JAVASCRIPT INTERVIEW Q-A (js)

What is JavaScript?

JavaScript is a high-level programming language that is primarily used for creating interactive and dynamic elements on websites. It allows developers to add functionality, manipulate web page content, and respond to user actions. JavaScript can be executed on both the client-side (in the user's web browser) and the server-side (on web servers), making it a versatile language for web development. It is known for its wide adoption, flexibility, and the ability to integrate with HTML and CSS seamlessly.

Variables and Data Types

-> Declaring variables

-> Variable scope

-> Primitive data types (string, number, boolean, null, undefined)

-> Complex data types (object, array, function)

-> Type conversion

Operators and Expressions

-> Arithmetic operators

-> Comparison operators

-> Logical operators

-> Assignment operators

-> Ternary operator

-> binary Operator

Control Flow and Conditional Statements

-> if...else statement

-> switch statement

-> ternary operator (conditional expression)

Loops and Iteration

-> for loop

-> while loop

-> do...while loop

-> for...each loop

-> loop control statements (break, continue)

Functions

-> Declaring and invoking functions

-> Function parameters and arguments

-> Returning values from functions

-> Function expressions and function hoisting

-> Anonymous functions and arrow functions

Arrays

-> Creating and manipulating arrays

-> Accessing array elements

-> Array methods (push, pop, shift, unshift, splice, slice, etc.)

-> Iterating over arrays (forEach, map, filter, reduce, etc.)

Objects

-> Creating objects

-> Object properties and methods

-> Accessing and modifying object properties

-> Object send in backend

Classes

-> Class syntax and declaration

-> Constructor and instance methods

-> Static methods

-> Inheritance

-> Abstraction

-> Getters and setters

Error Handling

-> try...catch statement

-> Throwing and handling errors

Strings

-> String manipulation and concatenation

-> String methods (length, indexOf, slice, substring, replace, etc.)

JavaScript Object Notation (JSON)

-> JSON syntax and data types

-> Parsing and stringifying JSON

Document Object Model (DOM)

-> **Introduction to DOM** : DOM provides a way to access and manipulate the content, structure, and style of a web page dynamically. It allows JavaScript to interact with the HTML elements on a web page, such as accessing or modifying their properties, adding or removing elements, and responding to user events.

-> Selecting Elements: You can select elements in the DOM using various methods, such as **getElementById**, **getElementsByClassName**, **getElementsByTagName**, or more advanced selectors like **querySelector** and **querySelectorAll**.

Browser Object Model (BOM)

-> Bom Introduction :- BOM stands for Browser Object Model in JavaScript. It is a set of objects provided by web browsers that represent the browser window or tab, and allows JavaScript code to interact with and manipulate various aspects of the browser environment.

-> window: The global object representing the browser window. It provides methods to manipulate the window (e.g., resize, close), access the document loaded in the window, and handle events.

-> **navigator**: Provides information about the browser and its capabilities, such as the browser name, version, and platform. It allows you to detect specific features supported by the browser.

-> **location**: Represents the URL of the current web page and provides methods to manipulate and navigate the browser to different URLs. It enables you to retrieve information about the current URL, modify it, or redirect the user to a new page.

-> **history**: Allows you to manipulate the browser's session history. You can go back or forward in the user's browsing history, navigate to a specific URL in the history, or even modify the history by adding or replacing entries.

-> screen: Represents the user's screen and provides information about its size, color depth, and orientation. It is useful for adapting the content or layout based on the available screen space.

Events

-> Event handling and event listeners

-> Mouse events (click, hover, etc.)

-> Keyboard events (keypress, keydown, etc.)

-> Form events (submit, input, etc.)

Asynchronous JavaScript

-> Callback functions

-> Promises

-> Async/await

-> Fetch API for making HTTP requests

Debugging and Error Handling

-> Debugging techniques and tools

-> Console methods (log, warn, error, etc.)

-> Handling and logging errors

1) Variables: Understanding variable declaration, assignment, and scoping in JavaScript.

2) Data Types: Basic data types like strings, numbers, booleans, arrays, date-time and objects.

3) Operators: Arithmetic, assignment, comparison, logical, and bitwise operators.

4) Control Flow: Conditional statements (if-else, switch), loops (for, while), and error handling (try-catch).

5) Functions: Declaring and using functions, parameters, return values, and function expressions.

6) Arrays: Manipulating arrays, accessing elements, adding/removing items, iterating over arrays.

