# **Disclaimer:**

COPYRIGHT NOTICE

©All content, materials, and intellectual property contained within this document, including but not limited to text, images, diagrams, graphics, code, and any other original works, are protected by copyright laws and international treaties. This intellectual property is the sole and exclusive property of SRIKANTH TEKUMUDI

Unauthorized use, reproduction, distribution, modification, or transmission of any portion of this document, in any form or by any means, electronic or mechanical, without the prior written permission of SRIKANTH TEKUMUDI, is strictly prohibited and may result in severe civil and criminal penalties.

By accessing, viewing, or using this document, you acknowledge and agree to abide by all applicable copyright laws and the terms and conditions outlined herein. This copyright notice serves as a legal warning and reminder that the unauthorized use of this intellectual property is strictly prohibited and will be vigorously enforced.

Thank you for respecting the intellectual property rights of SRIKANTH TEKUMUDI

### **1. Currying**

| function curry(func) {  const curriedFunc = function(...args) {  if (args.length >= func.length) {  return func.apply(this, args);  } else {  return (...moreArgs) => curriedFunc.call(this, ...args, ...moreArgs);  }  };  return curriedFunc; }  let concatenateStrings = (str1, str2, str3) => {  return `${str1}\_${str2}\_${str3}`; };  let curriedConcatenate = curry(concatenateStrings);  console.log(curriedConcatenate("Hello")("World", "2023")); console.log(curriedConcatenate("Hello")("World")("2023")); |
| --- |

### 

### **2. Array.flat**

| function flattenArray(arr, depth = 1) {  if (depth === 0) return arr;  const flattenedArray = [];  for (let element of arr) {  if (Array.isArray(element)) {  flattenedArray.push(...flattenArray(element, depth - 1));  } else {  flattenedArray.push(element);  }  }  return flattenedArray; }  const nestedArray = [1, [2], [3, [4]], [5, [6, [7, [8]]]]];  console.log(flattenArray(nestedArray)); |
| --- |

### **3.Deep copy object**

| function deepCopy(obj) {  let newObj = {};  for (let key in obj) {  if (obj.hasOwnProperty(key)) {  if (typeof obj[key] === 'object') {  newObj[key] = deepCopy(obj[key]);  } else {  newObj[key] = obj[key];  }  }  }  return newObj; }  let car = {  brand: 'BMW',  model: 'X5',  year: 2022,  address: {  city: 'Munich',  country: 'Germany'  } };  let copiedCar = deepCopy(car);  console.log(copiedCar); |
| --- |

### **4. Throttle**

| function throttle(func, wait) {  let timerId;  let lastArgs = [];   return function throttledFunction(...args) {  if (timerId) {  lastArgs = args;  return;  }   func(...args);  timerId = setTimeout(() => {  timerId = null;  if (lastArgs.length) {  func(...lastArgs);  lastArgs = [];  }  }, wait);  }; }  *// Example usage* function logMessage(message) {  console.log(message); }  const throttledLog = throttle(logMessage, 1000);  throttledLog('Message 1'); throttledLog('Message 2'); throttledLog('Message 3');  *// After 1 second, only the last message will be logged* |
| --- |

### **5. debounce()**

| function debounce(func, wait) {  let timeoutId;   return function debouncedFunction(...args) {  clearTimeout(timeoutId);  timeoutId = setTimeout(() => {  func.apply(this, args);  }, wait);  }; }  *// Example usage* function saveChanges(data) {  console.log('Saving changes:', data); }  const debouncedSave = debounce(saveChanges, 1000);  *// Simulating multiple rapid calls to debouncedSave* debouncedSave('Data 1'); debouncedSave('Data 2'); debouncedSave('Data 3');  *// After 1 second, only the last call will be executed* |
| --- |

### **6. Shuffle() an array**

| function shuffle(arr) {  for(let [idx, item] of arr.entries()) {  const randomIdx = idx + Math.floor(Math.random() \* (arr.length - idx));  *// console.log(randomIdx)*  [arr[idx], arr[randomIdx]] = [arr[randomIdx], arr[idx]]  }   return arr } |
| --- |

### **7. What is Composition? create a pipe()**

| *//left to right=> pipe* *// right to left==> compose* const add5 = (number) => number + 5; const multiplyBy2 = (number) => number \* 2; const square = (number) => number \*\* 2; const subtract10 = (number) => number - 10;  *// Example using compose* const calculateResultCompose = compose(subtract10, square, multiplyBy2, add5); const resultCompose = calculateResultCompose(3); console.log(resultCompose); *// 56*  *// Example using pipe* const calculateResultPipe = pipe(add5, multiplyBy2, square, subtract10); const resultPipe = calculateResultPipe(3); console.log(resultPipe); *// 42*  function compose(...funcs) {  return (args) => {  return funcs.reduceRight((acc, func) => func.call(this, acc), args);  }; }  function pipe(...funcs) {  return (args) => {  return funcs.reduce((acc, func) => func.call(this, acc), args);  }; } |
| --- |

### **8. General memoization function**

| function memoize(func) {  let cache = {};   return function memoizedFunction(...args) {  let key = JSON.stringify(args);   if (!(key in cache)) {  cache[key] = func.apply(this, args);  }   return cache[key];  }; }  *// Example usage* function calculateFactorial(n) {  if (n === 0 || n === 1) {  return 1;  }   return n \* calculateFactorial(n - 1); }  const memoizedFactorial = memoize(calculateFactorial);  console.log(memoizedFactorial(5)); *// 120 (calculated)* console.log(memoizedFactorial(5)); *// 120 (cached)* console.log(memoizedFactorial(6)); *// 720 (calculated)* console.log(memoizedFactorial(6)); *// 720 (cached)* |
| --- |

