1.Closures

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

2.Javascript Data Types
JavaScript provides different data types to hold different types of values. There are two types of data types in JavaScript.

1.Primitive data type
2.Non-primitive (reference) data type

1.Under Primitive data type we have

String = represents sequence of characters e.g. "hello"
Number=represents numeric values e.g. 100
Boolean=represents boolean value either false or true
Undefined=represents undefined value
Null=represents null means no value at all

2.Under Non-primitive (reference) data type we have

Object=represents instance through which we can access members
Array=represents group of similar values
RegExp=represents regular expression

3.Is JavaScript dynamic typed language?
JavaScript is a dynamic type language, means we don't need to specify type of the variable because it is dynamically used by JavaScript engine. we need to use var here to specify the data type. It can hold any type of values such as numbers, strings etc.

For example:
var a=40;//holding number
var b="Rahul";//holding string

4.Scope & scope chain in JavaScript
Scope refers to the area in which variables, functions, and objects are accessible.
JavaScript has three main types of scope:

Global Scope
Function Scope

Block Scope

Global Scope=Variables declared outside any function or block are in the global scope.They are accessible from anywhere in the code.

Function Scope=Variables declared inside a function are in the function scope. They are accessible only within that function.

Block Scope=Variables declared with let or const inside a block (a pair of curly braces {}) are in the block scope. They are accessible only within that block.ex:in a for loop or if conditions

scope schain:
When a variable is used, JavaScript starts looking for it in the current scope. If it is not found, it moves up to the next outer scope, continuing this process until it reaches the global scope.

5.Difference between var,let and const?

In JavaScript, var, let, and const are used for variable declaration.
var:
Variables declared with var are function scoped,in which they are declared in. If declared outside any function, they are global.
Variables declared with var are hoisted to the top of their scope and initialized with undefined.
Variables declared with var can be re-declared within the same scope

Let:
Variables declared with let are block({}) scoped
Variables declared with let are hoisted to the top of their block but are not initialized.They are in a "temporal dead zone" until the declaration happened.
Variables declared with let cannot be re-declared within the same scope.

Const:
Variables declared with let are block({}) scoped
Variables declared with let are hoisted to the top of their block but are not initialized.They are in a "temporal dead zone" until the declaration happened.
Variables declared with const cannot be re-declared or re-assigned within the same scope. They must be initialized at the time of declaration.

6.Hoisting
Hoisting is a JavaScript mechanism where variable and function declarations are moved to the top of their containing scope during the compilation phase. variables and functions can be used before they are declared in the code.

Variables declared with var are hoisted to the top of their function or global scope and initialized

with undefined.
incase of Variables declared with let and const are also hoisted to the top of their block scope, but they are not initialized. They are in a temporal dead zone from the start of the block until the declaration is happened.

7.Arrow functions

Arrow functions, introduced in ES6. provide a shorter syntax for writing function expressions. They are useful for writing concise one-liners and for maintaining the this context within methods.

Differences Between Arrow Functions and Regular Functions
Arrow functions have a more concise syntax, especially for simple one-liners.
Arrow functions do not have their own this context. Instead, they will inherit this from the enclosing lexical context.
Arrow functions do not have their own arguments object. If you need access to the arguments object, you should use a regular function.
Arrow functions cannot be used as constructors and will throw an error if used with the new keyword.

8.rest parameter and spread operator

Rest Parameter
rest parameter allows a function to accept an indefinite number of arguments as an array. This is useful for functions that need to handle a variable number of arguments.

Spread Operator
The spread operator allows an iterable (like an array or a string) to be expanded in places where zero or more arguments or elements are expected. It is commonly used for array and object manipulation, function calls.

9.Promise
Promise is an object.
Promise constructor takes an argument that callback function. This callback function takes two arguments resolve, and rejected.
promises are useful for handling asynchronus operations and callback hell issues.
Resolve: When the promise is executed successfully, the resolve argument is invoked, which provides the result.
Reject: When the promise is rejected, the reject argument is invoked, which results in an error.

10.async and await

async and await keywords provide a more straightforward syntax for working with promises,

making asynchronous code easier to read and write. They are syntactic sugar over promises, allowing you to write asynchronous code that looks synchronous.
async function is a function that returns a promise. the async keyword before the function definition.
await operator is used inside an async function to pause the execution of the function until the promise is resolved. It makes the function wait for the promise's resolution or rejection

When to Use Promises
Promises are suitable when we need to handle multiple asynchronous operations and chain them together, especially when we need to handle multiple levels of nested callbacks or when using APIs that return promises.

When to Use async/await
async/await is used when we want our asynchronous code to look and behave more like synchronous code. It is especially useful for reducing the complexity of promise chains and making the code more readable and maintainable.

Advantages of async/await
Readability: async/await makes asynchronous code look like synchronous code, improving readability.
Error Handling: Easier and more intuitive error handling using try/catch.
Debugging: Easier to debug due to the linear code flow.