7) Math & Objects: Creating Some In-Buil Functions Like : (map, filter, ceil, sqrt, pow) Creating objects, defining properties and methods, object-oriented programming concepts.

8) Prototypes and Classes: Understanding prototypes, constructor functions, and JavaScript classes.

9) Scope and Closures: Lexical scope, closure concepts, and practical uses.

10) DOM Manipulation: Interacting with HTML elements using JavaScript for dynamic web content.

11) Events: Handling user interactions and responding to events in the browser.

12) Asynchronous Programming: Working with asynchronous operations, callbacks, promises, and async/await.

13) Modules: Organizing code into reusable modules using the CommonJS or ES modules syntax.

14) JSON: Working with JSON data, parsing and serializing JSON objects.

15) Error Handling: Handling and throwing exceptions, debugging techniques.

16) Regular Expressions: Patterns for matching and manipulating text.

Example : const sentence = 'The quick brown fox jumps over the lazy dog.';

```
const pattern = /quick/;
const result = pattern.test(sentence);
```

17) ES6+ Features: New features introduced in ECMAScript 6 and later versions, such as arrow functions, template literals, destructuring, spread syntax, etc.

18) Browser APIs: Accessing browser-specific features like local storage, geolocation, Fetch API, etc.

19) Testing and Debugging: Techniques for testing JavaScript code and debugging using browser developer tools.

20) Form And Email Validation: Form and email validation refers to the process of verifying the correctness and integrity of user-inputted form data, particularly email addresses, to ensure they meet specific criteria or patterns.

JavaScript Interview Questions

Basic Javascript Interview Quetion

1) Explain Hoisting in javascript.

Hoisting is the default behaviour of javascript where all the variable and function declarations are moved on top.

Ex:-

```
console.log(x); // Output: undefined

var x = 10;


sayHello(); // Output: "Hello"

function sayHello() {
  console.log("Hello");
}
```

2) Difference between " == " and " === " operators.

Both are comparison operators. The difference between both the operators is that "==" is used to compare values whereas, " === " is used to compare both values and types.

3) Difference between var and let keyword in javascript.

The keyword **'Var'** has a function scope. Anywhere in the function, the variable specified using **var keyword** is accessible but the 'let' keyword is limited to the block in which it is declared.

```
function greet() {
    let a = 'hello';
    if(a == 'hello'){
        let b = 'world';
        console.log(a + ' ' + b);
    }
 // variable b cannot be used here if variable b is declare
// by var keyword then we can use.
    console.log(a + ' ' + b); // error showing bcz let b
scope only if condition
}
```

Output

```
hello world
console.log(a + ' ' + b); // error
ReferenceError: b is not defined
```

4) Explain Implicit Type Coercion in javascript.

Type Coercion refers to the process of automatic or implicit conversion of values from one data type to another. This includes conversion from Number to String, String to Number, Boolean to Number etc. when different types of operators are applied to the values.

Ex

String to number

```
var x = 10 + '20';

console.log(x);
```

Output

```
1020
```

Number to string

```
var w = 10 - '5';

var x = 10 * '5';

console.log(w);

console.log(x);
```

Output

```
5

50
```

5) Is javascript a statically typed or a dynamically typed language?

JavaScript is a dynamically typed language. In a dynamically typed language, the type of a variable is checked during run-time in contrast to a statically typed language, where the type of a variable is checked during compile-time.

Ex :

```
Var a = ‘nofal’

a = 34;

// change variable datatype at run time
```

6) What is NaN property in JavaScript?

NaN property represents the “Not-a-Number” value. It indicates a value that is not a legal number.

7) Explain passed by value and passed by reference in short

"Pass by value" means that a copy of the value is created and passed to the function. When we call

function with argument then it is copy in duplicate variable which in store in function parameter.

"pass by reference" means that the memory address of the value is passed to the function. This allows the function to directly access and modify the original value. Any changes made to the value inside the function will be reflected in the original value outside of it.

8) What is an Immediately Invoked Function Expression (IIFE) in JavaScript?

An Immediately Invoked Function Expression (IIFE) in JavaScript, often abbreviated as IIFE, is a function that is executed immediately after it is defined.

The structure of an IIFE involves wrapping the function expression within parentheses

When we declare IIFE function then it is automatically call itself.

Ex

```
(function () {
  // Code to be executed immediately
})();  // () it is automatically called (IIFE function)
```

9) What do you mean by strict mode in javascript and characteristics of javascript strict-mode?

