**JavaScript Interview Questions**

**1.what is JavaScript?**

Ans) Javascript is a client side language which is understood by the browsers and will be added in the HTML Code.

a) JavaScript is the world's most popular programming language.

b)JavaScript is a scripting or programming language that allows you to implement complex things on web pages

https://www.w3schools.com/css/css3\_variables.asp

**2.what is purpose of javascript ?**

Ans)To create the logic and control the content of HTML structure and css.

**3) Javascript Display possibilities**

Ans)

* Writing into an HTML element, using innerHTML.
* Writing into the HTML output using document.write().
* Writing into an alert box, using window.alert().
* Writing into the browser console, using console.log().

**3.Javascript statements?**

Ans) In a programming language, these **programming instructions** are called **statements**.

<p id="demo"></p>

<script>

let x, y, z; // Statement 1

x = 5; // Statement 2

y = 6; // Statement 3

z = x + y; // Statement 4

document.getElementById("demo").innerHTML =

"The value of z is " + z + ".";

</script>

The value of z is 11.

**4. When to Use JavaScript var?**

Always declare JavaScript variables with var,let, orconst.

The var keyword is used in all JavaScript code from 1995 to 2015.

The let and const keywords were added to JavaScript in 2015.

If you want your code to run in older browsers, you must use var.

## 5.When to Use JavaScript const?

If you want a general rule: always declare variables with const.

In this example, price1, price2, and total, are variables:

**6.Explain dffernt data types?**

Ans)Boolean ,string,number,undefind,null,function,object

**7.Why to use NaN in javscript?**

Ans)”NaN” is a not a number

NaN function is used to check the number in the argument.

If it is does not contain number then it will be return true else it will return false.

**8.Explain negative number in javascript?**

Ans)”Negative number “ is a number which is obtained from dividing the number by zero.

**9.Can javascript code be broken into multiple lines**

Ans)Yes it is possible to break the javascript code to multiple lines using ‘\’.

**10.What are the undeclared variables in java script?**

Ans)Undeclared variables are the ones which does not exit and not declared in the program.

So while running the program; runtime error will be thrown.

In JavaScript, an undeclared variable is a variable that has been used without being declared using the **var**, **let**, or **const** keywords. When you try to use an undeclared variable, JavaScript will throw a reference error.

Here's an example:

// Using a variable without declaring it

console.log(x); // ReferenceError: x is not defined

// Assigning a value to an undeclared variable

y = 10; // This creates a global variable 'y'

console.log(y); // Output: 10

To fix this issue, you should always declare variables before using them:

// Declaring and then using a variable

var x;

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

// Assigning a value to a declared variable

x = 5;

console.log(x); // Output: 5

In the above example, **x** is declared before being used, which prevents the reference error.

**11.What are the undefined variables in java script ?**

Ans)Undefined variables are the ones which does exit in the program but are not defined or not assigned any value for the variables .

**Example:**

l

             console.log(b);//not defined or undecleared

             var a;

             console.log(a);//undefined

             var c = null;

             console.log(c); //null

**12.Explain prompt box in javascript ?**

Ans)Prompt box is used to get the input from the user by popping up the window to the user.

**13.Why to use ”this” keyword in javascript?**

Ans)’this’ keyword is used to **refer the current object in the program** . ‘this’ keyword is used to mailnly inside the method to refer to current variable or current object.

**14.What is “setTimeOut() function in javascript?**

Ans)”SetTimeOut” function is used to call the function given as a parameter after some time dalay.

Below is the jaavscript code for the same

Syntax: setTimeOut(function(){

alert(“This is mssg box”);

},5000);

**15.What is “setInterval()” function in javascript?**

Ans)The setInterval() method calls a function at specified intervals (in milliseconds).

The setInterval() method continues calling the function until clearInterval() is called, or the window is closed.

**16.What is “clearInterval()” function in javascript?**

Ans) “clearInterval()” function is used to clear the time set from function – “setInterval” .

**17.How to give comments in javascript?**

Ans)For single line comment – “//”

For multi-line comments – “/\* \*/”

**18.Why to use “===” operator in javascript?**

Ans)”===” operator is called strick operator and it returns true when both the values are same and returns false when the values are not same.

