Salom! JavaScript mashqlarini o'rganish uchun quyidagi saytlarni tavsiya qilaman:

1. **freeCodeCamp** - [freecodecamp.org](https://www.freecodecamp.org)
   * Bu sayt JavaScript bo'yicha bepul kurslarni taklif qiladi va kod yozish uchun ko'plab amaliy mashqlar mavjud.
2. **Codecademy** - [codecademy.com](https://www.codecademy.com)
   * Bu platforma interaktiv JavaScript kurslarini taqdim etadi va mashqlar bilan amaliyot qilish imkoniyatini beradi.
3. **Exercism** - exercism.org
   * Bu sayt turli dasturlash tillari bo'yicha mashqlarni taqdim etadi va JavaScript bo'yicha ham ko'plab amaliyotlarni taklif etadi.
4. **HackerRank** - hackerrank.com
   * HackerRankda JavaScript bo'yicha ko'plab masalalar va mashqlar mavjud. "10 Days of JavaScript" kursi bilan boshlashingiz mumkin.
5. **LeetCode** - [leetcode.com](https://leetcode.com)
   * LeetCode'da JavaScript bo'yicha ko'plab masalalar va ularni yechish imkoniyatlari mavjud. Bu sayt sizga murakkab masalalarni hal qilishda yordam beradi.
6. **JavaScript.info** - [javascript.info](https://javascript.info)
   * Bu sayt JavaScript bo'yicha batafsil darsliklar va amaliy mashqlarni taklif etadi.
7. **Codewars** - [codewars.com](https://www.codewars.com)
   * Bu platformada turli xil JavaScript masalalarini topishingiz mumkin. Har bir masala uchun har xil murakkablik darajalari mavjud.

Bu saytlar sizga JavaScript bo'yicha bilim va ko'nikmalaringizni oshirishda yordam beradi. Har birini sinab ko'rib, qaysi biri sizga mos kelishini aniqlashingiz mumkin. Omad!

4o

endi shu saytlardagi mashqlarga o'xshash 100 ta mashqlarni yubor men ularni mustaqil yechimini topaman Mashqlar natijasi qanday bo'lish kerakligini ham yoz

ChatGPT

JavaScript bo'yicha mashqlarni mustaqil yechish uchun quyidagi 100 ta mashqni taqdim etaman. Har bir mashq uchun maqsad va natija qanday bo'lishi kerakligini ham kiritaman.

**1-10. Basic Syntax and Operations**

1. **Addition Function**
   * **Masala:** Ikki sonni qo'shadigan funksiya yozing.
   * **Natija:** add(2, 3) → 5
2. **Subtraction Function**
   * **Masala:** Ikki sonni ayiradigan funksiya yozing.
   * **Natija:** subtract(5, 3) → 2
3. **Multiplication Function**
   * **Masala:** Ikki sonni ko'paytiradigan funksiya yozing.
   * **Natija:** multiply(4, 3) → 12
4. **Division Function**
   * **Masala:** Ikki sonni bo'ladigan funksiya yozing.
   * **Natija:** divide(10, 2) → 5
5. **String Concatenation**
   * **Masala:** Ikki satrni birlashtiradigan funksiya yozing.
   * **Natija:** concat("Hello, ", "World!") → "Hello, World!"
6. **Array Length**
   * **Masala:** Massivning uzunligini qaytaradigan funksiya yozing.
   * **Natija:** arrayLength([1, 2, 3, 4]) → 4
7. **String Length**
   * **Masala:** Satr uzunligini qaytaradigan funksiya yozing.
   * **Natija:** stringLength("hello") → 5
8. **Find Maximum**
   * **Masala:** Massivdan eng katta elementni topadigan funksiya yozing.
   * **Natija:** findMax([1, 5, 3, 9, 2]) → 9
9. **Find Minimum**
   * **Masala:** Massivdan eng kichik elementni topadigan funksiya yozing.
   * **Natija:** findMin([1, 5, 3, 9, 2]) → 1
10. **Check Even Number**
    * **Masala:** Sonning juft yoki toq ekanligini tekshiradigan funksiya yozing.
    * **Natija:** isEven(4) → true, isEven(5) → false

