DOM

The DOM is an acronym for “Document Object Model which is a tree that shows relationships between elements as parents, children and siblings.

The Document Object Model (DOM) is the data representation of the objects that comprise the structure and content of a document on the web.

The Document Object Model (DOM) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

DOM stands for Document Object Model. It is a programming interface that allows us to create, change, or remove elements from the document.

The Document Object Model is a cross-platform and language-independent interface that treats an HTML or XML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree.

Shallow copy and deep copy (object copying)

1. Synchronous and asynchronous

https://www.geeksforgeeks.org/difference-between-synchronous-and-asynchronous-transmission/
https://www.w3schools.com/js/js_asynchronous.asp
https://www.geeksforgeeks.org/asynchronous-javascript/
https://www.geeksforgeeks.org/synchronous-and-asynchronous-in-javascript/
https://www.w3schools.com/js/js_async.asp

Synchronous code is executed in a single thread, meaning that each statement is executed one at a time and the program waits for each statement to finish before moving on to the next one. This can sometimes cause the program to pause or hang if a statement takes a long time to execute.

Asynchronous code, on the other hand, is executed in a non-blocking way. This means that instead of waiting for a statement to finish before moving on to the next one, the program continues executing and returns control to the caller. The task that needs to be executed is added to a queue and executed later, usually when it has completed or when there is available processing time. This can improve program performance by allowing multiple task s to be executed simultaneously and not blocking the main thread.

Asynchronous code is commonly used in programming tasks that involve network requests, file operations, or other tasks that may take a long time to complete. By using asynchronous code, the program can continue executing while waiting for the task to complete, and the user interface can remain responsive.

In JavaScript, asynchronous code is commonly implemented using callbacks, promises, and async/await syntax. These techniques allow code to be written in a more readable and maintainable way, while still taking advantage of the benefits of asynchronous execution.

2. Callbacks

https://www.w3schools.com/js/js_callback.asp
https://www.geeksforgeeks.org/javascript-callbacks/

A callback is *a function passed as an argument to another function,*

A callback function can run after another function has finished

Callbacks are used to handle the results of asynchronous operations in a non-blocking manner. Asynchronous operations are operations that take a significant amount of time to complete, such as network requests, file I/O, and database queries. If these operations were executed synchronously, the program would freeze and wait for the operation to complete before continuing. This can lead to a poor user experience, as the program would appear unresponsive.

3. Event loop

https://www.geeksforgeeks.org/what-is-an-event-loop-in-javascript/
https://www.geeksforgeeks.org/what-is-an-event-loop-in-javascript/
https://nodejs.org/en/learn/asynchronous-work/event-loop-timers-and-nexttick

the event loop is a mechanism that allows asynchronous code to be executed in a non-blocking way. It's a core component of the JavaScript runtime and is responsible for handling events and executing code in a specific order.

The event loop works by maintaining a queue of tasks, which includes all the code that needs to be executed. These tasks can be either synchronous or asynchronous. When a task is added to the queue, the event loop checks if there are any pending tasks to be executed. If there are no pending tasks, the event loop will take the first task from the queue and execute it.

If the task is synchronous, it will be executed immediately, and the event loop will move on to the next task in the queue.

If the task is asynchronous, it will be passed to the relevant API, such as a web API or a timer, and the event loop will move on to the next task in the queue. When the asynchronous task is completed, it will be added back to the queue, and the event loop will execute it.

In summary, the event loop is a mechanism that ensures that JavaScript code is executed in a specific order, allowing asynchronous code to be executed in a non-blocking way. It's a fundamental concept in modern web development, and understanding how it works is essential for writing efficient and performant JavaScript code.

4. Promises

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
https://www.w3schools.com/js/js_promise.asp
https://www.geeksforgeeks.org/javascript-promise/
https://www.w3schools.com/jsref/jsref_obj_promise.asp

Promises in JavaScript are a way of handling asynchronous operations. They are objects that represent the eventual completion or failure of an asynchronous operation and allow you to write asynchronous code that is easier to read and maintain.