### **9. Simple DOM wrapper**

| function DOMWrapper(element) {  const wrapper = {  setStyle(property, value) {  element.style[property] = value;  return wrapper;  }  };   return wrapper; }  *// Example usage* const myElement = document.getElementById('myElement'); const myWrapper = DOMWrapper(myElement);  myWrapper.setStyle('color', 'red')  .setStyle('font-size', '18px')  .setStyle('background-color', 'yellow'); |
| --- |

### **10. Event Emitter**

| class EventEmitter {  constructor() {  this.eventMap = new Map();  }   subscribe(eventName, callback) {  if (!this.eventMap.has(eventName)) {  this.eventMap.set(eventName, new Map());  }  const callbackMap = this.eventMap.get(eventName);  callbackMap.set(callback, (callbackMap.get(callback) ?? 0) + 1);   return {  release() {  callbackMap.get(callback) === 1  ? callbackMap.delete(callback)  : callbackMap.set(callback, callbackMap.get(callback) - 1);  }  };  }   emit(eventName, ...args) {  const callbackMap = this.eventMap.get(eventName);   if (callbackMap) {  for (let [callback, count] of callbackMap.entries()) {  for (let i = 0; i < count; i++) {  callback.apply(this, args);  }  }  }  } }  *// Example usage* const emitter = new EventEmitter();  const subscription1 = emitter.subscribe('event1', (arg1, arg2) => {  console.log(`Event1: ${arg1} ${arg2}`); }); const subscription2 = emitter.subscribe('event2', () => {  console.log('Event2 triggered'); }); const subscription3 = emitter.subscribe('event1', (arg1, arg2) => {  console.log(`Another Event1: ${arg1} ${arg2}`); });  emitter.emit('event1', 1, 2); *// Output:* *// Event1: 1 2* *// Another Event1: 1 2*  subscription3.release();  emitter.emit('event1', 4, 5); *// Output:* *// Event1: 4 5*  emitter.emit('event2'); *// Output:* *// Event2 triggered* |
| --- |

### **11. "Finding Corresponding Nodes in Identical DOM Trees”**

| */\*\*  \* @param {HTMLElement} sourceRoot  \* @param {HTMLElement} targetRoot - sourceRoot and targetRoot are clones of each other  \* @param {HTMLElement} sourceNode  \* @returns {HTMLElement|null} - Corresponding node in targetRoot or null if not found  \*/* const findCorrespondingNode = (sourceRoot, targetRoot, sourceNode) => {  if (sourceRoot === sourceNode) {  return targetRoot;  }   if (sourceRoot.childElementCount) {  for (let i = 0; i < sourceRoot.childElementCount; i++) {  const result = findCorrespondingNode(sourceRoot.children[i], targetRoot.children[i], sourceNode);  if (result) {  return result;  }  }  }   return null; };  *// Example usage* const sourceRoot = document.createElement('div'); sourceRoot.innerHTML = `  <div>  <h1 id="title">Hello, World!</h1>  <p>This is a paragraph.</p>  <div>  <span>This is a span.</span>  <ul>  <li>Item 1</li>  <li>Item 2</li>  <li>Item 3</li>  </ul>  </div>  </div>`;  const targetRoot = sourceRoot.cloneNode(true); const sourceNode = sourceRoot.querySelector('span'); const correspondingNode = findCorrespondingNode(sourceRoot, targetRoot, sourceNode);  console.log(correspondingNode); *// Returns the corresponding <span> element in targetRoot* |
| --- |

### **12. Curried Function for Summing Numbers**

| function createSum(num) {  const sumInner = (num2) => {  if (!num2) {  return num;  } else {  return createSum(num + num2);  }  };   return sumInner; }  const sum1 = createSum(1); console.log(sum1(2)(3)); *// Output: 6* |
| --- |

### **12. Reorder array with new indexes**

| function reorderArray(originalArray, newOrder) {  const reorderedArray = [];  for (let i = 0; i < originalArray.length; i++) {  const newIndex = newOrder[i];  reorderedArray[newIndex] = originalArray[i];  }  return reorderedArray; }  const fruit = ['Apple', 'Banana', 'Orange', 'Mango']; const newOrder = [2, 0, 3, 1];  console.log(reorderArray(fruit, newOrder)); |
| --- |

### **13. Custom Implementation of Object.assign()**

| function assignObject(target, ...sources) {  if (target === null || target === undefined) {  throw new Error('Target object cannot be null or undefined');  }   if (typeof target !== 'object') {  target = Object(target);  }   sources.forEach(source => {  if (source !== null && source !== undefined) {  Object.defineProperties(target, Object.getOwnPropertyDescriptors(source));  }  });   return target; }  *// Example usage* const person = {  name: 'John',  age: 30 };  const details = {  occupation: 'Engineer',  location: 'New York' };  const mergedObject = assignObject({}, person, details);  console.log(mergedObject); |
| --- |

### **14. Custom Implementation of clearAllTimeout()**

| const timeouts = [];  function clearAllTimeouts() {  timeouts.forEach((timeout) => {  clearTimeout(timeout);  });  timeouts.length = 0; }  const timeout1 = setTimeout(() => {  console.log('First timeout executed.'); }, 1000); timeouts.push(timeout1);  const timeout2 = setTimeout(() => {  console.log('Second timeout executed.'); }, 2000); timeouts.push(timeout2);  const timeout3 = setTimeout(() => {  console.log('Third timeout executed.'); }, 3000); timeouts.push(timeout3);  console.log('Timeouts set.');  setTimeout(() => {  clearAllTimeouts();  console.log('All timeouts cleared.'); }, 2500); |
| --- |

