**JavaScript Concepts (Interview)**

1. Learn how **event loop** works.
2. Learn how to use **Promise** and all its methods.

**Promise.all()**

**Promise.any()**

**Promise.race()**

**Promise.allSettled()**

1. Learn about **async** and **await.**
2. Learn about **call, apply, bind.**
3. Learn about **Hoisting.**
4. Learn about **Memoization.**
5. Learn about **Closures and Currying**.
6. Learn about **Callbacks.**
7. Learn about **rest** and **spread** operator.
8. Learn about **array.map(), array.filter(), array.reduce()** methods.
9. Learn about **setTimeout()** and **setInterval()** methods.
10. Learn the difference between **var, let, const.**
11. Learn about **this** keyword.
12. Learn **object-oriented** concepts.
13. Learn about **for, for..in, for..of, foreach** methods.
14. Learn about **get** and **set.**
15. Learn about **inheritance** and **proptotype** chain.
16. Learn about **Objects** and **Maps.**
17. Learn about **Temporal Dead Zone.**
18. Learn about **strict mode**.
19. Learn about **WeakMap** and **WeakSet.**
20. Learn about **shallow copy** and **deep copy.**
21. Practice **type coercion.**
22. Learn about **Event Emitters.**
23. Learn about **Promise vs Observable.**
24. Learn about **const vs Object.freeze().**
25. Learn about **arrow functions vs Regular functions** and when should we use them**.**
26. Learn about **async await vs generators.**
27. **Debounce** and **throttling.**
28. Immediately Invoked Function**.**
29. **Class and Objects**
30. **Http methods.**
31. **Rest Api**
32. **Cache**
33. **Cookies**
34. **Browser Storage**
35. **Lazy Loading**
36. **Event Bubbling**
37. **What does Babel do?**
38. **Learn about latest file structures.**
39. **Type vs Interface**
40. **What does webpack do?**
41. **Code splitting**
42. **Tree Shaking**
43. **ES6 new features**
44. **XSS**
45. **CORS**
46. **CSRF**

1)

**Call Stack:** The JavaScript runtime maintains a data structure called the call stack. The call stack keeps track of the execution context of currently running functions. When a function is called, a new entry (a frame) is pushed onto the call stack, and when the function returns, its frame is popped off the stack.

**Callback Queue:** JavaScript uses an event-driven, non-blocking I/O model. When an asynchronous operation, such as a timer, network request, or user interaction, is initiated, the operation is scheduled to run in the background. When the operation is completed, it is placed in a data structure known as the callback queue (or task queue).

**Event Loop:** The event loop is a continuous process that constantly checks two things: the call stack and the callback queue. It checks if the call stack is empty. If the call stack is empty, the event loop looks into the callback queue for any completed tasks.

If there are tasks in the callback queue, the event loop takes the first task and pushes it onto the call stack, effectively executing the associated callback function. If the callback function generates new asynchronous tasks, they are scheduled to run in the background, and their callbacks are placed in the callback queue.

**Concurrency:** The event loop's design allows JavaScript to handle multiple asynchronous operations simultaneously without blocking the main execution thread. This concurrency model ensures that your application can remain responsive, even when dealing with time-consuming tasks like I/O operations or animations.

2)

**Promises** are JavaScript objects that are used to handle asynchronous code execution. It has three states (resolved, rejected, pending).

**Promise.all()** takes multiple Promises and waits for all of them to complete or till one gets rejected.

**Promise.any()** takes multiple Promises and waits for only one of them to complete and that promise is fulfilled.

**Promise.race()** tales multiple Promises and it allows racing of Promises of which gets fulfilled first.

**Promise.allSettled()** waits for all Promises to complete whether they are fulfilled or rejected.

3)

**Async** keyword used behind function makes it an asynchronous function and **await** keyword is used inside async function where it waits for a code to be completed and then moves to the next line. Try catch blocks are used for error handling and async await are used to handle Promises instead of Promise chaining.

4)

The **call()** method is a special built-in function in JavaScript that allows you to invoke a function with a special **this** value and arguments provided individually.