**11-20. Intermediate Concepts**

1. **Sum of Array**
   * **Masala:** Massiv elementlarining yig'indisini hisoblaydigan funksiya yozing.
   * **Natija:** sumArray([1, 2, 3, 4]) → 10
2. **Reverse String**
   * **Masala:** Satrni teskari aylantiradigan funksiya yozing.
   * **Natija:** reverseString("hello") → "olleh"
3. **Palindrome Check**
   * **Masala:** Satr palindrom ekanligini tekshiradigan funksiya yozing.
   * **Natija:** isPalindrome("madam") → true, isPalindrome("hello") → false
4. **Factorial Calculation**
   * **Masala:** Sonning faktorialini hisoblaydigan funksiya yozing.
   * **Natija:** factorial(5) → 120
5. **Fibonacci Sequence**
   * **Masala:** Berilgan son qatoriga qarab Fibonacci ketma-ketligini qaytaradigan funksiya yozing.
   * **Natija:** fibonacci(5) → [0, 1, 1, 2, 3]
6. **Find Duplicates**
   * **Masala:** Massivdagi takroriy elementlarni topadigan funksiya yozing.
   * **Natija:** findDuplicates([1, 2, 3, 3, 4, 4, 5]) → [3, 4]
7. **Remove Duplicates**
   * **Masala:** Massivdan takroriy elementlarni olib tashlaydigan funksiya yozing.
   * **Natija:** removeDuplicates([1, 2, 3, 3, 4, 4, 5]) → [1, 2, 3, 4, 5]
8. **Capitalize String**
   * **Masala:** Satrning birinchi harfini katta harf bilan yozadigan funksiya yozing.
   * **Natija:** capitalize("hello") → "Hello"
9. **Flatten Array**
   * **Masala:** Ichma-ich massivni tekislaydigan funksiya yozing.
   * **Natija:** flattenArray([1, [2, 3], [4, [5, 6]]]) → [1, 2, 3, 4, 5, 6]
10. **Find Prime Numbers**
    * **Masala:** Berilgan son oralig'ida barcha tub sonlarni qaytaradigan funksiya yozing.
    * **Natija:** findPrimes(10) → [2, 3, 5, 7]

**21-30. Advanced Functions and Algorithms**

1. **Binary Search**
   * **Masala:** Berilgan sonni tartiblangan massivda qidiradigan binar qidiruv algoritmini yozing.
   * **Natija:** binarySearch([1, 2, 3, 4, 5], 3) → 2
2. **Merge Sort**
   * **Masala:** Massivni birlashgan sort algoritmi yordamida tartiblaydigan funksiya yozing.
   * **Natija:** mergeSort([4, 3, 2, 1]) → [1, 2, 3, 4]
3. **Quick Sort**
   * **Masala:** Massivni tez sort algoritmi yordamida tartiblaydigan funksiya yozing.
   * **Natija:** quickSort([4, 3, 2, 1]) → [1, 2, 3, 4]
4. **Matrix Multiplication**
   * **Masala:** Ikkita matritsani ko'paytiradigan funksiya yozing.
   * **Natija:** multiplyMatrices([[1, 2], [3, 4]], [[5, 6], [7, 8]]) → [[19, 22], [43, 50]]
5. **Depth-First Search (DFS)**
   * **Masala:** Grafda chuqurlik bo'yicha qidirish algoritmini amalga oshiradigan funksiya yozing.
   * **Natija:** dfs(graph, startNode) → [startNode, ...]
6. **Breadth-First Search (BFS)**
   * **Masala:** Grafda kenglik bo'yicha qidirish algoritmini amalga oshiradigan funksiya yozing.
   * **Natija:** bfs(graph, startNode) → [startNode, ...]
7. **Anagram Check**
   * **Masala:** Ikki satr anagram ekanligini tekshiradigan funksiya yozing.
   * **Natija:** isAnagram("listen", "silent") → true, isAnagram("hello", "world") → false
8. **Longest Common Subsequence**
   * **Masala:** Ikki satr orasidagi eng uzun umumiy ketma-ketlikni topadigan funksiya yozing.
   * **Natija:** longestCommonSubsequence("abcde", "ace") → "ace"
9. **Knapsack Problem**
   * **Masala:** Narsalar to'plami va ularning qiymatlari berilgan holda, eng yuqori qiymatni oladigan knapsack muammosini yechadigan funksiya yozing.
   * **Natija:** knapsack(weights, values, capacity) → maxValue
10. **Dijkstra's Algorithm**
    * **Masala:** Grafda eng qisqa yo'lni topadigan Dijkstra algoritmini amalga oshiradigan funksiya yozing.
    * **Natija:** dijkstra(graph, startNode) → { shortestPaths }

**31-40. Object-Oriented Programming (OOP)**

1. **Class Definition**
   * **Masala:** Oddiy class va uni konstruktori bilan yarating.
   * **Natija:** new Person("John", 25) → Person { name: "John", age: 25 }
2. **Inheritance**
   * **Masala:** Class merosxo'rligini amalga oshiring.
   * **Natija:** new Employee("John", 25, "Developer") → Employee { name: "John", age: 25, job: "Developer" }
3. **Method Overriding**
   * **Masala:** Merosxo'r classda methodni qayta yozing.
   * **Natija:** employee.greet() → "Hello, my name is John and I am a Developer."
4. **Static Methods**
   * **Masala:** Classda stat

4o

davom ettir

ChatGPT

Qolgan mashqlar ro'yxatini davom ettiraman:

**31-40. Object-Oriented Programming (OOP) (davomiy)**

1. **Static Methods**
   * **Masala:** Classda statik metod yarating.
   * **Natija:** Person.describe() → "This is a Person class"
2. **Private Properties**
   * **Masala:** Classda shaxsiy xususiyatlar va ularga kirish metodlarini yarating.
   * **Natija:**

javascript

Copy code

const person = new Person("John", 25);

person.getAge(); // 25

person.setAge(30);

person.getAge(); // 30

1. **Getters and Setters**
   * **Masala:** Classda getter va setter metodlarini yarating.
   * **Natija:**

javascript

Copy code

const person = new Person("John", 25);

person.age; // 25

person.age = 30;

person.age; // 30

1. **Polymorphism**
   * **Masala:** Bir nechta classlar uchun polymorphismni amalga oshiring.
   * **Natija:**

javascript

Copy code

const shapes = [new Circle(5), new Square(10)];

shapes.forEach(shape => shape.area()); // [78.5, 100]

