JavaScript Interview Questions - *By Murtaza*

1. **What is JavaScript?**

   JavaScript is client-side scripting language that executes in the user's browser. It possesses object-oriented capabilities and enable us to create interactive web-pages.

2. **What is the difference between programming language and scripting language?**

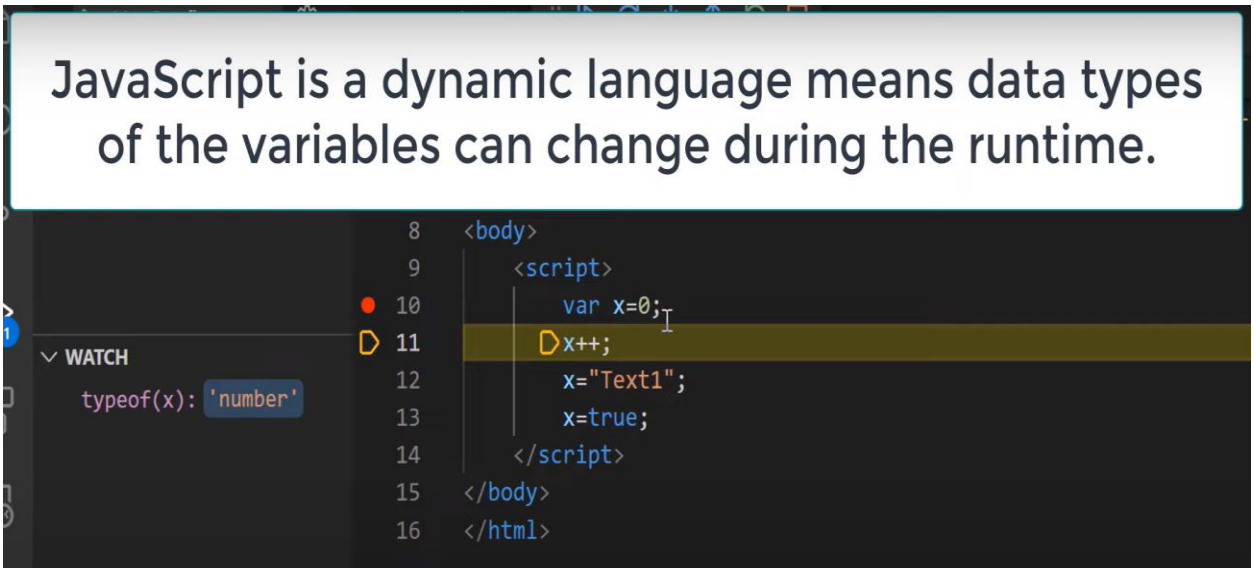
   A programming language is a formal language that can be used to create computer programs that instruct the computer to perform a task. The programming languages can be high level or low-level languages.

   These programs or source codes are converted into machine code using a compiler or an interpreter.

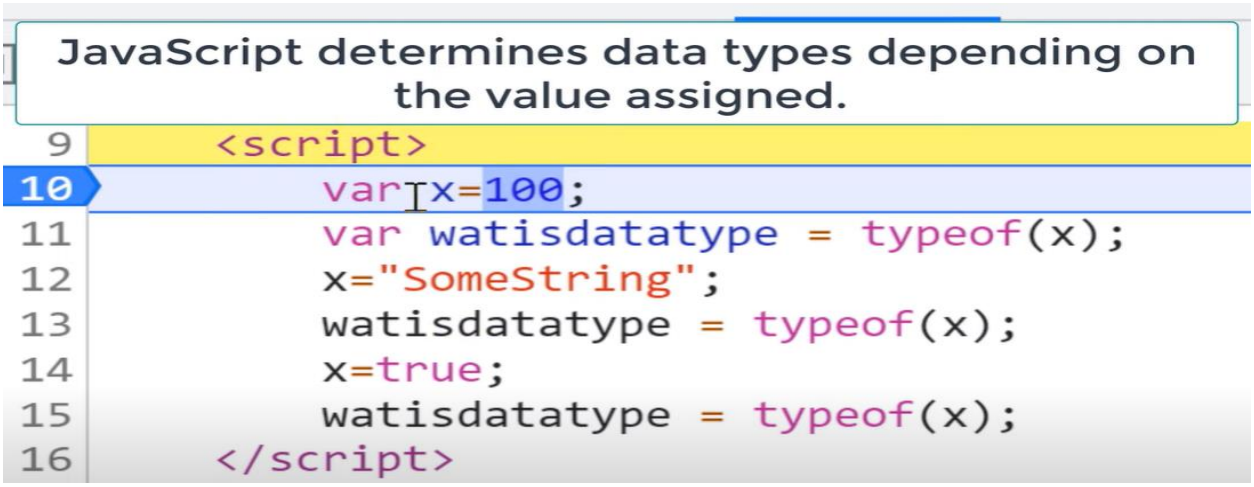
   A scripting language is a type of programming language which does not require explicit compilation step, and it is designed for a runtime system to automate the execution of tasks.