**19.How we can submit a form using javascript?**

Ans)For submitting form from javascript we have to use

Document.form[0].submit() method.

**20.Will javascript support automatic conversion of type?**

Ans)Yes javascript will support automatic conversion of type.

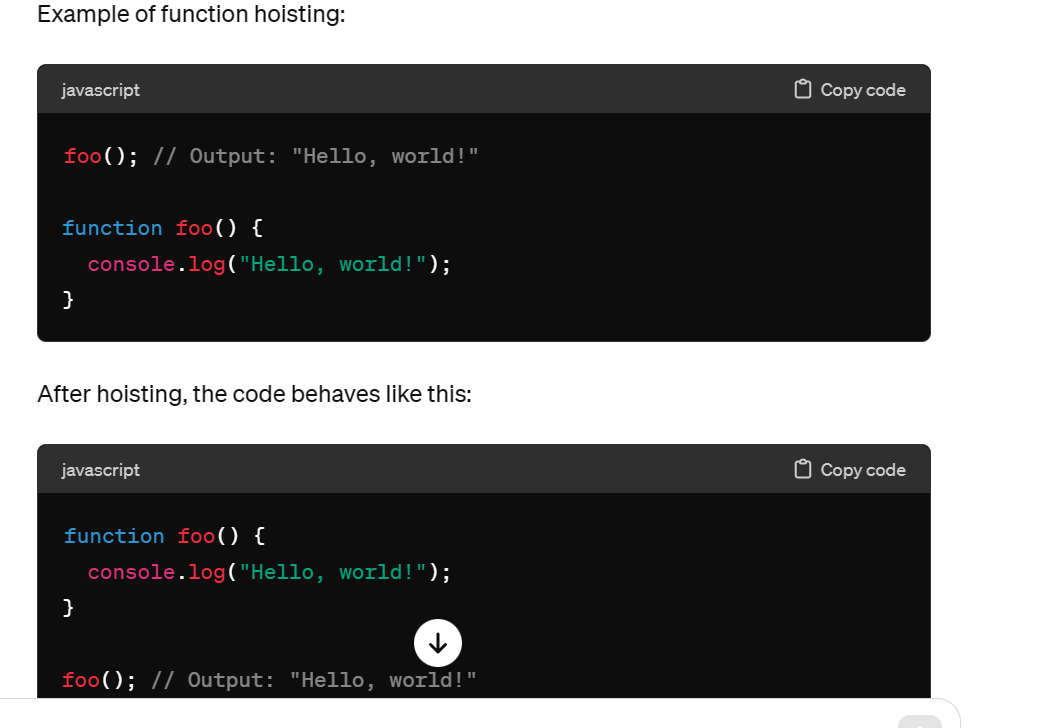
**21.How we can change the style of an element from javascript?**

Ans)Below code can be used to change the style of an element –

document.getElementById(“demo”).className = “testclass”; //camelcase

**22.What is javascript hosting?**

Ans) Hoisting is JavaScript's default behavior of moving declarations to the top of the current scope.



**23.what is javascript scope?**

Ans) Scope determines the accessibility (visibility) of variables.

JavaScript has 3 types of scope:

* Block scope
* Function scope
* Global scope
* Before ES6 (2015), JavaScript had only **Global Scope** and **Function Scope**.
* ES6 introduced two important new JavaScript keywords: let and const.
* These two keywords provide **Block Scope** in JavaScript.
* Variables declared inside a { } block cannot be accessed from outside the block:

{  
  let x = 2;  
}

## Local Scope

Variables declared within a JavaScript function, become **LOCAL** to the function.

function myFunction() {  
  let carName = "Volvo";  
  // code here CAN use carName  
}

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

## Function Scope

JavaScript has function scope: Each function creates a new scope.

Variables defined inside a function are not accessible (visible) from outside the function.

Variables declared with var, let and const are quite similar when declared inside a function.