### **15. implemet Promise.all()**

| function promiseAll(iterable) {  const results = [];  let completedCount = 0;   return new Promise((resolve, reject) => {  if (iterable.length === 0) {  resolve(results);  return;  }   iterable.forEach((item, index) => {  const promise = Promise.resolve(item);    promise.then(result => {  results[index] = result;  completedCount++;   if (completedCount === iterable.length) {  resolve(results);  }  })  .catch(error => {  reject(error);  });  });  }); }  const promise1 = new Promise(resolve => setTimeout(() => resolve('Hello'), 2000)); const promise2 = new Promise(resolve => setTimeout(() => resolve('World'), 1000)); const promise3 = new Promise(resolve => setTimeout(() => resolve('!'), 1500));  promiseAll([promise1, promise2, promise3])  .then(results => {  console.log(results.join(' ')); *// Output: Hello World !*  })  .catch(error => {  console.error(error);  }); |
| --- |

### **16. implement Promise.allSettled()**

| function allSettled(promises) {  let results = [];  let completedCount = 0;   return new Promise((resolve) => {  if (promises.length === 0) {  resolve(results);  return;  }   promises.forEach((promise, index) => {  Promise.resolve(promise)  .then(value => {  results[index] = { status: 'fulfilled', value };  completedCount++;   if (completedCount === promises.length) {  resolve(results);  }  })  .catch(reason => {  results[index] = { status: 'rejected', reason };  completedCount++;   if (completedCount === promises.length) {  resolve(results);  }  });  });  }); }  *// Example usage* const promise1 = Promise.resolve(50); const promise2 = new Promise((resolve, reject) =>  setTimeout(() => reject('Error occurred'), 1000) ); const promises = [promise1, promise2];  allSettled(promises)  .then(results => {  results.forEach(result => {  console.log(result.status, result.value);  });  })  .catch(error => {  console.error(error);  }); |
| --- |

### **17. implement Promise.race()**

| Promise.race() is a method that takes an iterable of promises as input and returns a new promise that is settled (either fulfilled or rejected) as soon as any of the promises in the iterable settles.  function racePromises(promises) {  return new Promise((resolve, reject) => {  promises.forEach((promise) => {  Promise.resolve(promise)  .then(value => {  resolve(value);  })  .catch(error => {  reject(error);  });  });  }); } |
| --- |

### **18. implement Promise.any()**

| Promise.any() is a method introduced in ES2021 (also known as ES12) that takes an iterable of Promises and returns a new Promise. This new Promise resolves as soon as any of the input Promises in the iterable resolves, with the value of the first resolved Promise. If all input Promises are rejected, then the returned Promise is rejected with an AggregateError, which contains an array of rejection values.  function any(promises) {   let errors = [];    return new Promise((resolve, reject) => {  for(let [idx, promise] of promises.entries()) {  promise  .then((res) => resolve(res))  .catch(e => {  errors[idx] = e;  if(errors.length === promises.length) {  reject(new AggregateError(  'No Promise in Promise.any was resolved',   errors  ))  }  })  }  }) } |
| --- |

### **19. calculate tree height**

| <!DOCTYPE html> <html> <head>  <title>DOM Tree Height Calculation</title> </head> <body>  <div id="container">  <h1>Title</h1>  <div>  <p>Paragraph 1</p>  <p>Paragraph 2</p>  <div>  <span>Span 1</span>  <span>Span 2</span>  </div>  </div>  <div>  <ul>  <li>Item 1</li>  <li>Item 2</li>  <li>Item 3</li>  </ul>  </div>  </div>   <script>  function calculateDOMTreeHeight(root) {  if (!root || !root.children || root.children.length === 0) {  return 0;  }   let maxHeight = 0;   for (let i = 0; i < root.children.length; i++) {  const childHeight = calculateDOMTreeHeight(root.children[i]);  maxHeight = Math.max(maxHeight, childHeight);  }   return 1 + maxHeight;  }   *// Example usage*  const htmlElement = document.getElementById('container');  const height = calculateDOMTreeHeight(htmlElement);  console.log(`The height of the DOM tree is ${height}`);  </script> </body> </html> |
| --- |

### **20. create your own Function.prototype.call**

| Function.prototype.mycall = function(thisArg, ...args) {  let func = this;  let identifier = Symbol('func');  let ctx = thisArg ?? window;  ctx = Object(ctx);   ctx[identifier] = func;   const response = ctx[identifier](...args);   delete ctx[identifier];   return response } |
| --- |

### **21. Given a DOM tree, please return all the tag names it has.**

| function getTags(tree, tags = new Set()) {  tags.add(tree.tagName.toLowerCase());   if (tree.childElementCount) {  for (let child of tree.children) {  getTags(child, tags);  }  }   return Array.from(tags); }  *// Example usage* const htmlElement = document.getElementById('container'); const tags = getTags(htmlElement); console.log('Tags:', tags); |
| --- |