   For example, a JavaScript program is not needed to be compiled before we run it. These are also known as very high-level programming languages because of working at a high level of abstraction. Scripting languages support "script," which is small program written for a specific runtime environment.

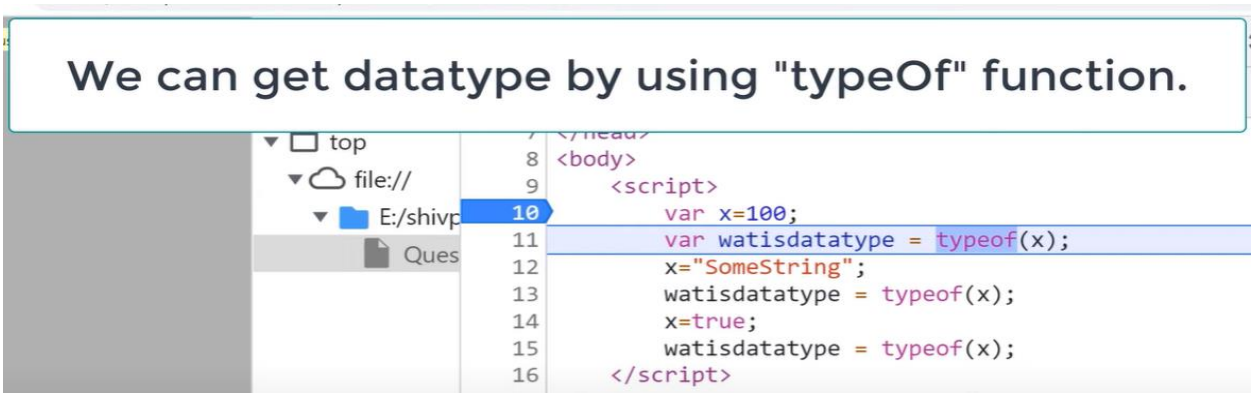
3. **Why do we call JavaScript as dynamic language?**



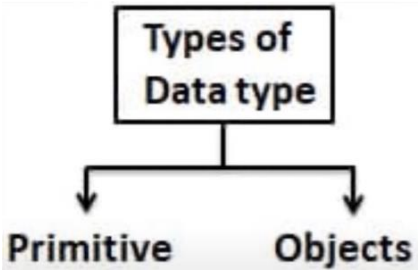
4. **How does JavaScript determine data types?**



5. **What is typeof function?**
   Or
   How to check data type in JavaScript?