Strict mode changes previously accepted "bad syntax" into real errors.

As an example, in normal JavaScript, without declare datatype (var, let, const) a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable.

Ex

```
'use strict';
x = 10; // global variable can’t declare without datatype
(function foo() {
  'use strict';
  var y = 20;
  delete y; // Throws a SyntaxError
})
```

Characteristics of strict mode in javascript

- Duplicate arguments are not allowed by developers.

- In strict mode, you won't be able to use the JavaScript keyword as a parameter or function name.
- The 'use strict' keyword is used to define strict mode at the start of the script. Strict mode is supported by all browsers.
- Engineers will not be allowed to create global variables in 'Strict Mode.

10) Explain Higher Order Functions in javascript.

Higher Order Functions in JavaScript are functions that can accept other functions as arguments or return functions as their results or values

Ex

```
function multiplier(factor) {
  return function (number) {
    return `number = ${number} , factor = ${factor}`;
  };
}
var double = multiplier(2);
console.log(double(5)); // number = 5 , factor = 2
```

11) Explain “this” keyword.

In JavaScript, the **this** keyword refers to the object that is currently executing the code. its value depends on how a function is called or how an object is accessed.

Ex

```
const person = {
  name: "John",
  greet: function() {
      console.log(`Hello, my name is ${this.name}.`);
  }};
person.greet(); // Output: "Hello, my name is John."
```

12) What is Anonymous functions in JavaScript

Anonymous functions in JavaScript are functions that are defined without a name. They can be created as function expressions or as arguments to other functions.

IIFE is Also Anonymous Funtion

Ex

```
const add = function(a, b) {
  return a + b;
};
```

```
const result = add(2, 3); // Output:
```

13) Explain call(), apply() and, bind() methods.

call(): The call() method is used to invoke a function with a specified this value and individual arguments passed as arguments. It accepts the this value as the first argument, followed by the function arguments.

Ex

```
function greet() {
  console.log(`Hello, ${this.name}!`);
}
const person = {
  name: 'John'
};
greet.call(person); // Output: Hello, John!
```

apply(): Similar to call(), the apply() method invokes a function with a specified this value, but it accepts arguments as an array-like object or an actual array.

```
function greet(greeting) {
```

```
  console.log(`${greeting}, ${this.name}!`);
}
const person = {
  name: 'John'
};
greet.apply(person, ['Hello']); // Output: Hello, John!
```

bind(): The bind() method creates a new function with a specified this value and initial arguments. It returns a new function that, when called, has its this value set to the provided value.

```
function greet() {
  console.log(`Hello, ${this.name}!`);
}
const person = {
  name: 'John'
};
const greetPerson = greet.bind(person);
greetPerson(); // Output: Hello, John!
```

14) What is the difference between exec () and test () methods in javascript?

exec(): The exec() method is used to execute a search for a substring in a string. It returns an array containing information about the match substring like:- substring, index,input, group or if no match is found then return **null**.

Ex:

```
const regex = /hello/;

const str = 'hello, world!';


const result = regex.exec(str);

console.log(result); // output:- [ 'hello', index: 0, input: 'hello, world!', groups: undefined ]
```

test(): The test() method is used to test a match substring exists in a string. It returns a boolean value (true or false) if substring find then return true else return false

```
const regex = /hello/;
```

```
const str = 'hello, world!';


const result = regex.test(str);

console.log(result); // true
```

15) What is currying in JavaScript?

Currying is a functional programming technique which transforms a function that takes multiple arguments into a sequence of nesting functions that each take only a single argument.

Ex

```
function add(x) {
  return function(y) {
    return x + y;
  };
}
// fn call method 1
const add5 = add(5);
console.log(add5(3)); // Output: 8
```

```
// fn call method 2

console.log(add(5)(3)); // output 8
```

16) What are some advantages of using External JavaScript?

External JavaScript is the JavaScript Code (script) written in a separate file with the extension.js, and then we link that file inside the <head> or <body> element of the HTML file where the code is to be placed.

Some advantages of external javascript are

- It allows web designers and developers to collaborate on HTML and javascript files.
- We can reuse the code.
- Code readability is simple in external javascript

17) Explain Scope and Scope Chain in javascript.

Scope in JS determines the accessibility of variables and functions at various parts of one's code

In general terms, the scope will let us know at a given part of code, what are variables and functions we can or cannot access.