The **apply()** method is same as **call()** method but instead of passing arguments individually it takes arguments in an array.

The **bind()** method is used to create a new function, when invoked has a special **this** value and prepends all the arguments to the new function when invoked. It is not invoked immediately, you should call it as a function.

5)

**Hoisting** is a process in which variables and functions are moved to the top during the compilation phase. In variable hoisting, the variable should have a keyword var otherwise we get an undeclared error.

7)

**Closures** is a concept where it retains access to its surrounding lexical scope variables even after the scope has finished executing. Used for data encapsulation.

Example: Parent and child function.

**Currying** is a concept where multiple arguments are taken as single arguments in a chain of returning functions. This allows for partial function where only one argument is passed at a time.

8)

**Callbacks** are regular JavaScript functions that are passed as arguments to another function. Advantages are non-blocking, async control flow.

Example: setTimeout(), array.map() etc.

9)

**Rest** operators are used to collect many elements or arguments into a single array. It uses … to represent rest operator. Rest operator must be the last parameter. It is also used to destructure arrays or objects.

**Spread** operators are used to spread elements of an array or properties of an object into an array or an object. It uses … to represent spread operator. It can be used with array, objects and strings.

10)

|  |  |  |
| --- | --- | --- |
| **map** | **filter** | **reduce** |
| Returns a new array | Returns a new array | Returns a single value |
| Length always equal to original array | Length always less than original array | Returns a single value. It takes a default value. |

11)

Practice use of setTimeout() and setInterval().

12)

|  |  |  |
| --- | --- | --- |
| **var** | **let** | **const** |
| It is function scoped and available only inside that function. | It is block scoped and is only available inside that block. | It is block scoped and is only available inside that block. |
| Can be hoisted. | Cannot be hoisted. | Cannot be hoisted. |
| Can be both reassigned and re declared. | Can be reassigned bot not re declared. | Cannot be reassigned and re declared. |

13)

**“this”** keyword is a special keyword which refres to the current context of the object in which it is used.

14)

**Inheritance:** The way in which we inherit the variables and function in the parent class and use them in our child class.

Use extend keyword for child class and use define the variable which you want to inherit in the super inside constructor.

Example: Use Person(Parent class) and Student(Child class) example.

**Encapsulation:** Process of bundling data and functions into a single object. It helps to hide the implementation of the object.

This can be achieved using \_variable to make it a private variable so that it cannot be accessed from outside. If we don’t use \_ then it is considered public variable

Example: Counter where \_count is a private variable.

**Polymorphism:** Process where you can access objects of different types through the same interface.

Create a base class and a child class which have the same function. Create a function which accepts an interface and call the function.

Example: Create a class called Shape and create 2 child classes Circle and Square. Create a function to display the area which accepts shape as parameter.

**Data Abstraction:** Process of hiding the data.

Create a base class and child class. If you try to run a function in base class which is also present in child class then throw an error.

Example: Create a base class called Shape and child class called circle. Let them both have same function to display area. Throw error when you try to display area using Shape class.

16)

Using **get** and **set** methods allows you to control the behavior of reading and writing properties in objects. They are commonly used to add additional logic or validation when accessing or modifying object properties.

18)

|  |  |
| --- | --- |
| **Objects** | **Maps** |
| In Objects keys are always strings and other data types are converted to strings. | Maps can have any data types as key. |
| Objects mostly maintain order in which they are inserted(ES6 onwards). | Maps always maintain the order in which they are inserted. |
| Objects doesn’t have built in property to get the size. | Maps have built in property to get the size. |
| Objects are slightly below in terms of performance when compared to maps. | Maps have an advantage over objects in terms of insertion and deletion. |

19)

If we use let keyword, then that variables are not hoisted. If we try to access the variable and then declare the variable, then that variable is said to be in **Temporal Dead Zone**.

20) To enable **strict mode** we need to have “use strict;” at the top of the script.

1. All variables must be declared before using them.
2. It doesn’t create global variables so when you try to assign a variable it gives reference error.
3. Certain keywords cannot be used such as eval, arguments, implements etc.
4. Octal literal cannot be used. They are treated as decimal values.
5. Assigning to this that is not inside a constructor is not allowed.
6. Cannot use delete operator.