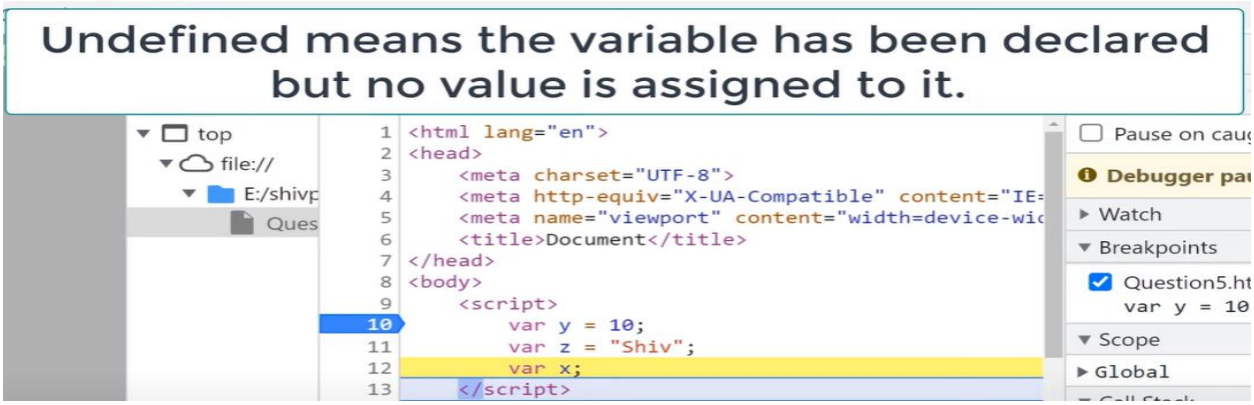


6. **What are different datatypes in JavaScript?**

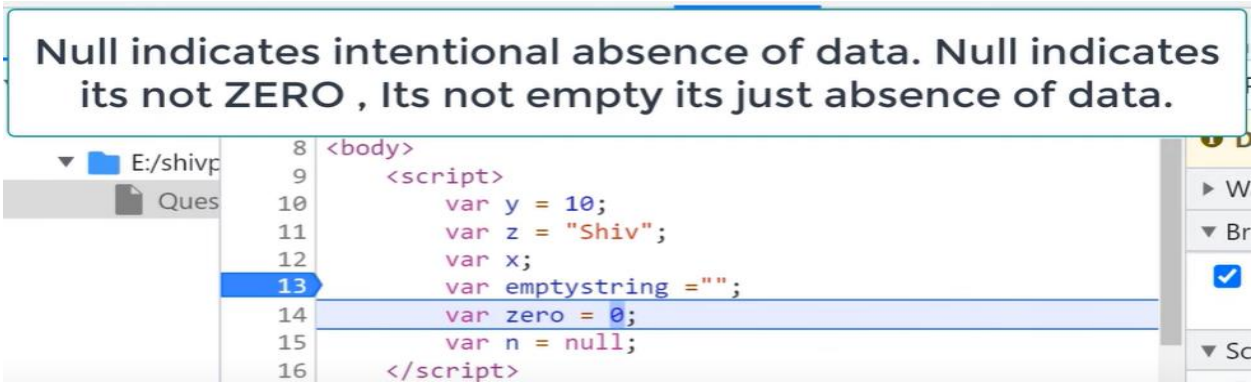



```
<script>
    // primitive
    var str = "Shiv"; // string
    var num = 10; // number
    var nu = null; // null
    var und = undefined; // undefined
    var bool = true; // boole
    var big = 100 // big int
    var sym = Symbol(); // symbol

    // objects
    var obj = new Object();// object
</script>
```