1. **Encapsulation**
   * **Masala:** Classlarda ma'lumotlarni inkapsulatsiya qiling.
   * **Natija:**

javascript

Copy code

class BankAccount {

constructor(balance) {

this.\_balance = balance;

}

deposit(amount) {

this.\_balance += amount;

}

withdraw(amount) {

this.\_balance -= amount;

}

get balance() {

return this.\_balance;

}

}

const account = new BankAccount(100);

account.deposit(50);

account.withdraw(30);

account.balance; // 120

1. **Inheritance with Override**
   * **Masala:** Merosxo'rlikni amalga oshiring va metodlarni override qiling.
   * **Natija:**

javascript

Copy code

class Animal {

speak() {

return "Animal speaks";

}

}

class Dog extends Animal {

speak() {

return "Dog barks";

}

}

const dog = new Dog();

dog.speak(); // "Dog barks"

1. **Abstract Class**
   * **Masala:** Abstract class va uning merosxo'rligini yarating.
   * **Natija:**

javascript

Copy code

class Shape {

constructor() {

if (this.constructor === Shape) {

throw new Error("Abstract class cannot be instantiated");

}

}

area() {

throw new Error("Abstract method");

}

}

class Circle extends Shape {

constructor(radius) {

super();

this.radius = radius;

}

area() {

return Math.PI \* this.radius \* this.radius;

}

}

const circle = new Circle(5);

circle.area(); // 78.5

**41-50. Functional Programming**

1. **Map Function**
   * **Masala:** Massivdagi har bir elementga funksiyani qo'llaydigan map funksiyasini yozing.
   * **Natija:** map([1, 2, 3], x => x \* 2) → [2, 4, 6]
2. **Filter Function**
   * **Masala:** Massivdagi elementlarni berilgan shart bo'yicha filtrlovchi funksiya yozing.
   * **Natija:** filter([1, 2, 3, 4], x => x % 2 === 0) → [2, 4]
3. **Reduce Function**
   * **Masala:** Massiv elementlarini bitta qiymatga kamaytiradigan reduce funksiyasini yozing.
   * **Natija:** reduce([1, 2, 3, 4], (acc, x) => acc + x, 0) → 10
4. **Currying Function**
   * **Masala:** Funksiyani currying qilishni amalga oshiring.
   * **Natija:**

javascript

Copy code

const add = x => y => x + y;

const add5 = add(5);

add5(10); // 15

1. **Compose Function**
   * **Masala:** Bir nechta funksiyalarni birlashtiradigan compose funksiyasini yozing.
   * **Natija:**

javascript

Copy code

const add = x => x + 1;

const multiply = x => x \* 2;

const compose = (f, g) => x => f(g(x));

const addThenMultiply = compose(multiply, add);

addThenMultiply(5); // 12

1. **Partial Application**
   * **Masala:** Funksiyani qisman qo'llashni amalga oshiring.
   * **Natija:**

javascript

Copy code

const add = (a, b, c) => a + b + c;

const partialAdd = add.bind(null, 1, 2);

partialAdd(3); // 6

1. **Memoization**
   * **Masala:** Funksiyani memoizatsiya qiling.
   * **Natija:**

javascript

Copy code

const memoize = fn => {

const cache = {};

return (...args) => {

const key = JSON.stringify(args);

if (!cache[key]) {

cache[key] = fn(...args);

}

return cache[key];

};

};

const add = (a, b) => a + b;

const memoizedAdd = memoize(add);

memoizedAdd(1, 2); // 3 (calculated)

memoizedAdd(1, 2); // 3 (cached)

1. **Flatten Deep Array**
   * **Masala:** Ichma-ich massivni tekislaydigan funksiyani yozing.
   * **Natija:**

javascript

Copy code

const flatten = arr => arr.reduce((acc, val) => acc.concat(Array.isArray(val) ? flatten(val) : val), []);

flatten([1, [2, [3, [4, 5]]]]); // [1, 2, 3, 4, 5]

1. **Group By Function**
   * **Masala:** Massiv elementlarini berilgan funksiya bo'yicha guruhlaydigan funksiyani yozing.
   * **Natija:**

javascript

Copy code

const groupBy = (arr, fn) => arr.reduce((acc, val) => {

const key = fn(val);

if (!acc[key]) acc[key] = [];

acc[key].push(val);

return acc;

}, {});

groupBy([6.1, 4.2, 6.3], Math.floor); // { 6: [6.1, 6.3], 4: [4.2] }

1. **Zip Function**
   * **Masala:** Ikki massivni zip qiling.
   * **Natija:**

javascript

Copy code

const zip = (arr1, arr2) => arr1.map((val, i) => [val, arr2[i]]);

zip([1, 2, 3], ['a', 'b', 'c']); // [[1, 'a'], [2, 'b'], [3, 'c']]

**51-60. DOM Manipulation**

1. **Create Element**
   * **Masala:** DOMda yangi element yarating va sahifaga qo'shing.
   * **Natija:**

javascript

Copy code

const createElement = (tag, content) => {

const el = document.createElement(tag);

el.textContent = content;

document.body.appendChild(el);