They all have **Function Scope**:

function myFunction() {  
  var carName = "Volvo";   // Function Scope  
}

function myFunction() {  
  let carName = "Volvo";   // Function Scope  
}

function myFunction() {  
  const carName = "Volvo";   // Function Scope  
}

## Global JavaScript Variables

A variable declared outside a function, becomes **GLOBAL**.

let carName = "Volvo";  
// code here can use carName  
  
function myFunction() {  
// code here can also use carName  
}

## Global Scope

Variables declared **Globally** (outside any function) have **Global Scope**.

**Global** variables can be accessed from anywhere in a JavaScript program.

Variables declared with var, let and const are quite similar when declared outside a block.

They all have **Global Scope**:

var x = 2;       // Global scope

let x = 2;       // Global scope

const x = 2;       // Global scope

**24.Javascript error?**

## Ans) Throw, and Try...Catch...Finally

The try statement defines a code block to run (to try).

The catch statement defines a code block to handle any error.

The finally statement defines a code block to run regardless of the result.

The throw statement defines a custom error.

<p id="demo"></p>  
  
<script>  
try {  
  adddlert("Welcome guest!");   
}  
catch(err) {  
  document.getElementById("demo").innerHTML = err.message;  
}  
</script>

25)

anS)

|  |  |
| --- | --- |
| **Method** | **Description** |
| Number() | Returns a number, converted from its argument |
|  |  |
| parseFloat() | Parses a string and returns a floating point number |
| parseInt() | Parses a string and returns an integer |

|  |  |
| --- | --- |
| **Method** | **Description** |
| new Map() | Creates a new Map |
| set() | Sets the value for a key in a Map |
| get() | Gets the value for a key in a Map |
| delete() | Removes a Map element specified by the key |
| has() | Returns true if a key exists in a Map |
| forEach() | Calls a function for each key/value pair in a Map |
| entries() | Returns an iterator with the [key, value] pairs in a Map |
| **Property** | **Description** |
| size | Returns the number of elements in a Map |

## The HTML DOM Document Object

---------------------------------------------------------------------------------------

Adding and Deleting Elements

|  |  |
| --- | --- |
| **Method** | **Description** |
| document.createElement(*element*) | Create an HTML element |
| document.removeChild(*element*) | Remove an HTML element |
| document.appendChild(*element*) | Add an HTML element |
| document.replaceChild(*new, old*) | Replace an HTML element |
| document.write(*text*) | Write into the HTML output stream |

Finding HTML Elements

|  |  |
| --- | --- |
| **Method** | **Description** |
| document.getElementById(*id*) | Find an element by element id |
| document.getElementsByTagName(*name*) | Find elements by tag name |
| document.getElementsByClassName(*name*) | Find elements by class name |

**25.What is the DOM?**

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

*"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

The W3C DOM standard is separated into 3 different parts:

* Core DOM - standard model for all document types
* XML DOM - standard model for XML documents
* HTML DOM - standard model for HTML documents

**26.What is the HTML DOM?**

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

* The HTML elements as **objects**
* The **properties** of all HTML elements
* The **methods** to access all HTML elements
* The **events** for all HTML elements

In other words:**The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

## **27.What is HTTP?**

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server.