Promises have three states:

Pending: the initial state, before the promise has resolved or rejected.

Fulfilled: the state when the promise has successfully completed.

Rejected: the state when the promise has failed.

Promises have two main methods:

then(): This method is used to handle a successful response from the promise. It takes two arguments: a callback function to handle the success case, and an optional callback function to handle the error case.

catch(): This method is used to handle an error response from the promise. It takes one argument: a callback function to handle the error case.

Promise.all

The Promise.all(iterable) method takes an iterable (such as an array) of promises as input and returns a single Promise that resolves when all of the promises in the iterable have resolved, or rejects as soon as one of the promises in the iterable rejects.

Promise.race

The Promise.race(iterable) method returns a Promise that fulfills or rejects as soon as one of the promises in the iterable fulfills or rejects, with the value or reason from that promise.

promise.settled/finally

The finally() method returns a Promise that is resolved or rejected when the original Promise is settled (resolved or rejected). It allows you to specify a callback function that is executed when the Promise is settled, regardless of whether it was fulfilled or rejected.

5. Hoisting

https://www.w3schools.com/js/js_hoisting.asp
https://www.geeksforgeeks.org/javascript-hoisting/
https://developer.mozilla.org/en-US/docs/Glossary/Hoisting

JavaScript Hoisting refers to the process whereby the interpreter appears to move the declaration of functions, variables, classes, or imports to the top of their scope, prior to execution of the code.

Hoisting is a term used in JavaScript to describe the behavior where variable declarations and function declarations are moved to the top of their respective scopes during the compilation or execution phase. This means that they are available for use before they are actually declared in the code.

It's important to note that only the declarations of variables and functions are hoisted, not their assignments. So if a variable is declared and initialized on separate lines, only the declaration will be hoisted, not the initialization.

Hoisting can make code easier to read and understand, but it can also lead to unexpected behavior if not used carefully. It's recommended to declare variables and functions at the beginning of their respective scopes to avoid confusion and bugs.

6. Arrow functions

https://www.w3schools.com/js/js_arrow_function.asp
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
https://www.geeksforgeeks.org/arrow-functions-in-javascript/

Arrow functions are a shorthand syntax for creating functions in JavaScript. They were introduced in ES6 (ECMAScript 2015) and provide a more concise syntax compared to traditional function expressions.

Arrow functions have a few key differences compared to traditional functions:

Shorter syntax: Arrow functions use a more concise syntax, with no need for the function keyword, return statement (for one-liner functions), or curly braces for single-line functions.

Lexical this: Arrow functions do not have their own this context, and instead inherit the this value from their enclosing scope. This can make them easier to use and avoid confusion with the this keyword in JavaScript.

No arguments object: Arrow functions also do not have their own arguments object, which is an array-like object that contains all the arguments passed to a function.

Instead, arrow functions can use the rest parameter syntax (...) to gather all the arguments passed to them into an array.

Overall, arrow functions are a powerful feature of JavaScript that can make code more concise and easier to read.

Here are some benefits of using arrow functions:

Concise syntax: Arrow functions use a shorter syntax compared to traditional function expressions, making the code easier to read and write.

Implicit return: Arrow functions automatically return the value of the expression without using the return keyword, as long as the expression is on a single line.

Lexical this binding: The value of this inside an arrow function is determined by the surrounding context, rather than the function itself. This can make it easier to access the correct this value without having to use workarounds like bind, call, or apply.

No binding of arguments: Arrow functions do not bind their own arguments object, which can make the code simpler and more predictable.

Better suited for functional programming: Arrow functions are well-suited for functional programming paradigms, such as passing functions as arguments or returning functions from other functions.