There are three types of scopes in JS:

- Global Scope // all program scope
- Local or Function Scope // function scope
- Block Scope // loop and other block scope

18) Explain Closures in JavaScript.

Closures in JavaScript are a powerful concept that allows functions to remember and access variables from the parent (outer) function, even after the outer function has finished executing.

Ex

```
function outerFunction() {
  var outerVariable = 'I am outside!';
  function innerFunction() {
    console.log(outerVariable);
  }
  return innerFunction;
}
var closure = outerFunction();
closure(); // Output: "I am outside!"
```

19) Mention some advantages of javascript.

- Javascript is executed on the client-side as well as server-side also. There are a variety of Frontend Frameworks like:react js, vie js, next js etc. However, if you want to use JavaScript on the backend, you'll need to learn NodeJS
- Javascript is a simple language to learn.
- Web pages now have more functionality because of Javascript.
- To the end-user, Javascript is quite quick

20) What are object prototypes?

Object Prototypes: Every object in JavaScript has a prototype, which is another object that it inherits properties and methods from. Prototypes form a chain called the prototype chain.

Ex

```
// Creating a prototype object
var personPrototype = {
  greet: function() {
    console.log('Hello!');
  }
};
// Creating a new object using the prototype
var person = Object.create(personPrototype);
```

```
// Using the inherited method
person.greet(); // Output: Hello!
```

21) What are callbacks ?

A callback is a function that will be executed after another function gets executed. In javascript.

Ex

```
function divideByHalf(sum){
  console.log(Math.floor(sum / 2));
}
function multiplyBy2(sum){
  console.log(sum * 2);
}
function operationOnSum(num1,num2,operation){
  var sum = num1 + num2;
  operation(sum);
}
operationOnSum(3, 3, divideByHalf); // Outputs 3
operationOnSum(5, 5, multiplyBy2); // Outputs 20
```

22) What are the types of errors in javascript?

There are two types of errors in javascript.

- **Syntax error**: Syntax errors are mistakes or spelling problems in the code. It is catch when we compile program,
- **Logical (Reasoning) error**: Reasoning mistakes occur when the syntax is proper but the logic or program is incorrect. These are sometimes more difficult to correct than syntax issues since these applications do not display error signals for logic faults.

23) What is Memoization

Memoization is a technique used in programming, specifically in JavaScript, to optimize the performance of functions by caching their results. In simple terms,when we pass argument in function then check result of that same argument in cache if get the result then return direct result if not get result then calculate result and return output and store that result in cache for next time.

Ex

```
// Function to calculate the factorial of a number
function factorial(n) {
  if (n === 0 || n === 1) {
    return 1;
```

```
  } else {
    return n * factorial(n - 1);
  }
}
// Applying memoization to the factorial function
function memoizedFactorial() {
  const cache = {};

  return function memoize(n) {
    if (n in cache) {
      return cache[n];
    } else {
      const result = factorial(n);
      cache[n] = result;
      return result;
    }
  };
}
// Using the memoized factorial function
const memoized = memoizedFactorial();
console.log(memoized(5)); // Calculates factorial of 5 and caches
the result
```

```
console.log(memoized(5)); // Returns the cached result directly

console.log(memoized(3)); // Calculates factorial of 3 and caches the result

console.log(memoized(3)); // Returns the cached result directly
```

24) What is Recursion in programming language

Recursion is a programming concept where a function calls itself during its execution until base condition not match. It is a way to solve complex problems by breaking them down into smaller,

Ex

```
function factorial(n) {
  if (n === 0) {
    return 1;
  }
  return n * factorial(n - 1);
}

console.log(factorial(5)); // Output: 120
```

25) What is the use of a constructor function in javascript?

In JavaScript, a constructor function is used to create and initialize objects.

It serves as a blueprint for creating multiple objects with similar properties and methods.