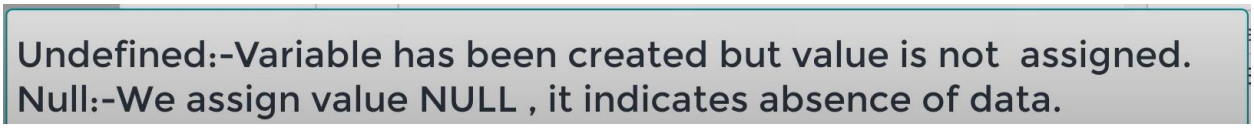
7. **Explain undefined data type.**



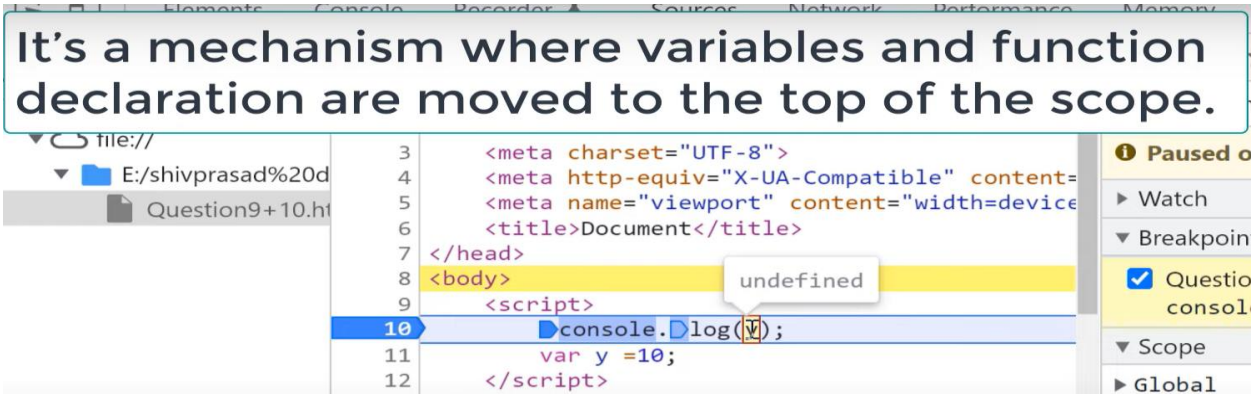
8. **What is null?**



9. **Null VS Undefined.**



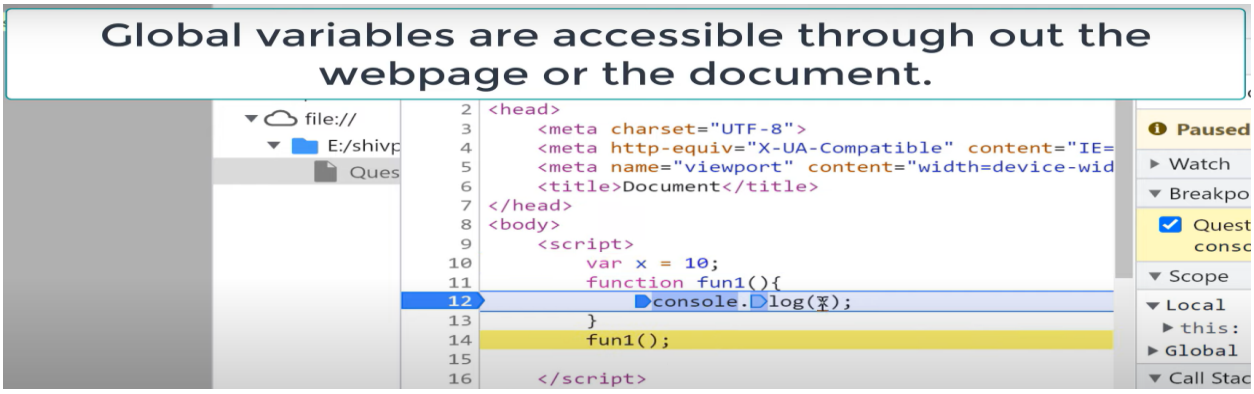
10. **Explain Hoisting.**



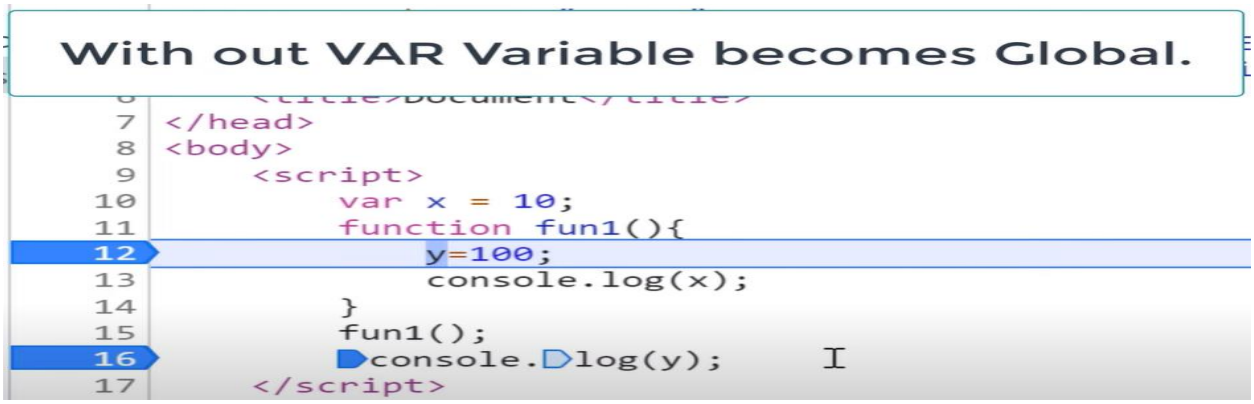
11. Are JavaScript initialization hoisted?

No.