### **22. implement custom *.get(*) as in jquery**

| function get(source, path, defaultValue = undefined) {  if (!Array.isArray(path)) {  path = path.replaceAll('[', '.')  .replaceAll(']', '').split('.');  }   if(!path.length) return defaultValue   let current = source;  for (let accessor of path) {  if (accessor in current) {  current = current[accessor]  } else {  return defaultValue;  }  }  return current }  const data = {  user: {  name: 'John Doe',  age: 30,  address: {  city: 'New York',  country: 'USA'  }  },  products: [  { id: 1, name: 'Product 1', price: 10 },  { id: 2, name: 'Product 2', price: 20 },  { id: 3, name: 'Product 3', price: 30 }  ] };  console.log(get(data, 'user.name')); *// "John Doe"* console.log(get(data, 'user.address.city')); *// "New York"* console.log(get(data, 'products[1].name')); *// "Product 2"* console.log(get(data, 'products[2].price')); *// 30* console.log(get(data, 'products[3].name', 'Unknown')); *// "Unknown"* console.log(get(data, 'user.gender', 'Not specified')); *// "Not specified"* |
| --- |

### **23. Implement Negative Array Indexing in JavaScript**

| function createWrappedArray(arr) {  return new Proxy(arr, {  get: (target, prop) => {  if (prop === Symbol.iterator) {  *// Preserve iterator functionality*  return target[prop].bind(target);  }   if (!isNaN(Number(prop))) {  let idx = parseInt(prop, 10);   *// Check if index is within range*  if (Math.abs(idx) > target.length) {  return undefined;  }   *// Handle negative indexing*  return target[idx < 0 ? target.length + idx : idx];  }   *// Preserve other property access*  return target[prop].bind(target);  },   set: (target, prop, value) => {  if (Number(prop)) {  let idx = parseInt(prop, 10);   *// Handle negative indexing*  if (idx < 0) {  idx = target.length + idx;  }   *// Check if index is valid*  if (idx < 0) {  throw new Error('Invalid index');  } else {  target[idx] = value;  return true;  }  }   *// Preserve other property assignment*  target[prop] = value;  return true;  }  }); }  *// Example usage* const arr = [1, 2, 3, 4, 5]; const wrappedArray = createWrappedArray(arr);  console.log(wrappedArray[0]); *// 1* console.log(wrappedArray[-1]); *// 5 (negative indexing)* console.log(wrappedArray[6]); *// undefined (out of range)* console.log([...wrappedArray]); *// [1, 2, 3, 4, 5]*  wrappedArray[-2] = 6; *// Assign value using negative index* console.log(wrappedArray[-2]); *// 6* console.log(wrappedArray[4]); *// 6* |
| --- |

### **24. Find Next Right Sibling in a DOM Tree**

| function findNextRightSibling(root, target) {  let queue = [root];   while (queue.length) {  let current = queue.shift();   if (current === target) {  *// If target found, return the next element in the queue (next right sibling)*  return queue.length ? queue.shift() : null;  }   if (current.childElementCount) {  *// Add children of the current element to the queue*  queue.push(...current.children);  }  }   *// If target not found or no next right sibling, return null*  return null; }  *// Example usage* const rootElement = document.getElementById('root'); const targetElement = document.getElementById('target');  const nextSibling = findNextRightSibling(rootElement, targetElement);  if (nextSibling) {  console.log('Next right sibling:', nextSibling); } else {  console.log('No next right sibling found.'); } |
| --- |

### **25. Traverse DOM level by level**

| function traverseLevelByLevel(root) {  if (!root) return [];   let queue = [root];  let visited = [];   while (queue.length) {  let current = queue.shift();  visited.push(current);   if (current.hasChildNodes()) {  for (let child of current.children) {  queue.push(child);  }  }  }   return visited; }  *// Example usage* const rootElement = document.getElementById('root');  const flattenedElements = traverseLevelByLevel(rootElement);  console.log('Flattened elements:', flattenedElements);  <div id="root">  <h2>Level 1</h2>  <div>  <h3>Level 2 - Child 1</h3>  <p>Level 3 - Child 1</p>  <p>Level 3 - Child 2</p>  </div>  <div>  <h3>Level 2 - Child 2</h3>  <p>Level 3 - Child 3</p>  <p>Level 3 - Child 4</p>  </div> </div> |
| --- |

### **26. Binary Tree Serialization and Deserialization implementation**

| function serializeBinaryTree(root) {  if (!root) return '[]';   let serialized = [];  let queue = [root];   while (queue.length) {  let current = queue.shift();   if (current) {  serialized.push(current.value);  queue.push(current.left);  queue.push(current.right);  } else {  serialized.push(null);  }  }   return JSON.stringify(serialized); }  function deserializeBinaryTree(str) {  let serialized = JSON.parse(str);   if (!serialized.length) return null;   let root = new Node(serialized.shift());  let queue = [root];   while (queue.length) {  let current = queue.shift();   let left = serialized.shift();  if (left !== null) {  current.left = new Node(left);  queue.push(current.left);  }   let right = serialized.shift();  if (right !== null) {  current.right = new Node(right);  queue.push(current.right);  }  }   return root; }  *// Example usage* class Node {  constructor(value) {  this.value = value;  this.left = null;  this.right = null;  } }  const root = new Node(1); root.left = new Node(2); root.right = new Node(3); root.left.left = new Node(4); root.left.right = new Node(5);  const serializedTree = serializeBinaryTree(root); console.log('Serialized Tree:', serializedTree);  const deserializedTree = deserializeBinaryTree(serializedTree); console.log('Deserialized Tree:', deserializedTree); |
| --- |