Here are a few key points about arrow functions and `this`:

1. No Own `this`: Arrow functions do not have their own `this` context. The value of `this` inside an arrow function is always inherited from the enclosing scope.
2. Cannot be used as constructors: Since arrow functions do not have their own `this`, they cannot be used as constructors.
3. Fixed `this` Binding: The `this` value inside an arrow function remains fixed and cannot be changed, even if the function is called in a different context.

Advantages of Arrow Functions

- Arrow functions reduce the size of the code.

- The return statement and function brackets are optional for single-line functions.
- It increases the readability of the code.
- Arrow functions provide a lexical this binding. It means, they inherit the value of “this” from the enclosing scope. This feature can be advantageous when dealing with event listeners or callback functions where the value of “this” can be uncertain.

Limitations of Arrow Functions

- Arrow functions do not have the prototype property.
- Arrow functions cannot be used with the new keyword.
- Arrow functions cannot be used as constructors.
- These functions are anonymous and it is hard to debug the code.
- Arrow functions cannot be used as generator functions that use the yield keyword to return multiple values over time.

7. Clousers

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures
https://www.geeksforgeeks.org/closure-in-javascript/
https://www.geeksforgeeks.org/difference-between-scope-and-closures-in-javascript/

Closures are an important concept in JavaScript that allow you to create functions with private variables and maintain state across multiple function calls.

A closure is created when a function is defined inside another function and has access to variables in the outer function's scope, even after the outer function has returned. This allows the inner function to "close over" the variables in the outer function and maintain a reference to them, even if they are no longer in scope.

Closures are a powerful feature of JavaScript that allow you to create functions with private variables and state, making it easier to write modular and reusable code. They are commonly used in libraries and frameworks, and can be used to create more advanced patterns like currying and memoization.

Closures are a powerful feature in JavaScript that allows a function to access variables from an enclosing scope, even after the outer function has finished executing.

Closures can be used to create private variables and methods.

Closures can be used to create functions with preset arguments.

Closures are often used in asynchronous code to maintain access to variables when a callback executes.

Closures can be used to cache the results of expensive function calls.

Using closures to create partially applied functions.

Reasons for Using Closures:

Encapsulation and Data Privacy:
Function Factories:
Callback Functions:
Memoization:
Module Pattern:

8. Debouncing and throttling

https://www.geeksforgeeks.org/difference-between-debouncing-and-throttling/
https://www.geeksforgeeks.org/difference-between-debouncing-throttling-in-javascript/

Debouncing is a technique that is used to limit the number of times a function is executed based on how frequently an event is triggered.

Throttling is a technique that is used to limit the frequency at which a function is called, regardless of how frequently an event is triggered.

Debouncing and throttling are two techniques used in JavaScript to optimize the performance of event handlers that can be triggered frequently, such as scroll or resize events. These techniques help to reduce the number of times the event handler is called and improve the efficiency of the code.

Debouncing involves delaying the execution of the event handler until after a certain amount of time has elapsed since the last time the event was triggered. This means that the event handler will only be called once the event has stopped being triggered for a specified period of time. This can help to prevent multiple rapid-fire executions of the event handler that may be unnecessary.

9. Call,Apply,bind

https://www.geeksforgeeks.org/explain-call-apply-and-bind-methods-in-javascript/

https://www.freecodecamp.org/news/understand-call-apply-and-bind-in-javascript-with-examples/

Call

Call is a function that helps you change the context of the invoking function.It helps you replace the value of this inside a function with whatever value you want.

Apply

Apply is very similar to the call function. The only difference is that in apply you can pass an array as an argument list.

bind

Bind is a function that helps you create another function that you can execute later with the new context of this that is provided.

10. Map

https://www.w3schools.com/jsref/jsref_map.asp
https://www.geeksforgeeks.org/javascript-array-map-method/
https://www.w3schools.com/js/js_maps.asp

map() is a higher-order function in JavaScript that operates on arrays. It takes a function as an argument and applies that function to each element of the array, creating a new array with the results. The new array will have the same length as the original array, with each element transformed according to the function provided.