};

createElement('p', 'Hello World'); // <p>Hello World</p> added to body

1. **Remove Element**
   * **Masala:** DOMdan elementni olib tashlang.
   * **Natija:**

javascript

Copy code

const removeElement = selector => {

const el = document.querySelector(selector);

if (el) el.remove();

};

removeElement('p'); // <p>Hello World</p> removed from body

1. **Toggle Class**
   * **Masala:** DOM elementiga klass qo'shish yoki olib tashlashni amalga oshiring.
   * **Natija:**

javascript

Copy code

const toggleClass = (selector, className) => {

const el = document.querySelector(selector);

if (el) el.classList.toggle(className);

};

toggleClass('p', 'highlight'); // <p> tag's class "highlight" toggled

1. **Set Attribute**
   * **Masala:** DOM elementiga atribut qo'shish yoki o'zgartirish.
   * **Natija:**

javascript

Copy code

const setAttribute = (selector, attr, value) => {

const el = document.querySelector(selector);

if (el) el.setAttribute(attr, value);

};

setAttribute('p', 'id', 'para1'); // <p id="para1">Hello World</p>

4o

davom ettir

ChatGPT

**51-60. DOM Manipulation (davomiy)**

1. **Get Attribute**
   * **Masala:** DOM elementining atribut qiymatini qaytarish.
   * **Natija:**

javascript

Copy code

const getAttribute = (selector, attr) => {

const el = document.querySelector(selector);

return el ? el.getAttribute(attr) : null;

};

getAttribute('p', 'id'); // "para1"

1. **Event Listener**
   * **Masala:** DOM elementiga voqea tinglovchisini qo'shish.
   * **Natija:**

javascript

Copy code

const addEventListener = (selector, event, handler) => {

const el = document.querySelector(selector);

if (el) el.addEventListener(event, handler);

};

addEventListener('p', 'click', () => alert('Paragraph clicked!')); // alerts on click

1. **Remove Event Listener**
   * **Masala:** DOM elementidan voqea tinglovchisini olib tashlash.
   * **Natija:**

javascript

Copy code

const removeEventListener = (selector, event, handler) => {

const el = document.querySelector(selector);

if (el) el.removeEventListener(event, handler);

};

const handler = () => alert('Paragraph clicked!');

addEventListener('p', 'click', handler);

removeEventListener('p', 'click', handler); // removes click alert

1. **Append Child**
   * **Masala:** DOM elementiga farzand element qo'shish.
   * **Natija:**

javascript

Copy code

const appendChild = (parentSelector, childTag, childContent) => {

const parent = document.querySelector(parentSelector);

if (parent) {

const child = document.createElement(childTag);

child.textContent = childContent;

parent.appendChild(child);

}

};

appendChild('body', 'div', 'New Div'); // <div>New Div</div> added to body

1. **Replace Child**
   * **Masala:** DOM elementidagi farzand elementni yangisiga almashtirish.
   * **Natija:**

javascript

Copy code

const replaceChild = (parentSelector, newChildTag, newChildContent) => {

const parent = document.querySelector(parentSelector);

if (parent && parent.firstChild) {

const newChild = document.createElement(newChildTag);

newChild.textContent = newChildContent;

parent.replaceChild(newChild, parent.firstChild);

}

};

replaceChild('body', 'div', 'Replaced Div'); // replaces first child with <div>Replaced Div</div>

1. **Clone Node**
   * **Masala:** DOM elementini klonlash.
   * **Natija:**

javascript

Copy code

const cloneNode = (selector, deep = false) => {

const el = document.querySelector(selector);

return el ? el.cloneNode(deep) : null;

};

const clonedNode = cloneNode('p', true);

document.body.appendChild(clonedNode); // clone of <p> element added to body

**61-70. Async Programming**

1. **Simple Callback**
   * **Masala:** Oddiy callback funksiyasini amalga oshiring.
   * **Natija:**

javascript

Copy code

const doSomething = (callback) => {

setTimeout(() => {

callback('Done');

}, 1000);

};

doSomething(result => console.log(result)); // "Done" logged after 1 second

1. **Promise**
   * **Masala:** Oddiy Promise yaratish va ishlatish.
   * **Natija:**

javascript

Copy code

const doSomething = () => {

return new Promise((resolve, reject) => {

setTimeout(() => {

resolve('Done');

}, 1000);

});

};

doSomething().then(result => console.log(result)); // "Done" logged after 1 second

1. **Async/Await**
   * **Masala:** Async/Await yordamida asinxron funksiyani amalga oshirish.
   * **Natija:**

javascript

Copy code

const doSomethingAsync = async () => {

const result = await doSomething();

console.log(result);

};

doSomethingAsync(); // "Done" logged after 1 second

1. **Fetch API**
   * **Masala:** Fetch API yordamida ma'lumot olish.
   * **Natija:**

javascript

Copy code

const fetchData = async (url) => {

const response = await fetch(url);

const data = await response.json();

return data;

};

fetchData('https://jsonplaceholder.typicode.com/todos/1').then(data => console.log(data));

