<script>
  const colors = ['red', 'green', 'blue', 'yellow', 'purple', 'orange'];
  const recognition = new webkitSpeechRecognition();
  recognition.onresult = function(event) {
    const color = event.results[0][0].transcript;
    if (colors.includes(color.toLowerCase())) {
      document.body.style.backgroundColor = color;
    } else {
      alert('Color not recognized.');
    }
  };
  function startListening() {
    recognition.start();
  }
</script>
<button onclick="startListening()">Change Background Color</button>
```

54. Intersection Observer API

Objective: Use the Intersection Observer API to load images lazily as they enter the viewport.

```
<img class="lazy-load" data-src="image1.jpg" alt="Lazy Image 1">
```

```
<img class="lazy-load" data-src="image2.jpg" alt="Lazy Image 2">
<!-- More lazy-load images -->
<script>
  const observer = new IntersectionObserver((entries, observer) => {
    entries.forEach(entry => {
      if (entry.isIntersecting) {
        const img = entry.target;
        img.src = img.getAttribute('data-src');
        observer.unobserve(img);
      }
    });
  });
  document.querySelectorAll('.lazy-load').forEach(img => {
    observer.observe(img);
  });
</script>
```

55. WebGL for 3D Graphics

Objective: Integrate a simple 3D object using WebGL through the Three.js library.

Ensure you have Three.js included, then:

```
<div id="webglOutput"></div>
<script>
  var scene = new THREE.Scene();
  var camera = new THREE.PerspectiveCamera(75,
window.innerWidth/window.innerHeight, 0.1, 1000);
  var renderer = new THREE.WebGLRenderer();
  renderer.setSize(window.innerWidth, window.innerHeight);

document.getElementById('webglOutput').appendChild(renderer.domEleme
nt);
  var geometry = new THREE.BoxGeometry();
  var material = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
  var cube = new THREE.Mesh(geometry, material);
  scene.add(cube);
  camera.position.z = 5;
  function animate() {
    requestAnimationFrame(animate);
    cube.rotation.x += 0.01;
    cube.rotation.y += 0.01;
    renderer.render(scene, camera);
  }
  animate();
</script>
```

56. Advanced CSS Animation

Objective: Create a complex CSS animation that simulates a loading sequence.

```
<style>
  @keyframes spin {
    from { transform: rotate(0deg); }
    to { transform: rotate(360deg); }
  }
  .loader {
    border: 16px solid #f3f3f3;
    border-radius: 50%;
    border-top: 16px solid #3498db;
    width: 120px;
    height: 120px;
    animation: spin 2s linear infinite;
  }
</style>
<div class="loader"></div>
```

57. Dynamic Form Creation

Objective: Dynamically add input fields to a form with JavaScript and submit the form data using fetch.

```
<form id="dynamicForm">
  <button type="button" onclick="addInputField()">Add Input</button>
  <button type="submit">Submit</button>
</form>
<script>
```

```
  document.getElementById('dynamicForm').addEventListener('submit', e
=> {
    e.preventDefault();
    const formData = new FormData(e.target);
    const data = {};
    formData.forEach((value, key) => { data[key] = value; });
    console.log('Form data', data);
    // Use fetch to send data
  });
  function addInputField() {
    const input = document.createElement('input');
    input.type = 'text';
    input.name = 'dynamicInput[]';
    document.getElementById('dynamicForm').appendChild(input);
  }
</script>
```

58. Canvas Drawing App

Objective: Create a simple drawing app using the HTML canvas where users can draw with their mouse.

```
<canvas id="drawingCanvas" width="800" height="600" style="border:1px
solid #000;"></canvas>
<script>
  const canvas = document.getElementById('drawingCanvas');
  const ctx = canvas.getContext('2d');
  let drawing = false;
  canvas.addEventListener('mousedown', () => drawing = true);
  canvas.addEventListener('mouseup', () => drawing = false);
  canvas.addEventListener('mousemove', e => {
    if (drawing) {
      ctx.lineTo(e.offsetX, e.offsetY);
      ctx.stroke();
      ctx.beginPath();
      ctx.moveTo(e.offsetX, e.offsetY);
    }
  });
</script>
```

59. Using Fetch API with Async/Await

Objective: Fetch data from an API using async/await and display it on the page.