The map() function takes a callback function as its argument, which is called once for each element in the array. The callback function takes three arguments: the current element being processed, the index of that element in the array, and the array itself. However, in most cases, we only need the first argument.

map() is a useful function for transforming data in an array without modifying the original array. It's often used in conjunction with other higher-order functions like filter() and reduce() to perform more complex operations on arrays.

11. Object Data Types:

https://www.w3schools.com/js/js_datatypes.asp
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures
https://www.geeksforgeeks.org/primitive-and-non-primitive-data-types-in-javascript/

12. Events

https://www.w3schools.com/js/js_events.asp
https://www.geeksforgeeks.org/javascript-events/
https://www.tutorialspoint.com/javascript/javascript_events.htm

Events in JavaScript are actions or occurrences that happen in the browser, such as a user clicking on a button or the browser finishing loading a page. In order to respond to these events, JavaScript provides a way to attach event listeners to elements in the DOM (Document Object Model).

Here are some of the most common events in JavaScript:

Click: This event is triggered when a user clicks on an element, such as a button or a link.

Mouseover: This event is triggered when the user moves the mouse over an element.

Mouseout: This event is triggered when the user moves the mouse away from an element.

Keydown: This event is triggered when the user presses a key on the keyboard.

Load: This event is triggered when a page finishes loading.

Submit: This event is triggered when a user submits a form.

To attach an event listener to an element in the DOM, you can use the addEventListener() method. This method takes two arguments: the name of the event you want to listen for, and a function that will be called when the event is triggered.

13. Object copying

In JavaScript, object copying can be done in several ways, depending on whether you need a shallow copy or a deep copy of the object. Let's explore both types of copying.

Shallow Copy

A shallow copy of an object copies the object's properties, but if the property is a reference to another object, the reference address is copied, not the actual object. This means changes to nested objects will affect both the original and the copied object.

Using Object.assign()

Using Spread Operator

Deep Copy

A deep copy of an object means that all objects are copied recursively, so the copy is completely independent of the original.

This ensures that modifications to the new object do not affect the original object and vice versa.

Using JSON Methods

Using a Recursive Function

Using a Library

14. Server

https://www.w3schools.com/jsref/jsref_obj_object.asp
https://www.geeksforgeeks.org/javascript-object-reference/

In the context of JavaScript, a server is typically referring to a web server that runs on a computer and handles incoming requests from clients, such as web browsers.

JavaScript is a client-side programming language, meaning it runs in the web browser of the user who accesses a website or web application. However, JavaScript can also be used on the server-side, for example with the Node.js runtime environment, which allows developers to write server-side applications using JavaScript.

Node.js allows JavaScript to be used on the server-side by providing a runtime environment that runs JavaScript code outside of the browser. This means that JavaScript can be used to write web servers that can handle incoming requests and send responses back to clients.

In summary, in the context of JavaScript, a server usually refers to a web server that runs on a computer and handles incoming requests from clients, which can be written using Node.js and JavaScript.

15. Object referencing

https://www.geeksforgeeks.org/how-to-clone-a-javascript-object/
https://developer.mozilla.org/en-US/docs/Glossary/Deep_copy
https://www.freecodecamp.org/news/clone-an-object-in-javascript/

object referencing refers to the way that variables can refer to objects, and how changes made to an object through one variable will be reflected in any other variables that reference that same object.

When an object is created in JavaScript, it is stored in memory, and a variable that refers to that object actually contains a reference to its location in memory, rather than a copy of the object itself.

16. Spread and rest operator

Spread Operator:
Creating shallow copies of arrays and objects.
Concatenating arrays or objects.
Passing array elements as individual arguments to functions.

Rest Parameter:
Capturing variable numbers of function arguments into an array.
Handling an unknown number of parameters in function definitions.