// logs todo item from the API

1. **Parallel Promises**
   * **Masala:** Bir nechta promise'larni parallel bajarish.
   * **Natija:**

javascript

Copy code

const promise1 = new Promise(resolve => setTimeout(() => resolve('One'), 1000));

const promise2 = new Promise(resolve => setTimeout(() => resolve('Two'), 500));

Promise.all([promise1, promise2]).then(results => console.log(results)); // ["One", "Two"] after 1 second

1. **Promise.race**
   * **Masala:** Birinchi bajarilgan promise'ni qaytarish.
   * **Natija:**

javascript

Copy code

const promise1 = new Promise(resolve => setTimeout(() => resolve('One'), 1000));

const promise2 = new Promise(resolve => setTimeout(() => resolve('Two'), 500));

Promise.race([promise1, promise2]).then(result => console.log(result)); // "Two" after 500ms

1. **Chained Promises**
   * **Masala:** Promise zanjirini amalga oshirish.
   * **Natija:**

javascript

Copy code

const doSomething = () => {

return new Promise((resolve) => {

setTimeout(() => {

resolve('First');

}, 1000);

});

};

doSomething()

.then(result => {

console.log(result); // "First"

return 'Second';

})

.then(result => console.log(result)); // "Second"

1. **Error Handling with Promises**
   * **Masala:** Promise'da xatolikni ushlash.
   * **Natija:**

javascript

Copy code

const doSomething = () => {

return new Promise((resolve, reject) => {

setTimeout(() => {

reject('Error');

}, 1000);

});

};

doSomething()

.catch(error => console.log(error)); // "Error" after 1 second

1. **Retry Promise**
   * **Masala:** Xato bo'lganda promise'ni qayta urinish.
   * **Natija:**

javascript

Copy code

const doSomething = () => {

return new Promise((resolve, reject) => {

setTimeout(() => {

Math.random() > 0.5 ? resolve('Success') : reject('Failure');

}, 1000);

});

};

const retry = (fn, retries) => fn().catch(err => retries > 1 ? retry(fn, retries - 1) : Promise.reject(err));

retry(doSomething, 3).then(result => console.log(result)).catch(error => console.log(error));

// "Success" or "Failure" after 3 attempts

1. **Debounce Function**
   * **Masala:** Debounce funksiyasini amalga oshirish.
   * **Natija:**

javascript

Copy code

const debounce = (func, wait) => {

let timeout;

return (...args) => {

clearTimeout(timeout);

timeout = setTimeout(() => func.apply(this, args), wait);

};

};

const log = () => console.log('Hello');

const debouncedLog = debounce(log, 1000);

window.addEventListener('resize', debouncedLog); // "Hello" logged after 1 second of no resize events

**71-80. Data Structures**

1. **Stack Implementation**
   * **Masala:** JavaScriptda stackni amalga oshiring.
   * **Natija:**

javascript

Copy code

class Stack {

constructor() {

this.items = [];

}

push(element) {

this.items.push(element);

}

pop() {

return this.items.pop();

}

peek() {

return this.items[this.items.length - 1];

}

isEmpty() {

return this.items.length === 0;

}

size() {

return this.items.length;

}

clear() {

this.items = [];

}

}

const stack = new Stack();

stack.push(1);

stack.push(2);

stack.pop(); // 2

stack.peek(); // 1

1. **Queue Implementation**
   * **Masala:** JavaScriptda queue'ni amalga oshiring.
   * **Natija:**

javascript

Copy code

class Queue {

constructor() {

this.items = [];

}

enqueue(element) {

this.items.push(element);

}

dequeue() {

return this.items.shift();

}

front() {

return this.items[0];

}

isEmpty() {

return this.items.length === 0;

}

size() {

return this.items.length;

}

clear() {

this.items = [];

}

}

const queue = new Queue();

queue.enqueue(1);

queue.enqueue(2);

queue.dequeue(); // 1

queue.front(); // 2

1. **Linked List Implementation**
   * **Masala:** JavaScriptda linked listni amalga oshiring.
   * **Natija:**

javascript

Copy code

class Node {

constructor(element) {

this.element = element;

this.next = null;

}

}

class LinkedList {

constructor() {

this.head = null;

this.size = 0;

}

add(element) {

const node = new Node(element);

let current;

if (this.head === null) {

this.head = node;

} else {

current = this.head;

while (current.next) {

current = current.next;

}

current.next = node;

}

this.size++;

}

insertAt(element, index) {

if (index < 0 || index > this.size) return false;

const node = new Node(element);

let current, previous;

current = this.head;

if (index === 0) {

node.next = this.head;

this.head = node;

} else {

current = this.head;

let it = 0;

while (it < index) {

it++;

previous = current;

current = current.next;

}

node.next = current;

previous.next = node;

}

this.size++;

}

removeFrom(index) {

if (index < 0 || index >= this.size) return null;

let current = this.head;

let previous = null;

let it = 0;

if (index === 0) {

this.head = current.next;

} else {

while (it < index) {

it++;

previous = current;

current = current.next;

}

previous.next = current.next;

}

this.size--;

return current.element;

}

removeElement(element) {

let current = this.head;

let previous = null;

while (current !== null) {

if (current.element === element) {

if (previous === null) {

this.head = current.next;

} else {

previous.next = current.next;

}

this.size--;

return current.element;

