Basic Mapping:

1. Given an array of numbers, double each number using map().

String Transformation:

1. Given an array of strings, capitalize the first letter of each string using map().

Array of Objects Transformation:

1. Given an array of objects representing people with name and age properties, create a new array containing only the names of these people using map().

Array Index Access:

1. Given an array of numbers, create a new array where each element is the original number multiplied by its index in the array using map().

Array Filtering and Mapping:

1. Given an array of numbers, create a new array containing only the even numbers, and double each even number using map().

Nested Array Transformation:

1. Given a 2D array of numbers, transform it into a new 2D array where each number is squared using map().

Using Map with Other Array Methods:

1. Given an array of strings, filter out the strings that contain fewer than 5 characters, and capitalize the first letter of each remaining string using map().

Mapping with Conditional Logic:

1. Given an array of numbers, create a new array where each number is replaced by "even" if it's even, and "odd" if it's odd using map().

Mapping with Function Composition:

1. Given an array of strings, transform each string to uppercase and then append " - LENGTH: {length}" to each string, indicating its length, using map().

Mapping with Arrow Functions:

1. Rewrite any of the above examples using arrow functions within the map() method.

Filter by Property Value:

1. Given an array of objects representing products with name and price properties, filter out the products with a price greater than $50.

Filter by String Length:

1. Given an array of strings, filter out the strings that have a length greater than 5 characters.

Filter by Number Range:

1. Given an array of numbers, filter out the numbers that are greater than 100 and less than 1000.

Filter by Property Value with Condition:

1. Given an array of objects representing students with name, age, and grade properties, filter out the students who are younger than 18 and have a grade lower than 80.

Filter by Multiple Conditions:

1. Given an array of objects representing employees with name, age, and department properties, filter out the employees who are older than 30 and work in the "Sales" department.

Filter Unique Values:

1. Given an array of numbers, filter out the duplicate numbers, so the resulting array contains only unique values.

Filter Falsy Values:

1. Given an array containing various data types (such as strings, numbers, booleans, null, undefined), filter out the falsy values (e.g., empty strings, zero, false, null, undefined).

Filter by Array Intersection:

1. Given two arrays of numbers, filter out the numbers that are common to both arrays.

Filter by Array Exclusion:

1. Given two arrays of numbers, filter out the numbers that are present in the first array but not in the second array.

Filter by Regular Expression:

1. Given an array of strings, filter out the strings that match a specific regular expression pattern.

Some practical JavaScript interview questions related to promises and arrays:

1. Write a function that takes an array of numbers and returns a promise that resolves with the sum of all numbers in the array.

2. Create a function that accepts an array of promises and returns a promise that resolves with an array of their resolved values.

3. Implement a function that takes an array of URLs and fetches them asynchronously using promises, returning a promise that resolves with an array of response data.

4. Write a program to filter an array of numbers asynchronously using promises, returning a promise that resolves with the filtered array.

5. Create a function that accepts an array of file paths and reads them asynchronously using promises, returning a promise that resolves with an array of file contents.

1. reversing using for loop

2. ⁠[2,5,7,9] convert to [7,9,2,5]

3. ⁠even numbers square cheyyan array lle

4. ⁠oru array ll korch number athil eathoke number add add cheythal ann oru value kittuka ennulath kanan eg. [2,4,6,8,9,7,15] ithil eathoke numbers add cheythal ann 21 kittuka aa numbers printcheyyuka

5. ⁠prime numbers print cheyyuka using for loop

6. ⁠array reverse cheyyuka puthiya oru variable create cheyyathe swapping method

7. ⁠duplicate elements print cheyyuka

8. ⁠odd number square cheyyukayum even number cube cheyyukayum cheyyuka in a array

9. ⁠second largest number print

10. ⁠3 rd largest number print

11. ⁠oru arrayile 1-100 ll prime numbers oke square cheyuka baaki ullath normal ayit print cheyyuka