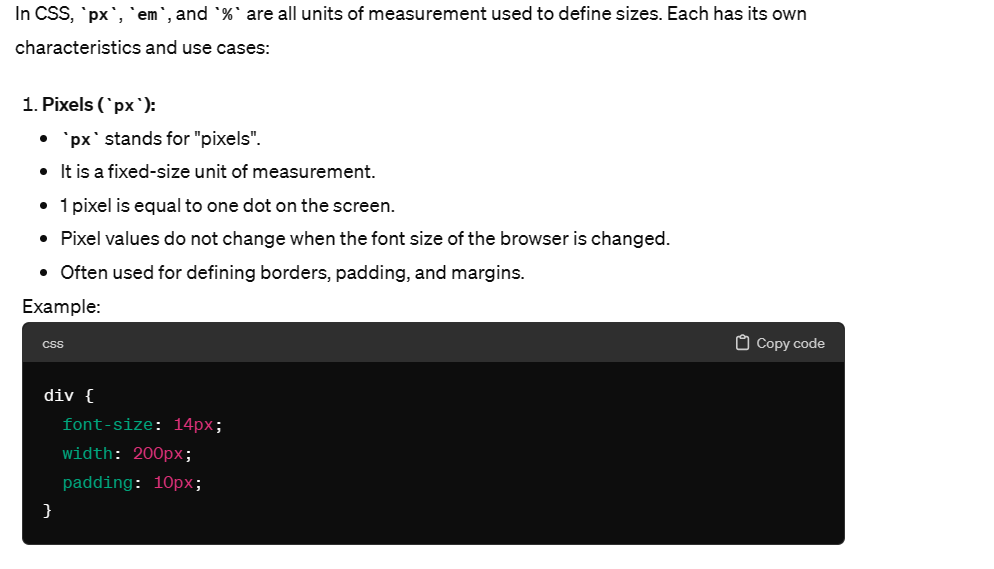
Example: A client (browser) sends an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

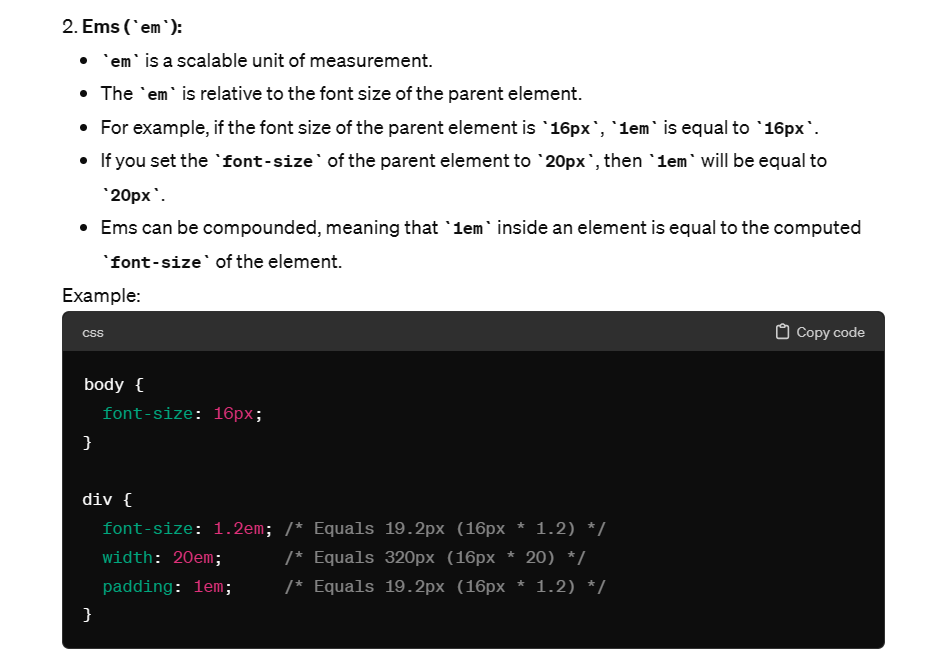
**28.HTTP Methods?**

* GET
* POST
* PUT
* HEAD
* DELETE
* PATCH
* OPTIONS
* CONNECT
* TRACE

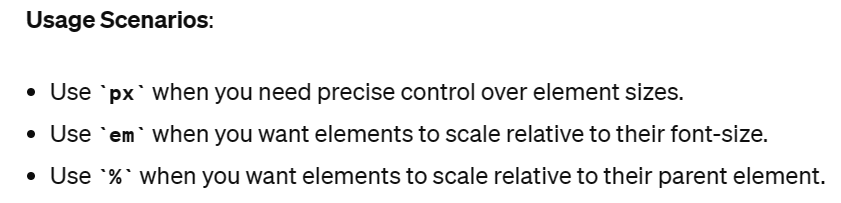
****29.What is the difference between PX, EM and Percent?****

* Pixel is a static measurement, while percent and EM are relative measurements. The size of an EM or percent depends on its parent. If the text size of body is 16 pixels, then 150% or 1.5 EM will be 24 pixels (1.5 \* 16). Look at [CSS Units](https://www.w3schools.com/cssref/css_units.asp) for more measurement units.