}

previous = current;

current = current.next;

}

return null;

}

indexOf(element) {

let count = 0;

let current = this.head;

while (current !== null) {

if (current.element === element) return count;

count++;

current = current.next;

}

return -1;

}

isEmpty() {

return this.size === 0;

}

sizeOfList() {

return this.size;

}

printList() {

let current = this.head;

let str = '';

while (current) {

str += current.element + ' ';

current = current.next;

}

return str.trim();

}

}

const ll = new LinkedList();

ll.add(1);

ll.add(2);

ll.insertAt(3, 1);

ll.removeFrom(2); // 2

ll.indexOf(3); // 1

ll.printList(); // "1 3"

1. **Binary Search Tree Implementation**
   * **Masala:** JavaScriptda binary search tree'ni amalga oshiring.
   * **Natija:**

javascript

Copy code

class Node {

constructor(data, left = null, right = null) {

this.data = data;

this.left = left;

this.right = right;

}

}

class BST {

constructor() {

this.root = null;

}

add(data) {

const node = this.root;

if (node === null) {

this.root = new Node(data);

return;

} else {

const searchTree = function (node) {

if (data < node.data) {

if (node.left === null) {

node.left = new Node(data);

return;

} else if (node.left !== null) {

return searchTree(node.left);

}

} else if (data > node.data) {

if (node.right === null) {

node.right = new Node(data);

return;

} else if (node.right !== null) {

return searchTree(node.right);

}

} else {

return null;

}

};

return searchTree(node);

}

}

findMin() {

let current = this.root;

while (current.left !== null) {

current = current.left;

}

return current.data;

}

findMax() {

let current = this.root;

while (current.right !== null) {

current = current.right;

}

return current.data;

}

find(data) {

let current = this.root;

while (current.data !== data) {

if (data < current.data) {

current = current.left;

} else {

current = current.right;

}

if (current === null) {

return null;

}

}

return current;

}

isPresent(data) {

let current = this.root;

while (current) {

if (data === current.data) {

return true;

}

if (data < current.data) {

current = current.left;

} else {

current = current.right;

}

}

return false;

}

remove(data) {

const removeNode = function (node, data) {

if (node == null) {

return null;

}

if (data == node.data) {

if (node.left == null && node.right == null) {

return null;

}

if (node.left == null) {

return node.right;

}

if (node.right == null) {

return node.left;

}

var tempNode = node.right;

while (tempNode.left !== null) {

tempNode = tempNode.left;

}

node.data = tempNode.data;

node.right = removeNode(node.right, tempNode.data);

return node;

} else if (data < node.data) {

node.left = removeNode(node.left, data);

return node;

} else {

node.right = removeNode(node.right, data);

return node;

}

};

this.root = removeNode(this.root, data);

}

}

const bst = new BST();

bst.add(9);

bst.add(4);

bst.add(17);

bst.add(3);

bst.add(6);

bst.add(22);

bst.add(5);

bst.add(7);

bst.add(20);

bst.findMin(); // 3

bst.findMax(); // 22

bst.isPresent(4); // true

1. **Graph Implementation**
   * **Masala:** JavaScriptda graphni amalga oshiring.
   * **Natija:**

javascript

Copy code

class Graph {

constructor() {

this.adjacencyList = {};

}

addVertex(vertex) {

if (!this.adjacencyList[vertex]) {

this.adjacencyList[vertex] = [];

}

}

addEdge(v1, v2) {

this.adjacencyList[v1].push(v2);

this.adjacencyList[v2].push(v1);

}

removeEdge(v1, v2) {

this.adjacencyList[v1] = this.adjacencyList[v1].filter(v => v !== v2);

this.adjacencyList[v2] = this.adjacencyList[v2].filter(v => v !== v1);

}

removeVertex(vertex) {

while (this.adjacencyList[vertex].length) {

const adjacentVertex = this.adjacencyList[vertex].pop();

this.removeEdge(vertex, adjacentVertex);

}

delete this.adjacencyList[vertex];

}

depthFirstRecursive(start) {

const result = [];

const visited = {};

const adjacencyList = this.adjacencyList;

(function dfs(vertex) {

if (!vertex) return null;

visited[vertex] = true;

result.push(vertex);

adjacencyList[vertex].forEach(neighbor => {

if (!visited[neighbor]) {

return dfs(neighbor);

}

});

})(start);

return result;

}

breadthFirst(start) {

const queue = [start];

const result = [];

const visited = {};

visited[start] = true;

let currentVertex;

while (queue.length) {

currentVertex = queue.shift();

result.push(currentVertex);

this.adjacencyList[currentVertex].forEach(neighbor => {

if (!visited[neighbor]) {

visited[neighbor] = true;

queue.push(neighbor);

}

});

}

return result;

}

}

const g = new Graph();

g.addVertex('A');

g.addVertex('B');

g.addVertex('C');

g.addVertex('D');

g.addVertex('E');

g.addVertex('F');

g.addEdge('A', 'B');

g.addEdge('A', 'C

4o

davom ettir

ChatGPT

javascript

Copy code

g.addEdge('B', 'D');

g.addEdge('C', 'E');

g.addEdge('D', 'E');

g.addEdge('D', 'F');

g.addEdge('E', 'F');

g.depthFirstRecursive('A'); // ['A', 'B', 'D', 'E', 'C', 'F']

g.breadthFirst('A'); // ['A', 'B', 'C', 'D', 'E', 'F']