### **27. DOM Virtualization and Rendering Functions**

| const virtualize = (element) => {  if (!element) return null;   let virtualElement = {  type: element.tagName.toLowerCase(),  props: {}  };   *// Attributes*  for (let attr of element.attributes) {  let name = attr.name === 'class' ? 'className' : attr.name;  virtualElement.props[name] = attr.value;  }   let children = [];   for (let node of element.childNodes) {  if (node.nodeType === 3) {  children.push(node.textContent);  } else {  children.push(virtualize(node));  }  }   virtualElement.props.children = children.length === 1 ? children[0] : children;   return virtualElement; };  */\*\*  \* @param {object} virtualElement - Valid object literal presentation of a DOM element  \* @return {HTMLElement} - Rendered DOM element  \*/* function renderDOM(virtualElement) {  if (typeof virtualElement === 'string') {  return document.createTextNode(virtualElement);  }   const { type, props: { children, ...attrs } } = virtualElement;  const element = document.createElement(type);   for (let [attr, value] of Object.entries(attrs)) {  element[attr] = value;  }   const childrenArr = Array.isArray(children) ? children : [children];   for (let child of childrenArr) {  element.append(renderDOM(child));  }   return element; }  <!DOCTYPE html> <html>  <head>  <title>Virtualize DOM Elements Example</title>  </head>  <body>  <h1 class="title">Welcome to Virtualize DOM Elements Example</h1>  <p>This is a demonstration of virtualizing and rendering DOM elements using JavaScript.</p>  <ul>  <li><a href="<https://www.example.com>">Example Website</a></li>  <li><a href="<https://www.google.com>">Google</a></li>  <li><a href="<https://www.github.com>">GitHub</a></li>  </ul>  </body> </html>  const container = document.documentElement; const virtualElement = virtualize(container); console.log(virtualElement);  {  "type": "html",  "props": {  "children": [  {  "type": "head",  "props": {  "children": [  {  "type": "title",  "props": {  "children": "Virtualize DOM Elements Example"  }  }  ]  }  },  {  "type": "body",  "props": {  "children": [  {  "type": "h1",  "props": {  "className": "title",  "children": "Welcome to Virtualize DOM Elements Example"  }  },  {  "type": "p",  "props": {  "children": "This is a demonstration of virtualizing and rendering DOM elements using JavaScript."  }  },  {  "type": "ul",  "props": {  "children": [  {  "type": "li",  "props": {  "children": {  "type": "a",  "props": {  "href": "<https://www.example.com>",  "children": "Example Website"  }  }  }  },  {  "type": "li",  "props": {  "children": {  "type": "a",  "props": {  "href": "<https://www.google.com>",  "children": "Google"  }  }  }  },  {  "type": "li",  "props": {  "children": {  "type": "a",  "props": {  "href": "<https://www.github.com>",  "children": "GitHub"  }  }  }  }  ]  }  }  ]  }  }  ]  } } |
| --- |

### **28. FIND Previous Left Sibling in DOM Tree**

| function findPreviousLeftSiblingInDOMTree(root, target) {  if (!root) return null;  if (target.previousElementSibling) {  return target.previousElementSibling;  }   let unvisitedNodes = [root];   while (unvisitedNodes.length) {  let current = unvisitedNodes.pop();   if (current === target) {  return unvisitedNodes.pop() ?? null;  }   unvisitedNodes.unshift(...current.children);  } }  <!DOCTYPE html> <html>  <body>  <div id="container">  <div>Element 1</div>  <div>Element 2</div>  <div id="target">Element 3</div>  <div>Element 4</div>  <div>Element 5</div>  </div>  </body> </html>  const root = document.getElementById("container"); const targetElement = document.getElementById("target"); const previousLeftSibling = findPreviousLeftSiblingInDOMTree(root, targetElement);  if (previousLeftSibling) {  console.log(previousLeftSibling.textContent); *// Output: Element 2* } else {  console.log("No previous left sibling found."); } |
| --- |

### **29. Remove Circular Reference in object**

| function promisifyFunction(func) {  return function(...args) {  return new Promise((resolve, reject) => {  const callback = (error, data) => {  if (error) {  reject(error);  } else {  resolve(data);  }  }  func.call(this, ...args, callback);  });  }; }  function readFile(filePath, callback) {  *// Simulating asynchronous file reading*  setTimeout(() => {  if (filePath === 'example.txt') {  callback(null, 'File content');  } else {  callback(new Error('File not found'));  }  }, 1000); }  const readFileAsync = promisifyFunction(readFile);  readFileAsync('example.txt')  .then(content => {  console.log(content); *// Output: File content*  })  .catch(error => {  console.error(error); *// Output: Error: File not found*  }); |
| --- |

### **30) Remove Circular Reference in object**

| function removeCircularReferences(obj) {  const visitedObjects = new WeakSet([obj]);   (function iterateObject(obj) {  for (let key in obj) {  if (obj.hasOwnProperty(key)) {  if (typeof obj[key] === 'object') {  if (visitedObjects.has(obj[key])) {  delete obj[key];  } else {  visitedObjects.add(obj[key]);  iterateObject(obj[key]);  }  }  }  }  })(obj); }  function removeCircularReferences(obj) {  const visitedObjects = new WeakSet([obj]);   (function iterateObject(obj) {  for (let key in obj) {  if (obj.hasOwnProperty(key)) {  if (typeof obj[key] === 'object') {  if (visitedObjects.has(obj[key])) {  delete obj[key];  } else {  visitedObjects.add(obj[key]);  iterateObject(obj[key]);  }  }  }  }  })(obj); } |
| --- |