11.Synchronus and asynchronus difference?

Synchronous Operations

Synchronous operations are tasks that are executed sequentially.
Each operation waits for the previous one to complete before executing.
Each operation blocks the execution of the next one until it completes.

Asynchronous Operations

Asynchronous operations allow tasks to be executed without waiting for other operations to complete.
Asynchronous Operations can proceed without waiting for other operations to finish.
Multiple operations can be in progress at the same time.
Suitable for tasks that involve waiting, like network requests or user inputs.

12.Is JavaScript is single-threaded?

Yes, JavaScript is single-threaded, meaning it has a single call stack, which means it can only do one thing at a time. However, JavaScript can handle asynchronous operations, which gives the multi-threading.

13.Event Loop?

Event loop lies in the javascript run time environment.which is mainly responsible for processing the code and exceuting the code based on the prority wise.first it will executes the normal synchronization code, and then micro task queues will be executed under this we have promises and then call back queues will be executes under this we have setTimeouts and setIntervals are comes.

14.shallow copy and deep copy?

Shallow Copy

A shallow copy of an object is a copy of the original object where only the first level of the object is copied. If the original object contains references to other objects (nested objects), the shallow copy will not create new instances of those nested objects; instead, it will copy the references to those objects. This means that changes to nested objects will affect both the original and the copied object.
we can achieve shallow copy Using Object.assign() and spread operator(...).

Deep Copy

A deep copy of an object copies all levels of the original object, creating new instances of all nested objects. the copied object is completely independent of the original object, if changes to nested objects in the copy will not affect the original object
we can achieve deep copy Using JSON.parse and JSON.stringify and Using lodash library and also Using the structuredClone Method

15.Object Creation ways in javascript?

1. Object Literals
Object literals are the way to create objects in JavaScript.

2. Using the new Object() Syntax
This is an older way to create objects, which is less used.

3. Constructor Functions
Constructor functions allow you to create multiple objects with the same properties and methods.

4. ES6 Classes
Classes in ES6 provide a cleaner syntax for constructor functions and prototype inheritance.

5. Object.create()
This method creates a new object with the specified prototype object and properties.

16.diference between === and == in javscript?

== Equality Operator compares values only, if both are equal then it returns true or else it returns false.
=== Strict Equality Operator compares values and also its data types, if both are equal then it returns true or else it returns false.

17.IIFE in javscript?

An Immediately Invoked Function Expression (IIFE) is a JavaScript function that runs as soon as it is defined.
In larger applications, using IIFEs can prevent variable name collisions by encapsulating the code in a local scope

18.Callback Function in JavaScript

A callback function is a function that is passed as an argument to another function and is executed after some operation has been completed.
When we pass a function as an argument to another function, we are passing the function definition and not the function's result. The function that receives the callback can execute it at the appropriate time.

19.callback hell issue in javascript?

whenever we try to define multiple nested callback functions, it lead to code that is difficult to read, maintain, and debug. This issue comes in asynchronous code, especially when dealing with tasks such as handling API requests, database operations. The deep nesting of callbacks can make the code look like a pyramid, that term we can call it as "callback hell."

Problems with Callback Hell

The code becomes difficult to read and understand due to the deep nesting of callbacks, making it harder to maintain and debug.
Managing errors in callback hell can be challenging. Handling errors at each level of nesting can lead to verbose and messy code.

20.this in javascript and explain?

In JavaScript, the this keyword is a identifier that refers to the context in which a function is called. The context can vary depending on how the function is invoked,

In the global scope (outside of any function), this refers to the global object
In a regular function, this refers to the global object or undefined (in strict mode).
In a method (a function inside an object), this refers to the object.
In a constructor function (used with new keyword), this refers to the newly created instance.

21.Prototypes & prototype chain in javacript?

Prototypes:
A prototype in JavaScript is an object from which other objects inherit properties. Every JavaScript object has a prototype, and these prototypes can have prototypes, form a chain we can call it as the prototype chain.When we create an object, JavaScript assigns it a prototype. If we create an object using a constructor function, the prototype of the newly created object will be the prototype property of the constructor function.

Prototype chain:
The prototype chain is the mechanism by which JavaScript objects inherit properties from other objects. When you access a property or method on an object, JavaScript will first look at the object itself. If the property is not found, it will then look at the object's prototype, and continue up the chain until it either finds the property or reaches the end of the chain (null).

22.DOM

It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects, making it possible to interact with the document.
The Document Object Model (DOM) is essential in web development for several reasons:

Dynamic Web Pages: It allows you to create dynamic web pages. It enables the JavaScript to access and manipulate page content, structure, and style dynamically which gives interactive and responsive web experiences, such as updating content without reloading the entire page or responding to user actions instantly.

Interactivity: With the DOM, you can respond to user actions (like clicks, inputs, or scrolls) and modify the web page accordingly.