17. Design Patterns

Some common JavaScript design patterns include:

Module Pattern: Encapsulates code into a single unit, providing a public API while hiding implementation details.
Singleton Pattern: Ensures a class has only one instance and provides a global point of access to it.
Observer Pattern: Allows an object (subject) to maintain a list of dependents (observers) and notify them of state changes.
Factory Pattern: Creates objects without specifying the exact class of object to be created.

18. Function expression

https://www.geeksforgeeks.org/javascript-function-expression/
https://developer.mozilla.org/en/docs/web/JavaScript/Reference/Operators/function

https://www.w3schools.com/js/js_function_definition.asp

a function expression is a way to define a function as a value and assign it to a variable or property. Function expressions can be named or anonymous, and can be defined in several ways.

Function expressions can also be defined as arrow functions, which are a shorthand way of writing function expressions:

Function expressions can be used in many different ways in JavaScript, including as arguments to other functions, as properties of objects, or as methods of objects. They are a powerful and flexible feature of the language that allows developers to write more expressive and functional code.

19. IIFE

https://www.geeksforgeeks.org/what-are-iife-immediately-invoked-function-expressions/
https://circleci.com/blog/ci-cd-for-js-iifes/#:~:text=A%20JavaScript%20IIFE%20(Immediately%20Invoked,useful%20for%20several%20use%20cases.
https://developer.mozilla.org/en-US/docs/Glossary/IIFE

IIFE stands for Immediately Invoked Function Expression. It is a design pattern in JavaScript that involves defining and calling a function at the same time. An IIFE is usually used to create a new scope, which can be used to prevent naming conflicts and to create private variables and functions.

IIFEs can also be used to create modules in JavaScript, which are collections of related functions and variables that can be reused across multiple files or parts of an application.

20. Array methods

https://www.w3schools.com/js/js_array_methods.asp
https://www.geeksforgeeks.org/javascript-array-methods/

JavaScript provides a variety of built-in methods for working with arrays. Here are some of the most commonly used array methods:

push: adds one or more elements to the end of an array and returns the new length of the array.

pop: removes the last element from an array and returns that element.

shift: removes the first element from an array and returns that element.

unshift: adds one or more elements to the beginning of an array and returns the new length of the array.

splice: changes the contents of an array by removing or replacing existing elements and/or adding new elements.

slice: returns a new array containing a portion of an existing array.

21. Execution context

https://www.freecodecamp.org/news/how-javascript-works-behind-the-scene-javascript-execution-context/
https://www.geeksforgeeks.org/execution-context-in-javascript/

the execution context is a conceptual container that holds information about the environment in which a particular piece of code is executed. Every time a function is called, a new execution context is created, which includes information about the function's variables, arguments, and scope chain.

The execution context can be thought of as a way of keeping track of where the code is currently running, and what variables and functions are available to it. It contains three main components:

Variable Environment - This component stores all the variables declared within the current scope, including function arguments and local variables. It also keeps track of any variables declared in the outer lexical environment (i.e., the enclosing function or global scope).

Lexical Environment - This component stores the current scope chain, which is used to resolve variable names. It's essentially a hierarchy of all the available scopes, with the innermost scope at the top and the outermost (global) scope at the bottom.

This Binding - This component refers to the value of the this keyword within the current context. It's determined based on how a function is called, and can be influenced by things like method binding, constructor calls, or explicit binding.

When a function is called, a new execution context is created and pushed onto the top of the execution stack. Once the function completes, its execution context is popped off the stack, and control returns to the calling context.

Understanding the execution context is essential for understanding how JavaScript code is executed, how variables and functions are scoped, and how the this keyword works. It's a foundational concept in JavaScript, and mastering it is key to becoming an effective JavaScript developer.