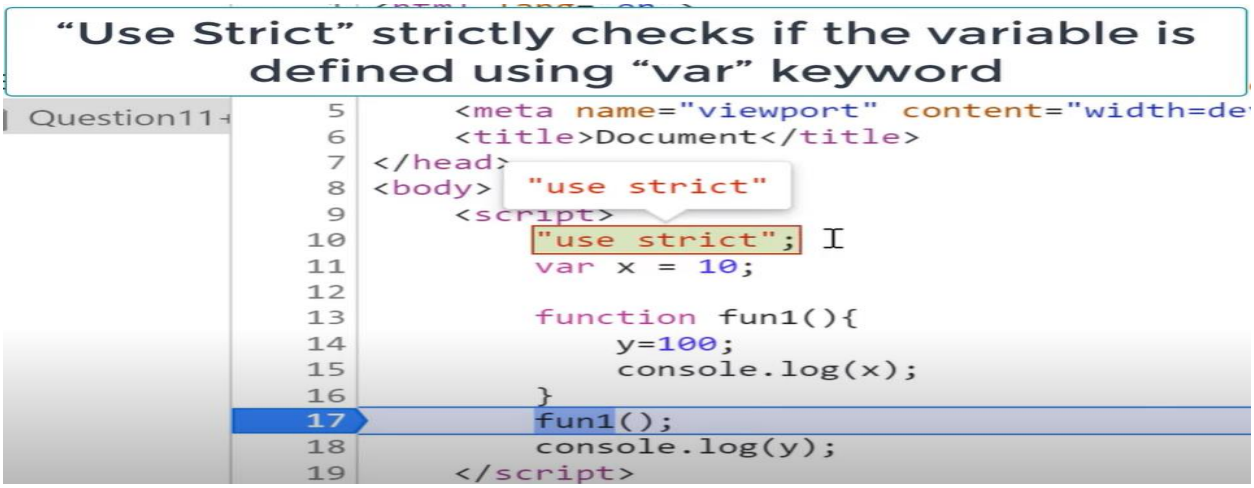
12. What are global variables?



13. What happens when you declare variable without var keyword?



14. What is Use Strict?
or
How to force developers to use var keyword?



15. How can we handle Global Variables?
or
How can we avoid Global Variables?

```
<script>
    var global = {};
    global.connectionString = "Test";
    global.logDir ="d:\Logs";

    var myGlobal = (function(){
                    var connectionString = "Test"
                    function GetConnection(){
                        return connectionString;
                    }
                    return{
                        GetConnection
                    }
                    })();
    var str = myGlobal.GetConnection();
</script>
```

Its difficult to avoid global variables. But we can organize it properly by doing two things: -

Put global variables in a proper Namespace.
Module pattern using Closures and IIFE.

16. What are Closures? Why do we need Closures?

Closures are functions inside function and it makes a normal function stateful.

```
<script>
    function SimpleFunction(){
        var x =0;
        x++;
    }
    function ClosureFunction(){
        var x =0;
        function Increment(){
            x++;
        }
        return {
            Increment
        }
    }
    SimpleFunction();
    SimpleFunction();
    var ref = ClosureFunction();
    ref.Increment();
    ref.Increment();
```


```
function ClosureFunction(){
    var x;
    function Increment(){ ···
    function GetXValue(){ ···
    function Init(){ ···
    Init();
    return {
        Increment,
        GetXValue
    }
}

var ref = ClosureFunction();
ref.Increment();
ref.Increment();
alert(ref.GetXValue());
ref.Init();
```

```
function ClosureFunction(){
    var x;
    function Increment(){
        x++;
    }
    function GetXValue(){
        return x;
    }
    function Init(){
        x=0;
    }
    Init();
    return {
        Increment,
        GetXValue
    }
}
```

17. What is IIFE?

IIFE stands for Immediately Invoked Function Expression.

```
9     <script>
10        var x=0;
11        (function(){
12           var y=10
13           alert("I am IIFE and I will execute once whe
14        })();
15        alert(y);
16    </script>
```

It's an anonymous function which gets immediately invoked.

18. What is the use of IIFE? What is the name collision in global scope? IIFE VS Normal function.

.Name collision happens when same name function names and variable names are declared.

```
4     <meta http-equiv="X-UA-Compatible" content="IE=edge">
5     <meta name="viewport" content="width=device-width, in
6     <title>Document</title>
7 </head>
8 <body>
9     <script>
10        // What is name collission ?
11        // Normal function vs IIFE
12        // What is the use of IIFE ?
13        function Init(){
14            // Some initliazation code
15            var x = 0;
16        }
17        var Init = 0;
18        Init();
```

Because IIFE does not have name , So there is no way you can get name collision

A normal function has a name while IIFE does not have name.

So with a normal function you can have a name collision but with IIFE you will not have name , you will not name collision.