**81-90. Functional Programming**

1. **Map Function Implementation**
   * **Masala:** Array.prototype.map funksiyasini o'zingiz amalga oshiring.
   * **Natija:**

javascript

Copy code

Array.prototype.customMap = function(callback) {

const result = [];

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

result.push(callback(this[i], i, this));

}

return result;

};

[1, 2, 3].customMap(x => x \* 2); // [2, 4, 6]

1. **Filter Function Implementation**
   * **Masala:** Array.prototype.filter funksiyasini o'zingiz amalga oshiring.
   * **Natija:**

javascript

Copy code

Array.prototype.customFilter = function(callback) {

const result = [];

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

if (callback(this[i], i, this)) {

result.push(this[i]);

}

}

return result;

};

[1, 2, 3].customFilter(x => x > 1); // [2, 3]

1. **Reduce Function Implementation**
   * **Masala:** Array.prototype.reduce funksiyasini o'zingiz amalga oshiring.
   * **Natija:**

javascript

Copy code

Array.prototype.customReduce = function(callback, initialValue) {

let accumulator = initialValue !== undefined ? initialValue : this[0];

for (let i = initialValue !== undefined ? 0 : 1; i < this.length; i++) {

accumulator = callback(accumulator, this[i], i, this);

}

return accumulator;

};

[1, 2, 3].customReduce((acc, x) => acc + x, 0); // 6

1. **Flat Function Implementation**
   * **Masala:** Array.prototype.flat funksiyasini o'zingiz amalga oshiring.
   * **Natija:**

javascript

Copy code

Array.prototype.customFlat = function(depth = 1) {

let result = [];

(function flat(array, depth) {

for (const item of array) {

if (Array.isArray(item) && depth > 0) {

flat(item, depth - 1);

} else {

result.push(item);

}

}

})(this, depth);

return result;

};

[1, [2, [3]]].customFlat(1); // [1, 2, [3]]

1. **Compose Function**
   * **Masala:** compose funksiyasini amalga oshirish.
   * **Natija:**

javascript

Copy code

const compose = (...fns) => value => fns.reduceRight((acc, fn) => fn(acc), value);

const add = x => x + 1;

const multiply = x => x \* 2;

const composed = compose(add, multiply);

composed(5); // 11 (5 \* 2 + 1)

1. **Pipe Function**
   * **Masala:** pipe funksiyasini amalga oshirish.
   * **Natija:**

javascript

Copy code

const pipe = (...fns) => value => fns.reduce((acc, fn) => fn(acc), value);

const add = x => x + 1;

const multiply = x => x \* 2;

const piped = pipe(add, multiply);

piped(5); // 12 ((5 + 1) \* 2)

1. **Memoization**
   * **Masala:** Funksiyani memoize qilish.
   * **Natija:**

javascript

Copy code

const memoize = fn => {

const cache = {};

return (...args) => {

const key = JSON.stringify(args);

if (!cache[key]) {

cache[key] = fn(...args);

}

return cache[key];

};

};

const add = (a, b) => a + b;

const memoizedAdd = memoize(add);

memoizedAdd(1, 2); // 3

1. **Currying**
   * **Masala:** Funksiyani currying qilish.
   * **Natija:**

javascript

Copy code

const curry = fn => {

const curried = (...args) => {

if (args.length >= fn.length) {

return fn(...args);

} else {

return (...next) => curried(...args, ...next);

}

};

return curried;

};

const add = (a, b) => a + b;

const curriedAdd = curry(add);

curriedAdd(1)(2); // 3

1. **Partial Application**
   * **Masala:** Funksiyaning qisman qo'llanishini amalga oshirish.
   * **Natija:**

javascript

Copy code

const partial = (fn, ...fixedArgs) => (...remainingArgs) => fn(...fixedArgs, ...remainingArgs);

const add = (a, b, c) => a + b + c;

const partialAdd = partial(add, 1, 2);

partialAdd(3); // 6 (1 + 2 + 3)

1. **Once Function**
   * **Masala:** Funksiya faqat bir marta bajariladigan qilib amalga oshiring.
   * **Natija:**

javascript

Copy code

const once = fn => {

let done = false;

return (...args) => {

if (!done) {

done = true;

return fn(...args);

}

};

};

const log = () => console.log('Called');

const onceLog = once(log);

onceLog(); // "Called"

onceLog(); // nothing happens

**91-100. Advanced Topics**

1. **Deep Clone**
   * **Masala:** JavaScript'da obyektni chuqur nusxalash.
   * **Natija:**

javascript

Copy code

const deepClone = obj => {

if (obj === null || typeof obj !== 'object') return obj;

if (Array.isArray(obj)) {

return obj.map(deepClone);

}

const clonedObj = {};

for (const key in obj) {

clonedObj[key] = deepClone(obj[key]);

}

return clonedObj;

};

const obj = { a: 1, b: { c: 2 } };

const clonedObj = deepClone(obj);

clonedObj.b.c = 3;

obj.b.c; // 2

1. **Debounce Function**
   * **Masala:** Debounce funksiyasini amalga oshirish.
   * **Natija:**

javascript

Copy code

const debounce = (func, delay) => {

let timeout;

return (...args) => {

clearTimeout(timeout);