### **31)Object Path Value Extractor**

| Object.prototype.findPropertyValueByPath = function(path) {  const accessors = path.split('.');  let current = this;    for (let key of accessors) {  if (current.hasOwnProperty(key)) {  current = current[key];  } else {  return;  }  }    return current; };  const user = {  name: {  first: 'John',  last: 'Doe'  },  age: 30,  address: {  city: 'New York',  country: 'USA'  } };  console.log(user.findPropertyValueByPath('name.first')); *// Output: John* console.log(user.findPropertyValueByPath('address.city')); *// Output: New York* console.log(user.findPropertyValueByPath('age')); *// Output: 30* console.log(user.findPropertyValueByPath('name.last')); *// Output: Doe* console.log(user.findPropertyValueByPath('address.country')); *// Output: USA* console.log(user.findPropertyValueByPath('address.state')); *// Output: undefined* |
| --- |

### **32)setTimeout, clearTimeout callback**



| function createSetTimeout() {  var timerId = 0;  var timerMap = {};   function setTimeoutPoly(callback, delay) {  var id = timerId++;  timerMap[id] = true;  var start = Date.now();   function triggerCallback() {  if (!timerMap[id]) return;  if (Date.now() > start + delay) {  callback();  } else {  requestIdleCallback(triggerCallback);   *//setTimeout(triggerCallback, 0); // Use native setTimeout to avoid blocking*  }  }  requestIdleCallback(triggerCallback);  return id;  }   function clearTimeoutPoly(id) {  delete timerMap[id];  }   return { setTimeoutPoly, clearTimeoutPoly }; }  var { setTimeoutPoly, clearTimeoutPoly } = createSetTimeout();  console.log("Start"); var myId = setTimeoutPoly(function() {  console.log("Welcome to jscafe"); }, 1000); |
| --- |

### **33)GroupBy polyfill**

| function groupBy(arr, groupByKey) {  const result = {};   for (let item of arr) {  const key = typeof groupByKey === "function" ? groupByKey(item) : item[groupByKey];   if (!result.hasOwnProperty(key)) {  result[key] = [];  }   result[key].push(item);  }   return result; }    let arr=["sri",'asdfw',"sfkwofinwfw","abcdwe"] let res=groupby(arr,(item)=>item.length) console.log(res,"result")  { "3": [ "sri" ], "5": [ "asdfw" ], "6": [ "abcdwe" ], "11": [ "sfkwofinwfw" ] } |
| --- |

### **34)Merge Objects**

| function merge(...args) {  let target = {};   function helper(obj1, obj2) {  for (let key in obj2) {  if (obj2.hasOwnProperty(key)) {  let value = obj2[key];   if (typeof value == "object") {  obj1[key] = helper(obj1[key] || {}, value);  } else {  obj1[key] = obj2[key];  }  }  }   return obj1;  }   for (let obj of args) {  target = helper(target, obj);  }  return target; }  let obj1={  a:"a",  b:"b",  c:{  d:"d",  e:"e",  f:"f"  } }  let obj2={  g:"g",  h:"h",  a:"c",  c:{  g:"g"  } }   let res=merge(obj1,obj2); console.log(res,"res"){  "a": "c",  "b": "b",  "c": {  "d": "d",  "e": "e",  "f": "f",  "g": "g"  },  "g": "g",  "h": "h" } |
| --- |

### **35)deep copy object (can contain arrays)**

| function deepCopy(obj){   let newObj=Array.isArray(obj)?[]: {}  for(let key in obj){   let value=obj[key]   if(typeof value == "object"){  newObj[key]=deepCopy(value)   }   else{  newObj[key]=obj[key]  }  }    return newObj  }    let data={  name:"srika",  books:{  types:[1,2,2,32],  count:0,  isVisible:false  } } let result=deepCopy(data) console.log(result) |
| --- |

### **36)Araay with event listers, pushEvent, popEvent**

| arr=[]  Array.prototype.listeners={  }    Array.prototype.popWithEvent=function (){  let arr=this;  let ele=arr.pop()  this.triggerEvent("pop",ele);      } Array.prototype.pushWithEvent=function (...ele){   let arr=this;  arr.push(...ele)  this.triggerEvent("push",ele);      }  Array.prototype.addEventListener=function (eventNam,callback){   if(!this.listeners[eventNam]){  this.listeners[eventNam]=[]  }  this.listeners[eventNam].push(callback)       }  Array.prototype.triggerEvent=function(eventName,...values){   if(this.listeners[eventName]){    let callbacks=this.listeners[eventName];  callbacks.forEach(fn=>fn.apply(this,values))   }  }  arr.addEventListener("push",(ele)=>{  console.log(ele,'PUshed') })  arr.addEventListener("pop",(ele)=>{  console.log(ele,'poped') })   arr.pushWithEvent(2,4) arr.pushWithEvent(4,3) arr.pushWithEvent(8)   arr.popWithEvent() |
| --- |