Question9+10.html
Question11+12+13+14+15.h...
Question15+16.html
Question17+18.html
Question19.html
Question20.html

```
11    // Normal function vs IIFE
12    // What is the use of IIFE ?
13    // function Init(){
14    //     // Some initliazation code
15    //     var x = 0;
16    // }
17    var Init = 0;
18    //Init();
19    (function(){
20        // init
21    })();
```

19. What are design patterns?

Design patterns are design level solutions for recurring problems that we software engineers come across often. It's not code - I repeat, ❌CODE. It is like a description on how to tackle these problems and design a solution.

Using these patterns is considered good practice, as the design of the solution is quite tried and tested, resulting in higher readability of the final code.

20. What are various ways to create JavaScript Objects?


```
// 1.literal
var pat = {"name":"","address":""};
pat.Admit = function(){
    alert("I am admitted");
}
```

```
// 2. object.create
var patnew = Object.create(pat);
patnew.age = 10;
```

```
function Patient(){
    this.name = "";
    this.address = "";
    this.Admit = function(){

    }
}
var pat1 = new Patient();
```

```
// 4. class
class PatientClass{
    constructor(name,address){
        this.name = "";
        this.address = "";
    }
}
var p = new PatientClass();
```

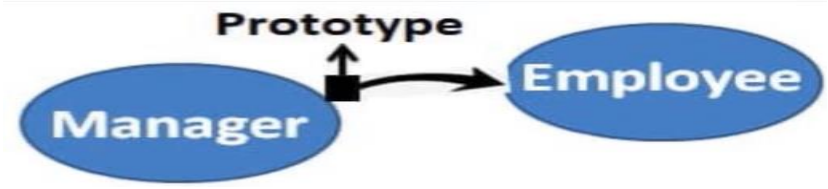
21. How can we do inheritance in JavaScript? What is prototype in JavaScript? Explain prototype chaining?

JavaScript uses object inheritance or prototypical inheritance. Inheritance is done using Prototype object.

```
function Employee(){
    this.Name = "";
    this.DoWork = function(){
        alert("Basic work");
    }
    this.Attendance = function(){
        alert("Attendance needed");
    }
}
```

```
function Manager(){
    this.Cabin = "";
    this.DoWork = function(){
        alert("Manages team");
    }
}
```

```
var emp = new Employee();
Manager.prototype = emp;
```




```
var man= new Manager();
man.Name = "Shiv";
man.Attendance();
man.DoWork();
```

Every JavaScript object has a Prototype object.

It's an inbuilt object provided by JavaScript.

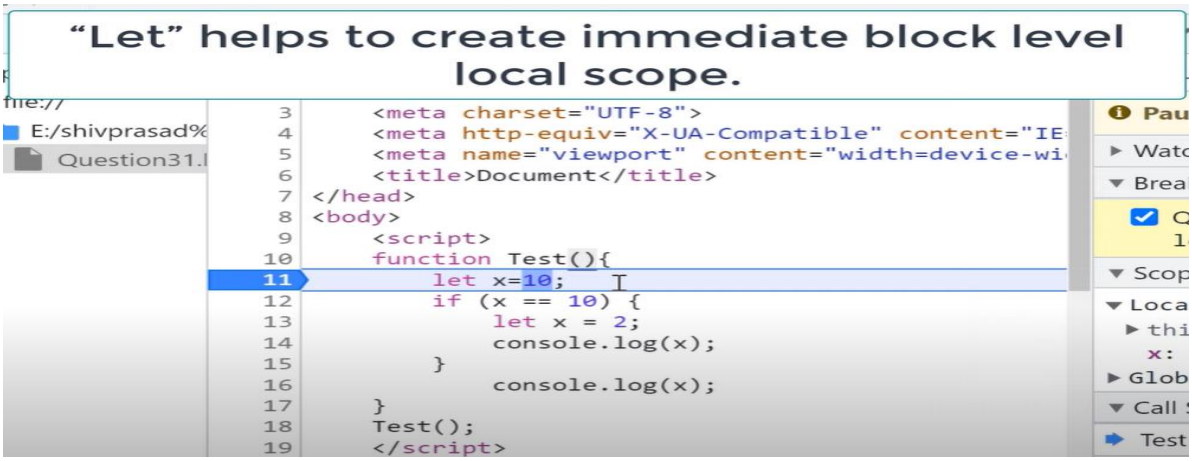
Prototype chaining is a process where the property/ methods are first checked in the current object ,

if not found then it checks in the prototype object ,

if does not find in that it try checking the prototypes prototype object , until he get the prototype object null.

22. What is let keyword?

"Let" helps to create immediate block level local scope.