22)

Common ways to create a shallow copy in JavaScript:

Object.assign

let original = { a: 1, b: { c: 2 } };

let copy = Object.assign({}, original);

Spread Operator (…):

let original = { a: 1, b: { c: 2 } };

let copy = { ...original };

Similar to Object.assign, the spread operator only copies the outermost object.

Deep Copy in JavaScript  
  
let original = { a: 1, b: { c: 2 } };

let copy = JSON.parse(JSON.stringify(original));

23)

1. Practice **type coercion.**

**Type Coercion in JavaScript**

**Type coercion** is the process by which JavaScript automatically converts one data type to another. This happens often in expressions involving mixed types, such as when a string is added to a number, or when logical operations involve non-boolean values.

There are two types of type coercion:

1. **Implicit Coercion**: Automatic type conversion by JavaScript.
2. **Explicit Coercion**: Manual type conversion by the developer.

25) Observable:

**Core Components of an Observable**

* **Producer**: The part that emits the data. This is essentially what generates the stream of values, such as user events or an API call.
* **Observer**: The code that listens for data (emissions) and reacts to it. An observer is notified when new data is emitted by subscribing to the Observable.
* **Subscription**: When you subscribe to an Observable, a connection is created between the producer and the observer. The subscription can also be used to cancel the listening for future emissions.
* **Operators**: Functions that can transform or manipulate the data stream (e.g., filtering, mapping, throttling the data).
* **Complete/Error**: An Observable can finish either by completing (no more data is emitted) or by encountering an error.

**Observables vs Promises**

Observables are more powerful and flexible than **Promises** because:

* **Multiple values**: Observables can emit multiple values over time, while Promises handle only a single value.
* **Lazy execution**: An Observable starts emitting values only when subscribed to, whereas a Promise executes immediately.
* **Operators**: Observables come with a wide range of operators that allow for complex data transformation and manipulation, unlike Promises.
* **Cancelation**: Observables can be canceled using the unsubscribe method, whereas canceling Promises requires extra work.

const { Observable } = require("rxjs");

const observe = new Observable((subscriber) => {

setTimeout(() => {

subscriber.next("cat");

subscriber.complete();

}, 1000);

});

observe.subscribe((result) => {

console.log(result);

});

26)

Object.freeze() is a method in JavaScript that allows you to **freeze** an object, meaning the object becomes **immutable**.

Object.freeze(obj);

Deep freeze when there is nested objects since object.freeze wont freeze the nested objects

function deepFreeze(obj) {

// Retrieve the object's property names

const propNames = Object.getOwnPropertyNames(obj);

// Freeze properties before freezing the object itself

for (const name of propNames) {

const value = obj[name];

// If the value is an object, recursively freeze it

if (value && typeof value === "object") {

deepFreeze(value);

}

}

// Finally, freeze the original object

return Object.freeze(obj);

}

const bike = {

brand: "Yamaha",

features: {

color: "black",

engine: "500cc",

},

};

deepFreeze(bike);

// Now, even nested objects are frozen

bike.features.color = "red"; // No effect

console.log(bike.features.color); // Output: "black"

30)

Immediately invoked functions are invoked without calling them. This can be achieved by wrapping a function with parentheses and adding () at the end.

42)

Webpack is a powerful and widely used open-source **module-bundler** for JavaScript files and various assets(like CSS, images, fonts, etc.) into a single file or multiple optimized files that can be served to the browser efficiently.

Webpack takes all the files (JavaScript, CSS, images, etc.) that your app depends on and bundles them into a smaller number of files (usually one or a few) to improve load time and organization. Webpack uses a dependency graph to understand how different files depend on each other and bundles them accordingly.

Webpack doesn’t just handle JavaScript. Through the use of **loaders**, it can also process other types of files, such as:

* **CSS**: Convert CSS into JavaScript and bundle it.
* **Sass/SCSS**: Compile Sass or SCSS files into CSS.
* **Images**: Load image files and optimize them.
* **Babel**: Transpile modern ES6+ JavaScript to a version of JavaScript that is compatible with older browsers (via babel-loader).