### **37)get and set path in obj**

| function get(obj,path=""){     let helper=(obj,pathArr)=>{   let[current,...next]=pathArr   if(next.length==0){  return obj[current]  }  else{   return helper(obj[current],next)   }   }   let pathArr=[]   if(typeof path=="string")  pathArr=path.replaceAll("[", ".").replaceAll("]","").split(".")  else{  pathArr=path  }   *// console.log(pathArr)*   let res=helper(obj,pathArr)  return res;   *// console.log(res,"result")* }   function set(obj,path,value){     let helper=(obj,pathArr,value)=>{   let[current,...next]=pathArr   if (next.length === 0) {    obj[current] = value;  }  else{    if(!obj[current]){  obj[current]=isNaN(next[0])?{}:[]  }    helper(obj[current],next,value)   }   }    let pathArr=[]   if(typeof path=="string")  pathArr=path.replaceAll("[", ".").replaceAll("]","").split(".")  else{  pathArr=path  }   console.log(pathArr)   let res=helper(obj,pathArr,value)  return res;   }  *// Initial object for testing* let obj = {  a: {  b: {  c: 1  }  },  d: [1, 2, 3],  e: {  f: [  { g: 'test' }  ]  } };  *// Test cases for get function* console.log(get(obj, 'a.b.c'));  *// Expected output: 1*  console.log(get(obj, 'd[1]'));  *// Expected output: 2*  console.log(get(obj, 'e.f[0].g'));  *// Expected output: 'test'*  console.log(get(obj, 'x.y.z'));  *// Expected output: undefined (non-existent path)*  *// Test cases for set function* set(obj, 'a.b.c', 42); console.log(obj.a.b.c);  *// Expected output: 42*  set(obj, 'd[1]', 99); console.log(obj.d[1]);  *// Expected output: 99*  set(obj, 'e.f[0].g', 'updated'); console.log(obj.e.f[0].g);  *// Expected output: 'updated'*  set(obj, 'x.y.z', 'new value'); console.log(obj.x.y.z);  *// Expected output: 'new value'* |
| --- |

### **38)implement Delay**

| function deplay(time){  return new Promise((res,rej)=>setTimeout(res,time)); }  async function getdata(){  console.log("arER")  await deplay(2000);  console.log("hi") }  getdata() |
| --- |

### **39)Cache any function return value**

| function memoise(fn){   let cache={}  return (...args)=>{   let key=JSON.stringify(args)   let entry=cache[key]     if(!entry){  let res=fn(...args)  cache[key]=res  }   return cache[key]   }  }  function add(a,b,c){  console.log("called")  return a+b+c; }  function factorial(n){    console.log("called")  let res=1  for(let i=1;i<=n;i++){  res=res\*i   }  return res; }  let memoiseAdd=memoise(add) memoiseAdd(1,2,3) memoiseAdd(1,2,3) let memoisefactorial=memoise(factorial)  console.log(memoisefactorial(5)) console.log(memoisefactorial(6)) console.log(memoisefactorial(5)) |
| --- |

### **40)fetch with cached responce**

| let fetchWithCache=(expireTIme)=>{  let cache={   }   *// console.log(cache,"Cache")*  return async (URL,params)=>{   let key=`${URL}${JSON.stringify(params)}`   let entry=cache[key]  if(!entry || Date.now()>entry.expiry){   try{  let resp= await fetch(URL,{})  let result=await resp.json()  cache[key]={value:result,expiry:Date.now()+expireTIme}   }  catch(ex){  console.log("some error in api call ")  }   }   console.log(key,"key")    return cache[key].value    }   }  let URL="https://jsonplaceholder.typicode.com/todos/1"  let call=fetchWithCache(10000)   call(URL,{}).then(res=>console.log(res))  setTimeout(()=>{  call(URL,{}).then(res=> console.log(res))   },2000)   setTimeout(()=>{  call(URL,{}).then(res=>console.log(res))   },2000) |
| --- |

### **41)Fetch with retry n times**

| let fetchWithRetry = async (url, params, n) => {  try {  return await fetch(url, params);  } catch (err) {  if (n == 0) {  throw err;  }  return await fetchWithRetry(url, params, n - 1);  } };  let URL = `https://jsonplaceholder1.typicode.com/posts`;  async function getData(){  try {  let resp = await fetchWithRetry(URL, {}, 5);  let res = await resp.json();  console.log(res);    } catch (ex) {  console.log(ex);  } } getData(); |
| --- |

### **42)Object.create()**

| if (typeof Object.create !== 'function') { Object.create = function (proto, propertiesObject) {  if (typeof proto !== 'object' && typeof proto !== 'function') {  throw new TypeError('Object prototype may only be an Object or null');  }   *// Create a temporary constructor function*  function F() {}   *// Set the prototype of the temporary constructor to the specified prototype object*  F.prototype = proto;   *// Create a new instance of the temporary constructor*  var obj = new F();   *// Optionally define properties on the new object*  if (typeof propertiesObject === 'object') {  Object.defineProperties(obj, propertiesObject);  }   return obj;  }; }  *// this is how define properties works* const obj = {};  Object.defineProperties(obj, {  name: {  value: 'John',  writable: true,  enumerable: true  },  age: {  value: 30,  writable: false,  enumerable: true  } });  let obj=Object.myCreate({name:"srikt"},{age:{  value:50,  writable:true }}) console.log(obj.\_\_proto\_\_) |
| --- |

### **42)Implement Publish, subscribe pattern in js**

| function pubSub() {  const subscribers = [];   function publish(data) {  subscribers.forEach(subscriber => subscriber(data));  }   function subscribe(fn) {  subscribers.push(fn);  *// Return an unsubscribe function*  return function unsubscribe() {  const index = subscribers.indexOf(fn);  if (index !== -1) {  subscribers.splice(index, 1);  }  };  }   return {  publish,  subscribe,  }; }  *// driver code* const pubSubObj = pubSub();  const unsubscribe1 = pubSubObj.subscribe(data => {  console.log('Subscriber 1: ' + data); }); const unsubscribe2 = pubSubObj.subscribe(data => {  console.log('Subscriber 2: ' + data); });  *// all subscribers will be called with the data on publish* pubSubObj.publish('Value is 10');  *// Unsubscribe the first subscriber* unsubscribe1();  *// Only the second subscriber will be called* pubSubObj.publish('Value is 20'); |
| --- |