12. ⁠array 1-10 ll even numbers true ennum odd numbers false ennum kanikuka

13. ⁠array list even or odd numbers remove chyth print cheyyuka like filter

14. ⁠[2,4,6,8,9,7,15]. 2 square cheyyuka 4 cube cheyyuka 6 square cheyyuka 8 cube like angane array ith thanne avanam enn illa

15. ⁠pinne count kanuka [2,6,3,4,2,1,6,5,7,8,8,6,5,9,0,5] ith oru array out kittendath duplicate ellemnts eathanennum ethra indennum like output = (6:3,2:2,5:2,8:2)ingane  
  
  
Primitive vs Non-Primitive Data Types:

Primitive Data Types: String, Number, Boolean, Null, Undefined, Symbol, and BigInt. They are immutable and stored by value.

Non-Primitive Data Types: Objects (including arrays, functions, etc.). They are mutable and stored by reference.

Is JavaScript a statically typed or a dynamically typed language?:

JavaScript is a dynamically typed language, meaning variables do not require a type declaration and types are resolved at runtime.

Undefined vs Not Defined:

Undefined: A variable declared but not assigned a value.

Not Defined: A variable that has not been declared in the accessible scope.

null vs undefined:

null: Explicitly set to indicate "no value".

undefined: Default value for uninitialized variables.

NaN:

Stands for "Not-a-Number". Indicates that a value is not a legal number.

Strict Mode in JavaScript:

A way to opt-in to a restricted variant of JavaScript by adding "use strict"; at the beginning of a script or function. Helps catch common coding errors and "unsafe" actions.

typeof Operator:

Used to determine the type of a variable. Example: typeof 42; // "number"

Type of Errors:

SyntaxError: Incorrect syntax.

ReferenceError: Referencing a variable that is not declared.

TypeError: Incorrect type usage.

RangeError: Numbers outside allowed range.

EvalError: Issues with eval().

Variables and Scoping

let vs var:

var: Function-scoped or globally-scoped.

let: Block-scoped.

const vs let:

const: Block-scoped and immutable (the binding itself is immutable, not the value).

let: Block-scoped and mutable.

Scope:

The context in which variables are accessible. Can be global or local.

Type of Scope:

Global, Function, Block (introduced with let and const).

Block Scope:

Variables declared inside a block {} with let or const are confined to that block.

Scope Chain:

Hierarchy of scopes used to resolve variable references. Functions can access variables from their own scope and outer scopes.

Shadowing:

Declaring a variable in a local scope with the same name as a variable in an outer scope.

Hoisting:

JavaScript's behavior of moving variable and function declarations to the top of their containing scope during compilation.

TDZ (Temporal Dead Zone):

The time between entering a block and the declaration being processed, where accessing a variable results in a ReferenceError.

Lexical Scope:

Scope determined by the physical structure of the code, where nested functions have access to variables declared in their outer scope.

Lexical Environment:

An environment containing the bindings of identifiers to their values in the lexical scope.

Functions

Function Statement:

Regular function declaration. Example: function foo() {}

Function Expression:

Function defined within an expression. Example: const foo = function() {};

Anonymous Function:

Function without a name. Example: const foo = function() {};

Named Function Expression:

Function expression with a name. Example: const foo = function bar() {};

First Class Functions:

Functions treated as first-class citizens, meaning they can be assigned to variables, passed as arguments, and returned from other functions.

Higher Order Functions:

Functions that take other functions as arguments or return them. Example: array.map(callback)

Arrow Functions:

Concise syntax for function expressions. They do not have their own this. Example: const foo = () => {};

This Keyword:

Refers to the context in which a function is called.

Function Object:

Functions in JavaScript are objects and have properties and methods.

Function Constructors:

Creating a function dynamically. Example: const sum = new Function('a', 'b', 'return a + b');

Generator Functions:

Functions that can pause and resume execution using yield. Example: function\* gen() { yield 1; yield 2; }

Factory Function:

A function that returns an object without using the new keyword.

IIFE (Immediately Invoked Function Expression):

A function that runs as soon as it is defined. Example: (function() { /\* code \*/ })();

Pure Functions:

Functions that return the same result given the same inputs and have no side effects.