**43)**

**Code Splitting:** Webpack allows for code splitting, which is the practice of splitting your JavaScript bundles into smaller chunks that can be loaded on demand. This is useful for optimizing load times, especially for large applications.

44)

**Tree Shaking**

Tree shaking is a feature in Webpack that removes unused or dead code from your final bundle, helping to keep your output files smaller and more efficient.

Webpack automatically applies tree shaking when used with ECMAScript modules (ES6+ import/export).

45)

New features of ES6

1. Introduction of block scope variable assignment: Let and const
2. Arrow function this here refers to the the enclosing object.
3. Rest and spread operators
4. Introduced support for modules import and export.
5. Promises
6. For…of loop

Security related questions

1. **Cross -site scripting (XSS)-**

XSS is a type of attack where malicious script are injected into the trusted website. Through this attacker can steal the sensitive data such as cookies, token or even perform other harmful actions. Basically, the attackers will trick the users to click the URL that contains a script which is reflected to the user’s browser and executed.  
To avoid XSS attack use content security policy(CSP)

1. which is a HTTP header that helps mitigate XSS by specifying which domains are allowed to execute scripts on our page.
2. Validate all the users input both on client side and server side
3. **CORS –(Cross-origin Resource sharing ):**

CORS is a security feature that restricts web pages from making requests to a different domain(cross-origin) than the one that served the web page. This is implements by browsers to prevent cross-site request Forgery attacks and protect user data.

1. Cross-site request Forgery(CSRF)

CSRF is a attack where an attacker tricks a user into executing actions they did not intent to perform, such as submitting a form or clicking a link that sents malicious requests on behalf of the user.  
we can prevent CSRF by using CSRF tokens in forms that require authentication

1. **Content Security Policy (CSP)**

CSP is a security feature that helps prevent XSS by specifying which domains the browser should consider as valid sources for scripts, styles, and other resources.

1. HTTPS encrypts the communication between the browser and the server, ensuring the integrity and confidentiality of data.

1. Learn how **event loop** works.
2. Learn how to use **Promise** and all its methods.

**Promise.all()**

**Promise.any()**

**Promise.race()**

**Promise.allSettled()**

1. Learn about **async** and **await.**
2. Learn about **call, apply, bind.**
3. Learn about **Hoisting.**
4. Learn about **Memoization.**
5. Learn about **Closures and Currying**.
6. Learn about **Callbacks.**
7. Learn about **rest** and **spread** operator.
8. Learn about **array.map(), array.filter(), array.reduce()** methods.
9. Learn about **setTimeout()** and **setInterval()** methods.
10. Learn the difference between **var, let, const.**
11. Learn about **this** keyword.
12. Learn **object-oriented** concepts.
13. Learn about **for, for..in, for..of, foreach** methods.
14. Learn about **get** and **set.**
15. Learn about **inheritance** and **proptotype** chain.
16. Learn about **Objects** and **Maps.**
17. Learn about **Temporal Dead Zone.**
18. Learn about **strict mode**.
19. Learn about **WeakMap** and **WeakSet.**
20. Learn about **shallow copy** and **deep copy**
21. Learn about **Event Emitters.**
22. Learn about **Promise vs Observable.**
23. Learn about **const vs Object.freeze().**
24. Learn about **arrow functions vs Regular functions** and when should we use them**.**
25. Learn about **async await vs generators.**
26. **Debounce** and **throttling.**
27. Immediately Invoked Function**.**
28. **Class and Objects**
29. **Http methods.**
30. **Rest Api**
31. **Cache**
32. **Cookies**
33. **Browser Storage**
34. **Lazy Loading**
35. **Event Bubbling**
36. **What does Babel do?**
37. **Learn about latest file structures.**
38. **Type vs Interface**
39. **What does webpack do?**

28)  
**generators** are special functions that can pause their execution and later resume it.

function\* greetting(name){

yield(“hello”);

yield(“how are you”);

yield(name)

}

Const greet = greeting(“Adam”);

Console.log(greet.next());