### **43)Implement bind polyfill**

| Function.prototype.mybind = function (context, ...args) {  let fn = this;   return function (...newargs) {  fn.apply(context, [...args, ...newargs]);  }; };  function greet(greeting, punctuation) {  console.log(`${greeting}, ${this.name}${punctuation}`); }  const boundGreet = greet.mybind({ name: 'John' }, 'Hello'); |
| --- |

### **44)Deep comparision of object (\_.equal(ob1,obj2) polyfill)**

| function deepComparison(first, second) {  if (first === second) return true;   *// Check if either argument is null or not an object*  if (first === null || second === null || typeof first !== 'object' || typeof second !== 'object') return false;   *// Get property names of both objects*  let firstKeys = Object.keys(first);  let secondKeys = Object.keys(second);   *// Check if the number of properties is different*  if (firstKeys.length !== secondKeys.length) return false;   *// Iterate through the properties and recursively check equality*  for (let key of firstKeys) {  if (!second.hasOwnProperty(key) || !deepComparison(first[key], second[key])) return false;  }   return true; } |
| --- |

### **45)Implement range Iterator**

| function rangeIterator(start, end) {  let current = start;  return {  [Symbol.iterator]: function() {  return this;  },  next() {  if (current <= end) {  return { value: current++, done: false };  } else {  return { done: true };  }  }  }; }  const numbers = rangeIterator(1, 5); for (const number of numbers) {  console.log(number); *// Output: 1, 2, 3, 4, 5* } |
| --- |

### **46)Implement range Iterator**

| if we give number =>it has wait till then handler attached it we give promise=>it has to call inside constructor  // Define the Promise class class MyPromise {  onResolve = undefined;  onReject = undefined;   *// isfulfilled = false;*  *// isrejected = false;*  iscalled = false;   status = "pending";  value = undefined;  error = undefined;   constructor(executor) {   let resolve = (val) => {  this.status = "fulfilled";  this.value = val;    *//need to check this because we need to call this once .then callback added. so*   if (typeof this.onResolve == "function" && !this.iscalled) {  this.onResolve(val);  this.iscalled = true;  }  };    let reject = (val) => {  this.error = val;  this.status = "rejected";   if (typeof this.onReject == "function" && !this.called) {  this.onReject(val);  this.iscalled = true;  }  };   try {  executor(resolve, reject);  } catch (ex) {}  }     then(thenHandler) {  this.onResolve = thenHandler;  if (!this.iscalled && this.status=="fulfilled") {  this.onResolve(this.value);  }  return this;   }   catch(catchHandler) {  this.onReject = catchHandler;  if(!this.iscalled && this.status=="rejected"){   this.onReject(this.error);    }  return this;   }    static resolve(val){  return new MyPromise((resolve,reject)=>{  resolve(val)   })  }    static reject(val){  return new MyPromise((resolve,reject)=>{  reject(this.val)  })  }  }   let p=new MyPromise((resolve,reject)=>{  setTimeout(()=>{   resolve("this is resolved async)  },2000) })  p.then((data)=>console.log(data))  Promise.resolve(34).then(data=>console.log(data)) Promise.reject("it is failaed syncway").catch(error=>console.log(error,"error")) |
| --- |

### **47)Aggregate array of objects on the given keys**

| const endorsements = [ { skill: 'css', user: 'Bill' }, { skill: 'javascript', user: 'Chad' }, { skill: 'javascript', user: 'Bill' }, { skill: 'css', user: 'Sue' }, { skill: 'javascript', user: 'Sue' }, { skill: 'html', user: 'Sue' } ]; console.log(aggregate(endorsements, "user", "skill"));  Given an array of objects and two keys "on" and "who", aggregate the "who" values on the "on" values.   function aggregrate(arr,on,who){   let obj={}   for(let item of arr){   let onValue=item[on]  let whoValue=item[who]    if(!obj[onValue]){    obj[onValue]={  [on]:onValue,  [who]:[]  }    }     obj[onValue][who].push(whoValue)         }  return Object.values(obj) }   const endorsements = [  { skill: 'css', user: 'Bill' },  { skill: 'javascript', user: 'Chad' },  { skill: 'javascript', user: 'Bill' },  { skill: 'css', user: 'Sue' },  { skill: 'javascript', user: 'Sue' },  { skill: 'html', user: 'Sue' }  ]; console.log(aggregrate(endorsements, "skill", "user")) //output  [  {  "skill": "css",  "user": [  "Bill",  "Sue"  ]  },  {  "skill": "javascript",  "user": [  "Chad",  "Bill",  "Sue"  ]  },  {  "skill": "html",  "user": [  "Sue"  ]  } ] |
| --- |

### **48)Currying variation**

| 4. Write a function to make the following code snippet work? console.log(sum(4, 6, 8, 10).value); //output - 28 console.log(sum(4)(6)(8)(10).value); //output - 28 Answer  function sum(...args) {  const ans = args.reduce((a, b) => a + b, 0); *//just to get sum of all the array elements*   const myFunc = (num) => {  return sum(num, ...args);  };   myFunc.value = ans;   return myFunc; }  console.log(sum(4, 6, 8, 10).value); *//output - 28* console.log(sum(4)(6)(8)(10).value); *//output - 28* |
| --- |

### **48)Write a function in Javascript to determine the type of a value**

| const getType = (val) => (val === null ? null : val?.constructor.name);  console.log(getType(42)); console.log(getType("Hello")); console.log(getType(true)); console.log(getType([1, 2, 3])); console.log(getType({ name: "John", age: 25 })); console.log(getType(null)); console.log(getType(undefined)); console.log(getType(function () {})); |
| --- |