Ex

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}
const person1 = new Person("John", 25);
const person2 = new Person("Jane", 30);
console.log(person1.name);  // Output: John
console.log(person2.age);   // Output: 30
```

26) What is DOM?

The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.

We can do following task using dom

- Accessing Elements
- Changing Element Content
- Modifying Styles
- Creating New Elements

27) Which method is used to retrieve a character from a certain index?

The charAt() function of the JavaScript string finds a char element at the supplied index.

28) What do you mean by BOM?

The Browser Object Model (BOM) is used to interact with the browser.

The default object of browser is window means you can call all the functions of window by specifying window or directly. For example:

⇨ All Above we describe about bom

29) What is the distinction between client-side and server-side JavaScript?

Client-side JavaScript:

- Executed on the client's web browser.
- The code is embedded within an HTML document or loaded as an external JavaScript file.
- Used to enhance the user interface and interactivity of web pages.
- Executes directly in the browser, allowing for dynamic updates and modifications to the web page without requiring a server round-trip.
- Examples include form validation, DOM manipulation, event handling, and AJAX requests.

Server-side JavaScript:

- Executed on the server.
- Commonly used in web development frameworks, such as Node.js.
- Runs on the server, generating dynamic content or processing requests before sending the result back to the client.
- Enables server-side tasks like accessing databases, handling file uploads, authentication, and business logic.
- Examples include building RESTful APIs, server-side rendering, real-time applications, and server-side data processing.

Advance Javascript Interview Quetion

30) What are arrow functions?

Arrow functions were introduced in the ES6 version of javascript. They provide us with a new and shorter syntax for declaring functions. Arrow functions can only be used as a function expression.

Arrow functions are declared without the function keyword. If there is only one returning expression then we don’t need to use the return keyword

```
// Arrow Function Expression

var arrowAdd = (a,b) => a + b;
```

31) What do mean by prototype design pattern?

Prototype Design Pattern is a creational design pattern that allows objects to share behavior by using a prototype object as a blueprint or template. This pattern promotes code reusability and improves performance by avoiding the duplication of methods and properties across multiple instances of similar objects.

Real-life example of the Prototype Design Pattern:

Let's consider a scenario where we want to create different types of cars, each with specific features and functionalities. Instead of creating each car instance from scratch, we can use the Prototype Design Pattern to

define a common prototype (template) for all cars and then customize each instance by modifying the prototype.

32) Differences between declaring variables using var, let and const.

keyword	const	let	var
global scope	no	no	yes
function scope	yes	yes	yes
block scope	yes	yes	no
can be reassigned	no	yes	yes

33) What is the rest parameter and spread operator?

Rest Parameter: The rest parameter is represented by three dots (...) and allows a function to accept an indefinite number of arguments as an array. It gathers the remaining arguments into an array, making it easier to work with them.

```
function sum(...numbers) {
  let total = 0;
  for (let num of numbers) {
    total += num;
  }
  return total;
```

```
}

console.log(sum(1, 2, 3));  // Output: 6

console.log(sum(4, 5, 6, 7));  // Output: 22
```

Spread Operator: The spread operator is also represented by three dots (...) and allows you to expand elements of an array or object. It can be used in function calls, array literals, or object literals.

```
function greet(name, age) {
  console.log(`Hello ${name}! You are ${age} years old.`);
}

const person = ['John', 25];

greet(...person);
```

34) In JavaScript, how many different methods can you make an object?

- Object.
- using Class.
- create Method.
- Object Literals.
- using Function.
- Object Constructor.

35) What is the use of promises in javascript?

Promises are used to handle asynchronous operations in javascript.

They help manage and control the flow of code execution, especially when dealing with operations that may take some time to complete, such as fetching data from a server or reading files.

```
// Create a promise
const fetchData = new Promise((resolve, reject) => {
setTimeout(() => {
    const data = { name: 'John', age: 25 };
    resolve(data);
    // reject('Error: Something went wrong');
  }, 2000);
});

// Use the promise
fetchData
  .then((data) => {
    console.log('Data:', data);
  })
  .catch((error) => {
    console.log('Error:', error);
```

```
});
```

36) What are classes in javascript?

In JavaScript, classes are a way to define objects and their behavior. They serve as blueprints for creating multiple instances of similar objects. Classes encapsulate data and functions that operate on that data.

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  greet() {
    console.log(`Hello, my name is ${this.name} and I'm ${this.age} years old.`);
  }
}
const john = new Person('John', 25);
john.greet(); // Output: Hello, my name is John and I'm 25 years old.
```

37) What are generator functions?

The Generator Function are a special type of function in javascript that allows you to pause and resume their execution, producing a sequence of values over time. They are denoted by using and asterisk('*') after function keyword.

Ex

```
Function* generatorFunction() {


}
```

38) Explain WeakSet in javascript.

The WeakSet is similar to a Set. However, WeakSet can only contain objects whereas a Set can contain any data types such as strings, numbers, objects, etc.