22. Filter()

https://www.w3schools.com/jsref/jsref_filter.asp
https://www.geeksforgeeks.org/how-to-filter-an-array-in-javascript/
https://www.tutorialspoint.com/javascript/array_filter.htm

In JavaScript, the filter() method is a built-in function that allows you to filter an array based on a given condition. It creates a new array containing only the elements that pass the condition specified by a provided function.

The filter() method takes a single argument, which is a callback function that is executed for each element in the array. The callback function takes three arguments: the current element being processed, the index of the current element, and the array being processed.

array.filter(callback(element[, index[, array]])[, thisArg])

Here's a breakdown of each parameter:

callback - the function that is called for each element in the array.

element - the current element being processed.

index (optional) - the index of the current element being processed.

array (optional) - the array being processed.

thisArg (optional) - the value to be used as this when executing the callback function.

The callback function must return a Boolean value. If it returns true, the current element will be included in the new filtered array. If it returns false, the current element will be excluded from the new array.

23. Reduce

https://www.w3schools.com/jsref/jsref_reduce.asp
https://www.geeksforgeeks.org/javascript-array-reduce-method/
https://www.tutorialspoint.com/javascript/array_reduce.htm

reduce() method is a built-in function that allows you to "reduce" an array to a single value. It works by executing a provided function for each element in the array, and "accumulating" the result of each function call into a single value.

The reduce() method takes two arguments: a callback function and an optional initial value. The callback function takes two arguments: an accumulator value and the current element being processed. The callback function can also take two additional arguments: the index of the current element, and the array being processed.

Here's the syntax for using the reduce() method:

array.reduce(callback(accumulator, currentValue[, index[, array]])[, initialValue])

Here's a breakdown of each parameter:

callback - the function that is called for each element in the array.

accumulator - the value returned by the previous iteration of the callback function, or the initialValue if this is the first iteration.

currentValue - the current element being processed.

index (optional) - the index of the current element being processed.

array (optional) - the array being processed.

initialValue (optional) - the initial value to be used as the accumulator. If this is not provided, the first element of the array will be used as the initial value.

24. Event bubbling/capturing/delegation

https://medium.com/@magenta2127/event-bubbling-and-event-delegation-4209bf40575c

https://www.geeksforgeeks.org/event-delegation-in-javascript/

events can propagate through the DOM tree in two ways: event bubbling and event capturing. Event delegation is a technique that leverages event bubbling to handle events more efficiently.

Event bubbling is the default propagation mode for events in the DOM. It means that an event starts at the element that was clicked on, and then bubbles up through its ancestors. For example, if you have a div element inside a section element inside the body element, and you click on the div, the event will first be triggered on the div, then on the section, and finally on the body.

What is Event Bubbling? Event Bubbling is a concept in the DOM (Document Object Model). It happens when an element receives an event, and that event bubbles up (or you can say is transmitted or propagated) to its parent and ancestor elements in the DOM tree until it gets to the root element.

Event capturing is the opposite of event bubbling. It means that an event starts at the highest ancestor in the DOM tree and then propagates down to the element that was clicked on. For example, if you have a div element inside a section element inside the body element, and you click on the div, the event will first be triggered on the body, then on the section, and finally on the div.

Event delegation is a technique that leverages event bubbling to handle events more efficiently. Instead of attaching an event listener to each individual element, you can attach a single event listener to a parent element and then use event.target to determine which child element was clicked on. This is especially useful when you have a large number of elements that you want to attach the same event listener to.

In JavaScript, call, bind, and apply are three methods that can be used to manipulate the this keyword and pass arguments to a function.

call and apply allow you to invoke a function with a specified this value and arguments passed in as an array (or an array-like object in the case of arguments), whereas bind allows you to create a new function with a specified this value and pre-defined arguments.

Here's a breakdown of each method:

25. Prototypal inheritance

https://javascript.info/prototype-inheritance
https://www.geeksforgeeks.org/explain-prototype-inheritance-in-javascript/
https://www.geeksforgeeks.org/prototypal-inheritance-using-__proto__-in-javascript/

Prototypal inheritance is a fundamental concept in JavaScript that allows objects to inherit properties and methods from their prototype. Every object in JavaScript has a prototype object, which is used to provide the object with its properties and methods.