```
<div id="apiData">Loading...</div>
<script>
  async function fetchData() {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    document.getElementById('apiData').innerHTML = JSON.stringify(data,
null, 2);
  }
  fetchData();
</script>
```

60. Custom Video Background

Objective: Implement a full-screen video background for a web page section.

```
<style>
  .video-background {
    position: fixed;
    right: 0;
    bottom: 0;
    min-width: 100%;
    min-height: 100%;
    z-index: -100;
  }
</style>
<video autoplay loop muted class="video-background">
  <source src="background.mp4" type="video/mp4">
</video>
<div>Your content here</div>
```

These exercises introduce a variety of advanced web development concepts, including offline capabilities, 3D rendering, real-time communication, and much more, aimed at enhancing your skill set and understanding of complex web functionalities.

User

create 10 more
ChatGPT
Advancing further into the realms of web development, here are 10 more exercises that push the boundaries of what's possible with JavaScript, HTML, and CSS. These exercises introduce new challenges such as creating a web-based game, integrating machine learning models, and enhancing user interaction through sophisticated animations and data visualizations.

61. Memory Card Game

Objective: Build a simple memory card game where the user flips two cards at a time to find a match.

```
<!DOCTYPE html>
<html>
<head>
    <title>Memory Card Game</title>
    <style>
        .card { width: 100px; height: 100px; background-color: #f0f0f0; margin:
10px; display: inline-block; vertical-align: top; }
        .card.flipped { background-color: #0f0; }
    </style>
</head>
<body>
    <div id="gameBoard"></div>
    <script>
        const cards = ['A', 'A', 'B', 'B', 'C', 'C'];
        let flippedCards = [];
        function shuffle(array) {
            for (let i = array.length - 1; i > 0; i--) {
                const j = Math.floor(Math.random() * (i + 1));
                [array[i], array[j]] = [array[j], array[i]];
            }
        }
        function setupBoard() {
            shuffle(cards);
            const gameBoard = document.getElementById('gameBoard');
            cards.forEach((card, index) => {
                const cardElement = document.createElement('div');
                cardElement.classList.add('card');
                cardElement.dataset.cardValue = card;
                cardElement.addEventListener('click', flipCard);
                gameBoard.appendChild(cardElement);
            });
        }
        function flipCard() {
            if (flippedCards.length === 2) return;
            this.classList.add('flipped');
            flippedCards.push(this);
            if (flippedCards.length === 2) {
                checkForMatch();
            }
        }
        function checkForMatch() {
            if (flippedCards[0].dataset.cardValue ===
flippedCards[1].dataset.cardValue) {
                flippedCards.forEach(card => card.removeEventListener('click',
flipCard));
                flippedCards = [];
            } else {
                setTimeout(() => {
                    flippedCards.forEach(card => card.classList.remove('flipped'));
                    flippedCards = [];
                }, 1000);
            }
        }
        setupBoard();
    </script>
</body>
</html>
```

62. Real-Time Chat Application with Socket.IO

Objective: Implement a basic real-time chat application using Socket.IO to enable live messaging between users.

Note: This requires a server setup with Node.js and Socket.IO. Below is the client-side example.

```
<!DOCTYPE html>
<html>
<head>
  <title>Real-Time Chat</title>
</head>
<body>
  <ul id="messages"></ul>
  <form id="form" action="">
    <input id="input" autocomplete="off" /><button>Send</button>
  </form>
  <script src="/socket.io/socket.io.js"></script>
  <script>
    var socket = io();
    var form = document.getElementById('form');
    var input = document.getElementById('input');
    form.addEventListener('submit', function(e) {
      e.preventDefault();
      if (input.value) {
        socket.emit('chat message', input.value);
        input.value = '';
      }
    });
    socket.on('chat message', function(msg) {
      var item = document.createElement('li');
      item.textContent = msg;
      document.getElementById('messages').appendChild(item);
      window.scrollTo(0, document.body.scrollHeight);
    });
  </script>
</body>
</html>
```

63. Browser-Based Machine Learning with TensorFlow.js

Objective: Utilize TensorFlow.js to integrate a pre-trained machine learning model into a web application for image classification.
Note: You'll need to include TensorFlow.js and load a pre-trained model.
This example shows a very basic setup.