23. Explain Temporal Dead Zone?

TDZ it's a period or it's a state of a variable where variables are named in memory but they are not initialized with any value.

24. Let VS Var

	Var	Let
Scope	Scoped to the Immediate function body.	Scoped to the immediate enclosing block.
Initialized value	Value initialized with Undefined	Value initialized with nothing.

25. What is Currying in JavaScript?

Currying is a technique of evaluating function with multiple arguments, into sequence of functions with single argument. In other words, when a function, instead of taking all arguments at one time, takes the first one and return a new function that takes the second one and returns a new function which takes the third one, and so forth, until all arguments have been fulfilled.

There are several reasons why currying is ideal:

- Currying is a checking method to make sure that you get everything you need before you proceed
- It helps you to avoid passing the same variable again and again
- It divides your function into multiple smaller functions that can handle one responsibility. This makes your function pure and less prone to errors and side effects
- It is used in functional programming to create a higher-order function.

26. 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.

Example:

```
var x = 2;
var y = "2";
(x == y)   // Returns true since the value of both x and y is the same
(x === y)  // Returns false since the typeof x is "number" and typeof y is "string"
```

27. What is NaN property in JavaScript?

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

typeof of NaN will return a Number.

To check if a value is NaN, we use the isNaN() function,

Note- isNaN() function converts the given value to a Number type, and then equates to NaN.

28. What is this keyword in JavaScript?

In JavaScript, the ‘this’ keyword refers to an object.

Which object depends on how **this** is being invoked (used or called).

The ‘this’ keyword refers to different objects depending on how it is used:

- In an object method, this refers to the object.
- Alone, this refers to the global object.
- In a function, this refers to the global object.
- In a function, in strict mode, this is undefined.
- In an event, this refers to the element that received the event.
- Methods like call(), apply(), and bind() can refer this to any object.

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

- **call():**

  It’s a predefined method in javascript.
  This method invokes a method (function) by specifying the owner object.

  Example 1:

```
function sayHello(){
  return "Hello " + this.name;
}

var obj = {name: "Sandy"};

sayHello.call(obj);

// Returns "Hello Sandy"
```

  call() method allows an object to use the method (function) of another object.

  Example 2:

```
var person = {
  age: 23,
  getAge: function(){
    return this.age;
  }
}
var person2 = {age:  54};
```

```
person.getAge.call(person2);
// Returns 54
```

call() accepts arguments:

```
function saySomething(message){
  return this.name + " is " + message;
}
var person4 = {name:  "John"};
saySomething.call(person4, "awesome");
// Returns "John is awesome"
```

- **apply()**

  The apply method is similar to the call() method. The only difference is that,

  call() method takes arguments separately whereas, apply() method takes arguments as an array.

```
function saySomething(message){
  return this.name + " is " + message;
}
var person4 = {name:  "John"};
saySomething.apply(person4, ["awesome"]);
```

- **bind():**

  This method returns a new function, where the value of “this” keyword will be bound to the owner object, which is provided as a parameter.

  Example with arguments:

```
var bikeDetails = {
    displayDetails: function(registrationNumber,brandName){
    return this.name+ " , "+ "bike details: "+ registrationNumber + " , " +
brandName;
  }
}

var person1 = {name:  "Vivek"};

var detailsOfPerson1 = bikeDetails.displayDetails.bind(person1, "TS0122",
"Bullet");
```

```
// Binds the displayDetails function to the person1 object

detailsOfPerson1();
// Returns Vivek, bike details: TS0452, Bullet
```

30. What are callbacks?

A callback is a function that will be executed after another function gets executed. In javascript, functions are treated as first-class citizens, they can be used as an argument of another function, can be returned by another function, and can be used as a property of an object.

Functions that are used as an argument to another function are called callback functions.

Example:

```
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
```

31. What is Memoization?

Memoization is a form of caching where the return value of a function is cached based on its parameters. If the parameter of that function is not changed, the cached version of the function is returned.