Call, Apply, Bind:

Methods to set the this value explicitly.

call: Calls a function with a given this value and arguments.

apply: Calls a function with a given this value and arguments provided as an array.

bind: Returns a new function with a given this value.

Currying:

Transforming a function with multiple arguments into a series of functions each taking a single argument.

Closures:

Functions that have access to their own scope, outer function’s scope, and the global scope.

Advanced Functions and Promises

Promise:

An object representing the eventual completion or failure of an asynchronous operation.

Promise States:

Pending: Initial state, neither fulfilled nor rejected.

Fulfilled: Operation completed successfully.

Rejected: Operation failed.

Promise Chain:

Using then and catch to chain multiple asynchronous operations.

Promise Create:

Creating a new promise. Example: new Promise((resolve, reject) => { /\* code \*/ });

Async/Await:

Syntactic sugar for promises, making asynchronous code look synchronous.

Callback Hell:

Pyramid of doom caused by deeply nested callbacks.

Event Loop:

JavaScript’s concurrency model that handles asynchronous operations.

Micro Tasks:

Tasks that are processed after the currently executing script and before the next rendering.

Event Delegation:

Using a single event listener to manage events for multiple elements.

Event Bubbling and Stop Propagation:

Event Bubbling: Event propagates from the target element to the root.

stopPropagation: Method to stop the propagation of an event.

Events and DOM Manipulation

Event Listeners:

Functions that wait for a specific event to occur. Example: element.addEventListener('click', callback);

Event Loop Working:

Mechanism that handles execution of code, events, and callbacks.

JS Engine Working:

Executes JavaScript code, often through Just-In-Time (JIT) compilation.

V8 Engine:

JavaScript engine developed by Google, used in Chrome and Node.js.

Event Handling:

Managing events like clicks, key presses, etc.

Event Capture:

Phase where the event goes down to the target element.

Event Propagation:

Includes both capturing and bubbling phases.

Data Structures and Algorithms

Array Methods: Push, Pop, Shift, Unshift:

push: Adds elements to the end.

pop: Removes the last element.

shift: Removes the first element.

unshift: Adds elements to the beginning.

String Methods:

Common methods: charAt, concat, includes, indexOf, slice, split, toLowerCase, toUpperCase, trim.

Map, Reduce, Filter, forEach:

map: Creates a new array with the results of calling a function for every array element.

reduce: Executes a reducer function on each element, resulting in a single output value.

filter: Creates a new array with all elements that pass the test implemented by the provided function.

forEach: Executes a provided function once for each array element.

Map vs ForEach:

map returns a new array; forEach does not.

Set, WeakSet:

Set: Collection of unique values.

WeakSet: Collection of unique objects held weakly.

Map, WeakMap:

Map: Collection of key-value pairs.

WeakMap: Collection of key-value pairs where keys are objects held weakly.

Deep Copy vs Shallow Copy:

Shallow Copy: Copies only the first level of the object.

Deep Copy: Recursively copies all levels of the object.

Object-Oriented Programming

OOP Concepts in JavaScript:

Classes, Objects, Inheritance, Polymorphism, Encapsulation, Abstraction.

Prototype:

Mechanism by which JavaScript objects inherit properties and methods.

Prototypal Inheritance:

Objects inherit directly from other objects.

Classical Inheritance:

Creating classes and inheriting from them (introduced in ES6).

Constructor:

Special method for creating and initializing an object created with a class.

Prototype Chain:

Chain of objects connected through their prototypes.

Object Methods:

Methods like Object.assign, Object.keys, Object.values, Object.entries.

Object Function:

Functions like Object.create, Object.defineProperty.

Object Destructuring:

Extracting properties from objects into variables. Example: const {a, b} = obj;

Object.defineProperty vs defineProperties:

Object.defineProperty: Defines a new property or modifies an existing one.

Object.defineProperties: Defines new or modifies existing properties.

Object.fromEntries vs Object.entries:

Object.entries: Converts object to array of key-value pairs.

Object.fromEntries: Converts array of key-value pairs to object.