Ex

```
const weakSet = new WeakSet();

console.log(weakSet); // WeakSet {}

let obj = {
   message: 'Hi',
   sendMessage: true
}
```

```
// adding object (element) to WeakSet

weakSet.add(obj);

console.log(weakSet); // WeakSet {{message: "Hi", sendMessage: true}}
```

39) Why do we use callbacks?

1) Handling Asycronouse Operations
2) Event Handling
3) Avoiding Callback Hell
4) Modularity and Reusability
5) Handling Errors

40) Explain WeakMap in javascript.

WeakMap is similar to a regular Map, but with some important differences. the keys must be objects, and the reference to the key object is held weakly, meaning that it does not prevent the key from being garbage collected if there are no other references to it.

```
// Create a WeakMap

const weakMap = new WeakMap();

// Create some objects

const obj1 = {};
```

```
const obj2 = {};

// Set key-value pairs in the WeakMap

weakMap.set(obj1, 'Value 1');

weakMap.set(obj2, 'Value 2');

// Get values using the keys

console.log(weakMap.get(obj1)); // Output: Value 1

console.log(weakMap.get(obj2)); // Output: Value 2
```

41) What is Object Destructuring?

JavaScript object destructuring is a technique that allows you to extract specific values from an object and assign them to variables in a more concise and convenient way. It simplifies accessing object properties by providing a shorthand syntax.

```
const person = { name: 'John', age: 30, city: 'New York' };

const { name, age } = person;

console.log(name); // Output: John

console.log(age); // Output: 30
```

42) Difference between prototypal and classical inheritance

Prototypal Inheritance:

- In prototypal inheritance, objects inherit directly from other objects.
- Each object has an internal link to another object called its prototype.
- When you access a property or method on an object, JavaScript looks for that property/method in the object itself first, and if it doesn't find it, it looks for it in the prototype chain.

Ex

```
// Creating an object to be used as a prototype
const vehiclePrototype = {
  start() {
    console.log("The vehicle starts.");
  },
};
// Creating a new object that inherits from vehiclePrototype
const car = Object.create(vehiclePrototype);
car.drive = function () {
  console.log("The car is driving.");
};
car.start(); // Outputs: "The vehicle starts."
car.drive(); // Outputs: "The car is driving."
```

Classical Inheritance:

- Classical inheritance is based on the concept of classes and uses constructor functions and the new keyword to create objects.
- You define a class as a blueprint and create instances (objects) based on that class.
- In classical inheritance, objects inherit from classes, not directly from other objects

Ex

```
// Defining a class using constructor function
function Vehicle() {}
// Adding methods to the class prototype
Vehicle.prototype.start = function () {
  console.log("The vehicle starts.");
};
// Creating a new instance of the Vehicle class
const car = new Vehicle();
car.start(); // Outputs: "The vehicle starts."
```

43) What is a Temporal Dead Zone?

A Temporal Dead Zone (TDZ) is a concept in JavaScript that occurs when you try to access a variable before it has been declared using the let or const keywords. In simple terms, it's a zone in your code where a variable exists but cannot be accessed.

Ex

console.log(myVariable); // Throws a ReferenceError: myVariable is not defined

let myVariable = 10; // Variable declaration

44) What do you mean by JavaScript Design Patterns?

45) Difference between Async/Await and Generators usage to achieve the same functionality.

- Async/await is a syntax introduced in ES2017 (ES8) and is built on top of Promises.
- It allows you to write asynchronous code that looks and behaves like synchronous code, making it easier to read and understand.
- The 'async' keyword is used to define an asynchronous function, and the 'await' keyword is used to pause the execution of the function until a promise is resolved.
- You can use try/catch blocks to handle errors in an asynchronous manner.
- Async/await is well-suited for linear control flow, where you want to await multiple promises sequentially or in parallel.

46) What are the primitive data types in JavaScript?

- **Boolean**
- **Undefined**

- **Null**
- **Number**
- **String**

47) What is the role of deferred scripts in JavaScript?

Deferred scripts in JavaScript play a role in controlling the order in which scripts are executed on a web page. When a browser encounters a regular script tag, it halts the parsing of the HTML document and executes the script immediately. However, deferred scripts allow the browser to continue parsing the HTML while the script is being fetched and only execute the script after the HTML parsing is complete.

48) What has to be done in order to put Lexical Scoping into practice?

https://www.interviewbit.com/javascript-interview-questions/#difference-between-var-and-let-keyword-in-javascript