When a property or method is called on an object, JavaScript first checks if that property or method exists on the object itself. If it does not, JavaScript looks to the object's prototype for the property or method. If it still does not exist on the prototype, JavaScript continues to look up the prototype chain until it reaches the top-level Object prototype.

To create an object that inherits from another object, you can use the Object.create() method. This method takes an object as its argument and creates a new object that inherits from the given object.

25 Prototype

Prototypes are a fundamental concept in JavaScript that enables inheritance and sharing of properties and methods across objects.

A prototype is an object from which other objects inherit properties and methods. Every function in JavaScript has a `prototype` property, which is used to build the `__proto__` property of instances created with that function.

Purpose:

- Share methods and properties across all instances of a constructor.
- Facilitate inheritance and method sharing to save memory.

you can not add a new property to an existing object constructor:
To add a new property to a constructor, you must add it to the constructor function:

6. Currying

https://www.geeksforgeeks.org/what-is-currying-function-in-javascript/
https://www.tutorialspoint.com/javascript/javascript_currying.htm
https://www.geeksforgeeks.org/why-is-currying-in-javascript-useful/

Currying is a functional programming technique that involves transforming a function that takes multiple arguments into a sequence of functions that each take a single argument. In JavaScript, you can use currying to create new functions by partially applying the arguments of an existing function.

By using currying, we can create reusable functions that are more flexible and easier to compose with other functions.

Features of currying

- **Higher-order functions:** Currying relies on the concept of higher-order functions, where functions can accept other functions as arguments and return functions as values.
- **Closure: Currying in JavaScript** utilizes closures to capture the state of the outer function, allowing inner functions to access variables defined in their lexical scope even after the outer function has finished executing.
- **Flexibility:** Curried functions offer flexibility by allowing arguments to be passed incrementally, making them suitable for scenarios where certain arguments may be provided later.

27. Memoization

https://www.geeksforgeeks.org/javascript-memoization/
https://www.freecodecamp.org/news/memoization-in-javascript-and-react/
https://www.geeksforgeeks.org/how-to-write-a-simple-code-of-memoization-function-in-javascript/

Memoization is a technique used in programming to optimize the performance of functions that are called repeatedly with the same input. It involves caching the results of a function call so that subsequent calls with the same input can be retrieved from the cache instead of re-computing the result.

By using memoization, we can avoid redundant computations and improve the performance of our functions, especially when dealing with large or complex data sets.

28. Micro/macro tasks

https://www.geeksforgeeks.org/what-are-the-microtask-and-macrotask-within-an-event-loop-in-javascript/

In JavaScript, there are two types of tasks that are executed by the event loop: microtasks and macrotasks.

Microtasks are small asynchronous tasks that are executed immediately after the current synchronous task is finished. They are usually higher-priority tasks that need to be executed as soon as possible. Examples of microtasks include promises and process.nextTick() in Node.js.

Understanding the difference between microtasks and macrotasks is important for writing efficient and predictable asynchronous code in JavaScript.

29. SetTimeout v/s setInterval

https://www.geeksforgeeks.org/java-script-settimeout-setinterval-method/\
https://www.geeksforgeeks.org/difference-between-settimeout-setinterval/

setTimeout and setInterval are both functions in JavaScript that allow you to execute a block of code after a certain amount of time has passed. However, there are some differences between the two functions:

setTimeout is used to execute a function once after a specified delay. It takes two arguments: a callback function and a delay time in milliseconds. The callback function will be executed after the delay time has passed.

setInterval is used to repeatedly execute a function with a fixed time delay between each execution. It also takes two arguments: a callback function and a delay time in milliseconds. The callback function will be executed repeatedly after the delay time has passed.