Content Updates: When you want to update the content without refreshing the entire page, the DOM enables targeted changes making the web applications more efficient and user-friendly.

23.forEach
forEach is used to execute a function on each element in an array. It does not return a new array, and it is typically used when you want to perform side effects rather than transformations.

Use Case:
Iterating over an array to perform an action for each element, such as logging or modifying elements in place.

24.map
map creates a new array by applying a function to each element of the original array. It is used when you want to transform elements in an array and return a new array of the same length.

Use Case:
Transforming each element in an array to produce a new array.

25.filter
filter creates a new array with all elements that pass the test implemented by the provided function. It is used when you want to select a subset of elements from an array.

Use Case:
Selecting elements from an array that meet certain conditions.

26.reduce
reduce applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single value. It is used for summing, concatenating, or any operation where a single output value is produced from an array.

Use Case:
Aggregating array elements into a single value.

27.array methods

push: Adds one or more elements to the end of an array.
pop: Removes the last element from an array.
shift: Removes the first element from an array.
unshift: Adds one or more elements to the beginning of an array.
splice: Adds/removes elements from an array at a specific index.
slice: Returns a shallow copy of a portion of an array.
concat: Merges two or more arrays.

28.object methods

Object.assign(): Use to copy properties from one or more source objects to a target object.
Object.create(): Use to create a new object with a specified prototype.
Object.keys(): Use to get an array of an object's own property names.
Object.values(): Use to get an array of an object's own property values.

Object.entries(): Use to get an array of an object's own [key, value] pairs.
Object.freeze(): Use to freeze an object, making it immutable.
Object.seal(): Use to seal an object, preventing the addition or deletion of properties but allowing modifications to existing properties.

29.string methods

toString(): Convert different types to a string representation.
toLowerCase(): Convert a string to lowercase.
toUpperCase(): Convert a string to uppercase.
charAt(): Get the character at a specific index.
indexOf(): Find the index of a substring.
substring(): Extract part of a string between two indices.
slice(): Extract a section of a string.
split(): Split a string into an array based on a delimiter.
replace(): Replace part of a string with another string.
trim(): Remove whitespace from both ends of a string.
includes(): Check if a string contains another string.
startsWith(): Check if a string starts with another string.
endsWith(): Check if a string ends with another string.

30.event bubbling and event capturing

Event bubbling and event capturing are two phases of event propagation in the DOM (Document Object Model) in JavaScript.

Event Bubbling: Events propagate from the target element up to the root of the DOM tree.
Event Capturing: Events propagate from the root of the DOM tree down to the target element.
stopPropagation(): Prevents further propagation of the event in the capturing and bubbling phases.

31.Event delegation

It is a technique in JavaScript where you use a single event listener to manage all events of a particular type for child elements. Instead of attaching individual event listeners to each child element, you attach a single event listener to a common parent element. This technique leverages event bubbling to handle events more efficiently, especially when dealing with a large number of elements or dynamically added elements.

How It Works: Attach the event listener to a parent element, use event.target to identify the actual target element, and handle the event accordingly.

Use Cases: Handling events for dynamically added elements, improving performance by reducing the number of event listeners, and simplifying code maintenance.

32.debounce and throttle in javacript and use cases

Debounce
Definition: Delays execution of the function until after a certain period has passed since the last time the function was invoked.

Use Case: User actions that happen frequently and can result in performance issues if handled in real-time (e.g., search input fields, form validation).

Example: If a user types in a search field, an API call to fetch results is made only after the user stops typing for a specified duration.

Throttle
Definition: Ensures a function is executed at most once in a specified interval, regardless of how many times the event occurs during that interval.

Use Case: Continuous events that can fire many times in a short period (e.g., scroll events, window resize events).

Example: If a user is resizing a window, the resize handler function is called at most once per second, regardless of how many times the resize event fires.

33.Local Storage, Session storage & Cookie

Local Storage
Scope: Data stored in local storage persists even after the browser is closed and is accessible across browser sessions.

Storage Limit: Typically, the storage limit is around 5MB per domain.

Use Cases:Storing user preferences/settings that should persist across sessions.Caching static data or resources to improve performance.Storing non-sensitive data that doesn't expire.

Session Storage

Scope: Data stored in session storage is available only for the duration of the browser session. It is cleared when the browser tab is closed or the session ends.

Storage Limit: Similar to local storage, around 5MB per domain.

Use Cases:Storing temporary data that is needed for a single browser session.Handling data that should be cleared once the user closes the browser.

Cookies

Scope: Cookies are small pieces of data stored in the user's browser. They can have an expiry date or be session cookies that expire when the browser is closed.

Storage Limit: Each cookie has a size limit of about 4KB, and the total number of cookies per domain is limited (usually around 20-50).

Use Cases:Maintaining user authentication and session management (session cookies).Tracking user behavior, preferences, or site analytics (persistent cookies).Storing data that needs to be sent to the server with each HTTP request.