```
<!DOCTYPE html>
<html>
<head>
   <title>Image Classification with TensorFlow.js</title>
   <script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
</head>
<body>
   <input type="file" id="imageUpload">
   <script>
      let model;
      async function loadModel() {
         model = await
tf.loadLayersModel('https://storage.googleapis.com/tfjs-models/tfjs/mobilen
et_v1_0.25_224/model.json');

document.getElementById('imageUpload').addEventListener('change',
predictImage);
      }
      async function predictImage(event) {
         let image = event.target.files[0];
         let tensor = tf.browser.fromPixels(await
createImageBitmap(image)).resizeNearestNeighbor([224,
224]).toFloat().expandDims();
         let prediction = await model.predict(tensor).data();
         console.log(prediction); // Do something with the prediction
      }
      loadModel();
   </script>
</body>
```

```
</html>
```

64. Customizable Data Dashboard

Objective: Create a customizable data dashboard where users can add and configure widgets to display different data visualizations.

This exercise requires a more complex setup, including a backend to store user configurations and a frontend capable of dynamically adding, removing, and configuring widgets. Here's a conceptual starting point for the frontend:

```
<!DOCTYPE html>
<html>
<head>
  <title>Data Dashboard</title>
</head>
<body>
  <div id="dashboard"></div>
  <button onclick="addWidget()">Add Widget</button>
  <script>
    function addWidget() {
      const widget = document.createElement('div');
      widget.innerText = 'New Widget';
      // Add functionality to customize the widget, e.g., type, data source
      document.getElementById('dashboard').appendChild(widget);
    }
  </script>
</body>
</html>
```

65. Interactive 3D Globe with Three.js

Objective: Utilize Three.js to create an interactive 3D globe that users can rotate and zoom.

Note: This requires Three.js. The example below is simplified:

```
<!DOCTYPE html>
<html>
<head>
  <title>3D Globe</title>
```

```
   <script
src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></scri
pt>
</head>
<body>
   <div id="globeContainer"></div>
   <script>
      var scene = new THREE.Scene();
      var camera = new THREE.PerspectiveCamera(75,
window.innerWidth/window.innerHeight, 0.1, 1000);
      var renderer = new THREE.WebGLRenderer();
      renderer.setSize(window.innerWidth, window.innerHeight);

document.getElementById('globeContainer').appendChild(renderer.domEle
ment);
      var geometry = new THREE.SphereGeometry(5, 32, 32);
      var material = new THREE.MeshBasicMaterial({color: 0x00ff00});
      var sphere = new THREE.Mesh(geometry, material);
      scene.add(sphere);
      camera.position.z = 15;
      function animate() {
         requestAnimationFrame(animate);
         sphere.rotation.x += 0.01;
         sphere.rotation.y += 0.01;
         renderer.render(scene, camera);
      }
      animate();
   </script>
</body>
</html>
```

66. Dynamic Spreadsheet

Objective: Implement a basic spreadsheet that allows users to add data, with automatic calculation of sums and averages.
This exercise would ideally use a JavaScript library like Handsontable or simple HTML and JavaScript for DOM manipulation. Here's a conceptual start:

```
<!DOCTYPE html>
<html>
<head>
  <title>Dynamic Spreadsheet</title>
</head>
<body>
  <table id="spreadsheet">
    <tr>
      <th>Data</th>
      <th>Sum</th>
      <th>Average</th>
    </tr>
    <tr>
      <td contenteditable="true">0</td>
      <td id="sum">0</td>
      <td id="average">0</td>
    </tr>
  </table>
  <button onclick="addRow()">Add Row</button>
  <script>
    function addRow() {
      const spreadsheet = document.getElementById('spreadsheet');
      const newRow = spreadsheet.insertRow(-1);
      const newCell = newRow.insertCell(0);
      newCell.contentEditable = "true";
      newCell.textContent = "0";
      updateCalculations();
      // Add event listeners to update calculations on cell change
    }
    function updateCalculations() {
      // Implement sum and average calculations
    }
  </script>
</body>
</html>
```