30. ES6 Arrow Function

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
https://www.w3schools.com/js/js_arrow_function.asp
https://www.geeksforgeeks.org/es6-arrow-function/

https://www.geeksforgeeks.org/arrow-functions-in-javascript/

Arrow functions are introduced in ES6, which provides you a more accurate way to write the functions in JavaScript. They allow us to write smaller function syntax. Arrow functions make your code more readable and structured.

Arrow functions are anonymous functions (the functions without a name and not bound with an identifier). They don't return any value and can declare without the function keyword. Arrow functions cannot be used as the constructors. The context within the arrow functions is lexically or statically defined. They are also called as Lambda Functions in different languages.

Arrow functions do not include any prototype property, and they cannot be used with the new keyword.

31. Prototype

https://www.w3schools.com/js/js_object_prototypes.asp
https://www.geeksforgeeks.org/prototype-in-javascript/

Every object in JavaScript has a built-in property, which is called its prototype.

The prototype is an object that is associated with every functions and objects by default in JavaScript, where function's prototype property is accessible.

Prototype is used to provide additional property to all the objects created from a constructor function.

32. Advanmtage and Disadvantages of Javascript.

https://www.geeksforgeeks.org/advantages-and-disadvantages-of-javascript/
https://www.tutorialspoint.com/advantages-and-disadvantages-of-javascript

JavaScript is a powerful and popular programming language, but like any technology, it has its limitations and drawbacks. Here are some of the main disadvantages of JavaScript:

Browser Support: JavaScript is a client-side language, meaning it runs in the browser. While it is supported by all modern browsers, some older browsers may not support the latest features, which can cause compatibility issues.

Security: Because JavaScript code runs on the client side, it can be vulnerable to security threats such as cross-site scripting (XSS) attacks. Developers need to be careful when handling sensitive data and implement appropriate security measures to protect against these types of attacks.

Performance: JavaScript is an interpreted language, which means that it can be slower than compiled languages like C++ or Java. This can lead to performance issues, especially when working with large datasets or complex applications.

Debugging: Debugging JavaScript can be difficult, especially for beginners. The language has a lot of flexibility, which can make it challenging to find and fix bugs.

Lack of Type Checking: JavaScript is a dynamically-typed language, which means that variables are not assigned a specific data type. This can lead to errors when working with complex applications, as it is harder to catch type-related issues before they cause problems.

Code Maintainability: JavaScript can be harder to maintain than other programming languages due to its flexibility and loose structure. This can lead to code that is harder to read, debug, and modify over time.

Overall, while JavaScript is a powerful and popular language, developers need to be aware of its limitations and work to mitigate its disadvantages through proper coding practices and implementation of best practices.

33. Rest or Spread operator

https://www.w3schools.com/react/react_es6_spread.asp
https://www.w3schools.com/howto/howto_js_spread_operator.asp
https://www.geeksforgeeks.org/javascript-spread-operator/

The rest operator (...) in JavaScript is used within function parameters to gather all remaining arguments into a single array.

spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.

34. == and === operators

https://www.tutorialspoint.com/difference-between-and-operator-in-javascript

35. optimize JavaScript performance in a web application

36. What are generators and how do they differ from regular functions? Provide an example.

Generators are functions that can be paused and resumed, allowing for asynchronous programming. They are defined with an asterisk (*) and use the yield keyword to yield values.

37. How does garbage collection work in JavaScript?

A: JavaScript uses automatic garbage collection to manage memory. The most common form of garbage collection is mark-and-sweep. The garbage collector finds and marks all reachable objects and then sweeps the heap, collecting all unmarked objects and reclaiming their memory.

38. Explain the concept of promises all and promise chaining.

39. What is a service worker and how does it work?

A: A service worker is a script that runs in the background, separate from the web page, allowing for features like background sync, push notifications, and offline capabilities. It intercepts network requests, serving responses from the cache if available, and updating the cache with new responses.