Modern JavaScript (ES6+)

ES6 Features:

New features like let, const, arrow functions, template literals, destructuring, classes, modules.

Template Literals:

String literals allowing embedded expressions. Example: Hello, ${name}

Destructuring:

Extracting values from arrays or properties from objects into variables.

Arrow Functions vs Regular Functions:

Arrow functions have a shorter syntax and lexically bind the this value.

Rest Operator:

Collects all remaining elements into an array. Example: function(...args) {}

Spread Operator:

Spreads elements of an array or object. Example: [...array]

Default Parameters:

Function parameters with default values. Example: function(a = 1) {}

Symbols:

Unique and immutable data type used as object keys.

BigInt:

For representing integers larger than 2^53-1.

Optional Chaining:

Allows safe access to deeply nested properties. Example: obj?.prop

Nullish Coalescing Operator:

Provides a default value when dealing with null or undefined. Example: a ?? b

Modules: Import and Export:

Syntax for importing and exporting modules.

Asynchronous JavaScript

Synchronous vs Asynchronous:

Synchronous: Tasks are completed one after the other.

Asynchronous: Tasks can be initiated and completed at a later time.

Event Loop:

Handles asynchronous operations and manages the execution of code.

Callbacks:

Functions passed as arguments to be executed after an operation completes.

Promises:

Objects representing the eventual completion of an asynchronous operation.

Async/Await:

Syntactic sugar for promises, making asynchronous code look synchronous.

Advanced Topics

JSON: stringify vs parse:

JSON.stringify: Converts JavaScript object to JSON string.

JSON.parse: Converts JSON string to JavaScript object.

LocalStorage vs SessionStorage vs Cookies:

LocalStorage: Stores data with no expiration date.

SessionStorage: Stores data for the duration of the page session.

Cookies: Stores data with an expiration date, sent to server with every HTTP request.

Service Worker:

Script that runs in the background, enabling features like push notifications and background sync.

Web APIs:

Browser APIs for interacting with the web, like fetch, localStorage.

HTTP Status Codes:

Codes indicating the result of an HTTP request (e.g., 200, 404).

Axios vs Fetch:

Axios: Library for making HTTP requests.

Fetch: Native JavaScript function for making HTTP requests.

Tree Shaking:

Optimization technique to remove unused code during bundling.

Polyfills:

Code that provides modern functionality on older browsers that do not support it.

PWA (Progressive Web Apps):

Web applications that use modern web capabilities to deliver app-like experiences.

Zone.js:

Library that helps track asynchronous operations and improve error handling in Angular.

Miscellaneous

NaN:

Indicates "Not-a-Number".

IsNaN:

Function to determine if a value is NaN. Example: isNaN('abc') // true

Regular Expressions:

Patterns used to match character combinations in strings.

Math Functions:

Built-in functions like Math.max, Math.min, Math.abs.

Math.random():

Generates a random number between 0 (inclusive) and 1 (exclusive).

Unary Operators:

Operators with a single operand. Example: ++, --, +, -.

Ternary Operator:

Shorthand for if-else. Example: condition ? value1 : value2

Bitwise Operators:

Operators for performing bitwise operations. Example: &, |, ^.

Comma Operator:

Evaluates each operand from left to right and returns the value of the last operand. Example: let a = (1, 2, 3); // a = 3

Eval:

Executes a string of JavaScript code. Use with caution due to security risks.

Uneval:

Creates a string representation of the source code of an object.

Type Casting: Implicit and Explicit:

Implicit: Automatic type conversion by JavaScript.

Explicit: Manual type conversion using constructors or methods.

Instanceof:

Checks if an object is an instance of a constructor or class. Example: obj instanceof Array

Event Queue:

Queue that holds events to be processed by the event loop.

Call Stack:

Stack data structure that stores function calls to be executed.

Default Parameters:

Function parameters with default values. Example: function greet(name = 'Guest') {}

Nested Templates:

Template literals inside other template literals.

Enhanced Objects:

Shorthand property names, method definitions, and computed property names introduced in ES6.

Single vs Multi Thread:

Single-threaded: Executes one command at a time.