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

};

};

const log = () => console.log('Debounced');

const debouncedLog = debounce(log, 1000);

window.addEventListener('resize', debouncedLog);

1. **Throttle Function**
   * **Masala:** Throttle funksiyasini amalga oshirish.
   * **Natija:**

javascript

Copy code

const throttle = (func, limit) => {

let inThrottle;

return (...args) => {

if (!inThrottle) {

func(...args);

inThrottle = true;

setTimeout(() => (inThrottle = false), limit);

}

};

};

const log = () => console.log('Throttled');

const throttledLog = throttle(log, 1000);

window.addEventListener('resize', throttledLog);

1. **Deep Merge**
   * **Masala:** Ikkita obyektni chuqur birlashtirish.
   * **Natija:**

javascript

Copy code

const deepMerge = (target, source) => {

for (const key in source) {

if (source[key] instanceof Object && key in target) {

Object.assign(source[key], deepMerge(target[key], source[key]));

}

}

Object.assign(target || {}, source);

return target;

};

const obj1 = { a: 1, b: { c: 2 } };

const obj2 = { b: { d: 3 } };

const mergedObj = deepMerge(obj1, obj2);

mergedObj; // { a: 1, b: { c: 2, d: 3 } }

1. **Flatten Array**
   * **Masala:** Array'ni chuqur tekislash.
   * **Natija:**

javascript

Copy code

const flatten = array => {

const result = [];

(function flat(arr) {

arr.forEach(el => {

if (Array.isArray(el)) {

flat(el);

} else {

result.push(el);

}

});

})(array);

return result;

};

flatten([1, [2, [3, [4]]]]); // [1, 2, 3, 4]

1. **Event Emitter**
   * **Masala:** JavaScript'da Event Emitter'ni amalga oshirish.
   * **Natija:**

javascript

Copy code

class EventEmitter {

constructor() {

this.events = {};

}

on(event, listener) {

if (!this.events[event]) {

this.events[event] = [];

}

this.events[event].push(listener);

}

emit(event, ...args) {

if (this.events[event]) {

this.events[event].forEach(listener => listener(...args));

}

}

off(event, listenerToRemove) {

if (!this.events[event]) return;

this.events[event] = this.events[event].filter(listener => listener !== listenerToRemove);

}

}

const emitter = new EventEmitter();

const log = msg => console.log(msg);

emitter.on('message', log);

emitter.emit('message', 'Hello World'); // "Hello World"

emitter.off('message', log);

emitter.emit('message', 'Hello Again'); // nothing happens

1. **Promise.all Implementation**
   * **Masala:** Promise.all funksiyasini o'zingiz amalga oshiring.
   * **Natija:**

javascript

Copy code

const customPromiseAll = promises => {

return new Promise((resolve, reject) => {

let results = [];

let completedPromises = 0;

promises.forEach((promise, index) => {

promise

.then(result => {

results[index] = result;

completedPromises++;

if (completedPromises === promises.length) {

resolve(results);

}

})

.catch(err => reject(err));

});

});

};

const p1 = Promise.resolve(1);

const p2 = Promise.resolve(2);

customPromiseAll([p1, p2]).then(results => console.log(results)); // [1, 2]

1. **Promise.race Implementation**
   * **Masala:** Promise.race funksiyasini o'zingiz amalga oshiring.
   * **Natija:**

javascript

Copy code

const customPromiseRace = promises => {

return new Promise((resolve, reject) => {

promises.forEach(promise => {

promise.then(resolve).catch(reject);

});

});

};

const p1 = new Promise(resolve => setTimeout(resolve, 500, 'one'));

const p2 = new Promise(resolve => setTimeout(resolve, 100, 'two'));

customPromiseRace([p1, p2]).then(result => console.log(result)); // "two"

1. **Polyfill for Object.assign**
   * **Masala:** Object.assign funksiyasini o'zingiz amalga oshiring.
   * **Natija:**

javascript

Copy code

const customAssign = (target, ...sources) => {

sources.forEach(source => {

if (source) {

for (const key in source) {

if (source.hasOwnProperty(key)) {

target[key] = source[key];

}

}

}

});

return target;

};

const obj1 = { a: 1 };

const obj2 = { b: 2 };

const obj3 = customAssign({}, obj1, obj2);

obj3; // { a: 1, b: 2 }

1. **Polyfill for Object.create** - **Masala:** Object.create funksiyasini o'zingiz amalga oshiring. - **Natija:**

javascript

Copy code

const customCreate = proto => {

function F() {}

F.prototype = proto;

return new F();

};

const person = { isHuman: false };

const me = customCreate(person);

me.isHuman = true;

me.isHuman; // true

Ushbu mashqlar sizga JavaScript va TypeScriptni chuqurroq o'rganishga yordam beradi. Har bir masala uchun keltirilgan natija kodini o'z-o'zidan sinab ko'ring va tushuning. Agar savollaringiz bo'lsa yoki qo'shimcha tushuntirishlar kerak bo'lsa, bemalol so'rang. Omad!

4o

Начало формы

Конец формы