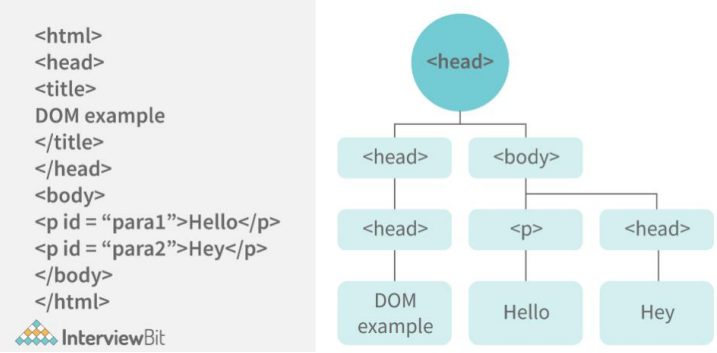
Consider the following function:

```
function addTo256(num){
  return num + 256;
```

```
}

addTo256(20); // Returns 276

addTo256(40); // Returns 296

addTo256(20); // Returns 276 //memorized cache value
```

32. What is DOM?

- DOM stands for Document Object Model. DOM is a programming interface for HTML and XML documents.
- When the browser tries to render an HTML document, it creates an object based on the HTML document called DOM. Using this DOM, we can manipulate or change various elements inside the HTML document.
- Example of how HTML code gets converted to DOM:



33. 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.

Let's compare the normal function declaration and the arrow function declaration in detail:

```
// Traditional Function Expression
var add = function(a,b){
  return a + b;
}

// Arrow Function Expression
var arrowAdd = (a,b) => a + b;
```

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 as well in an arrow function as shown in the example above. Also, for functions having just one line of code, curly braces { } can be omitted.

34. What is the rest parameter and spread operator?

Rest parameter (…):

It provides an improved way of handling the parameters of a function.
Using the rest parameter syntax, we can create functions that can take a variable number of arguments.

Any number of arguments will be converted into an array using the rest parameter.
It also helps in extracting all or some parts of the arguments.

Rest parameters can be used by applying three dots (...) before the parameters.

```
function extractingArgs(...args){
  return args[1];
}

// extractingArgs(8,9,1); // Returns 9

function addAllArgs(...args){
  let sumOfArgs = 0;
  let i = 0;
  while(i < args.length){
    sumOfArgs += args[i];
    i++;
  }
  return sumOfArgs;
}

addAllArgs(6, 5, 7, 99); // Returns 117
addAllArgs(1, 3, 4); // Returns 8
```

**Note- Rest parameter should always be used at the last parameter of a function:

```
// Incorrect way to use rest parameter

function randomFunc(a,...args,c){
```

```
//Do something
}

// Correct way to use rest parameter

function randomFunc2(a,b,...args){
//Do something
}
```

Spread operator (…):

Although the syntax of the spread operator is exactly the same as the rest parameter, the spread operator is used to spreading an array, and object literals. We also use spread operators where one or more arguments are expected in a function call.

```
function addFourNumbers(num1,num2,num3,num4){
  return num1 + num2 + num3 + num4;
}

let fourNumbers = [5, 6, 7, 8];
addFourNumbers(...fourNumbers);

// Spreads [5,6,7,8] as 5,6,7,8

let array1 = [3, 4, 5, 6];
let clonedArray1 = [...array1];

// Spreads the array into 3,4,5,6

console.log(clonedArray1); // Outputs [3,4,5,6]


let obj1 = {x:'Hello', y:'Bye'};
let clonedObj1 = {...obj1}; // Spreads and clones obj1
console.log(obj1);

let obj2 = {z:'Yes', a:'No'};
let mergedObj = {...obj1, ...obj2}; // Spreads both the objects and merges it
console.log(mergedObj);

// Outputs {x:'Hello', y:'Bye',z:'Yes',a:'No'};
```

***Note- Key differences between rest parameter and spread operator:

- Rest parameter is used to take a variable number of arguments and turns them into an array while the spread operator takes an array or an object and spreads it
- Rest parameter is used in function declaration whereas the spread operator is used in function calls.

35. What do you mean by JavaScript Design Patterns?

JavaScript design patterns are repeatable approaches for errors that arise sometimes when building JavaScript browser applications. They truly assist us in making our code more stable.
They are divided mainly into 3 categories

1. Creational Design Pattern
2. Structural Design Pattern
3. Behavioral Design Pattern.

- **Creational Design Pattern:** The object generation mechanism is addressed by the JavaScript Creational Design Pattern. They aim to make items that are appropriate for a certain scenario.
- **Structural Design Pattern:** The JavaScript Structural Design Pattern explains how the classes and objects we've generated so far can be combined to construct bigger frameworks. This pattern makes it easier to create relationships between items by defining a straightforward way to do so.
- **Behavioral Design Pattern:** This design pattern highlights typical patterns of communication between objects in JavaScript. As a result, the communication may be carried out with greater freedom.