Multi-threaded: Executes multiple commands concurrently.

Precedence:

Order in which operators are evaluated.

Accessors:

Getters and setters for object properties.

Everything: Advantages/Disadvantages, Differences:

General comparison and contrast of various features.

Loosely Typed Language:

Languages where variables do not have a fixed type and can change types dynamically.

Explain the difference between null and undefined in JavaScript.

How does the this keyword behave in different contexts, such as in global scope, object methods, and arrow functions?

What is a closure, and how can it be used to create private variables?

Describe how JavaScript's prototypal inheritance works.

Explain the concept of "hoisting" and how it affects variable and function declarations.

What is the event loop, and how does it affect asynchronous code execution in JavaScript?

How do setTimeout and setInterval work, and what are the differences between them?

Discuss the different ways to create objects in JavaScript and the advantages/disadvantages of each.

What are "IIFE" (Immediately Invoked Function Expressions) and their typical use cases?

How does the apply, call, and bind methods work, and how do they differ from each other?

Explain the concept of promises and how they are used to handle asynchronous operations.

How do async/await work, and how do they simplify working with promises?

Describe the various ways to handle errors in JavaScript (try-catch, promise rejections, etc.).

What is the difference between == and === in JavaScript?

Explain how JavaScript handles type coercion.

How does the JavaScript engine optimize performance through techniques like JIT (Just-In-Time) compilation?

What are Map and Set, and how do they differ from regular objects and arrays?

Describe the concept of "debouncing" and "throttling" in JavaScript and their use cases.

What is the purpose of the Symbol type, and how is it used in JavaScript?

Explain the concept of "dynamic typing" and how it affects JavaScript variables and functions.

How do JavaScript modules work, and what are the differences between CommonJS and ES6 modules?

What are the new features introduced in the latest ECMAScript specifications (e.g., ES6/ES2015, ES7/ES2016, etc.)?

Explain how the Object.create() method works and its use cases.

How can you use JavaScript's Proxy and Reflect objects to intercept and redefine operations on objects?

What is the difference between Object.freeze(), Object.seal(), and Object.preventExtensions()?

Describe the WeakMap and WeakSet types and their use cases.

Explain the concept of "currying" and how it is implemented in JavaScript.

How does JavaScript handle memory management and garbage collection?

What is the purpose of the Reflect API, and how does it interact with JavaScript's metaprogramming capabilities?

How do class and constructor work in JavaScript, and how do they relate to prototypal inheritance?

What are "generators" and "iterators," and how are they used in JavaScript?

Explain how the Promise.all(), Promise.race(), and Promise.any() methods work and their use cases.

What are "decorators" in JavaScript, and how can they be used with classes and methods?

How do JavaScript's import and export statements work, and how do they support module loading?

Explain how the Function.prototype methods like bind, call, and apply modify the behavior of functions.

What is the "prototype chain," and how does it affect object inheritance in JavaScript?

Describe the WeakRef and FinalizationRegistry objects and their use cases in JavaScript.

How does JavaScript's event delegation work, and why is it useful for optimizing event handling?

Explain the concept of "shadowing" in the context of variable scope and closures.

What is the difference between var, let, and const in terms of scoping and hoisting?

How can you create a "singleton" pattern in JavaScript, and what are its benefits and drawbacks?

Describe how the fetch API works and its advantages over XMLHttpRequest.

How can you use JavaScript to create and manage web workers for concurrent execution?

What are the differences between synchronous and asynchronous JavaScript, and how do they affect program execution?

How does JavaScript handle inheritance when using Object.setPrototypeOf() and Object.getPrototypeOf()?

Explain the role of async and await in handling asynchronous code and their impact on code readability.

What is "functional programming" in JavaScript, and how can concepts like "map", "filter", and "reduce" be used?

How do JavaScript's Array.prototype methods like find, some, and every work, and what are their use cases?

Describe how localStorage, sessionStorage, and cookies differ in terms of persistence and usage.+

How does the concept of "polymorphism" apply to JavaScript's type system and function behavior?