# 



# **30.Attribute presence and value selectors**

* Item 1
* Item 2
* Item 3

li[class] {

font-size: 200%;

}

li[class="a"] {

background-color: yellow;

}

li[class~="a"] {

color: red;

}

<h1>Attribute presence and value selectors</h1>

<ul>

<li>Item 1</li>

<li class="a">Item 2</li>

<li class="a b">Item 3</li>

<li class="ab">Item 4</li>

</ul>

<https://www.w3schools.com/css/css_combinators.asp>

31.What is difference between property binding and attr binding

Ans)1. [ngClass]=”expression” //property binding

[attr.className]=”c1” //attribute binding

2.Some attributes are not natively supported for elements

🡪[colspan]=”colVal”

🡪[attr.colspan]=”colVal”

3.Angular encourages to use property binding

🡪attrbute binding

32.Alternative of writing ngModel

🡪property binding and event binding on the same element

E.g :<input [value]=”data” (input)=”$event.target.value” />

Angular provides a built-in directive called “ngModel”

33.Why are using this multiple routers?

-Avoid this use case in applications

-you can inject components

Queryselector is defines particular section or Div

document.querySelector("p").style.backgroundColor = "red";

34.What is function In javascript?

A JavaScript function is **a block of code designed to perform a particular task.**

**Ex:**

**function greet() {**

**console.log("Hello there!");**

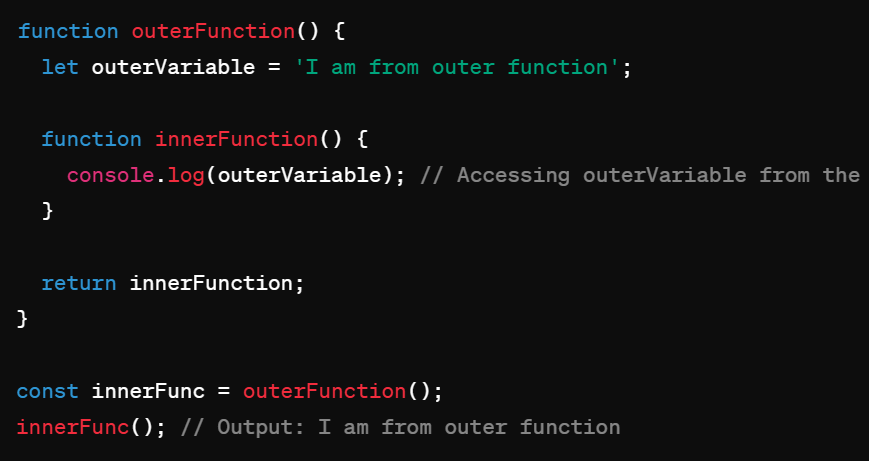
**}**

**// calling the function**

**greet();**

35. **Closures**: In a function calling object outside (global variable) and inside (local variable) , then we can call it closures.

Ex: function salutation () {  
    let name = 'Aarush';  
  
    function greet() {  
        console.log(`Hello ${name}!`);  
    }  
    return greet;  
}  
  
let wish = salutation();  
wish();



**36.JavaScript - Main Concepts:**

**Memorization**: optimizing techniques that can be used to reduce  
time-consuming calculations by saving previous input to something called cache and returning the result from it

**Call, Apply, Bind – Method:**

The call() method takes arguments **separately**.

The apply() method takes arguments as an **array**.

**Example of call():**

const person = {  
  fullName: function(city, country) {  
    return this.firstName + " " + this.lastName + "," + city + "," + country;  
  }  
}  
  
const person1 = {  
  firstName:"John",  
  lastName: "Doe"  
}  
  
person.fullName.call(person1, "Oslo", "Norway");

**Example of apply():**

const person = {  
  fullName: function(city, country) {  
    return this.firstName + " " + this.lastName + "," + city + "," + country;  
  }  
}  
  
const person1 = {  
  firstName:"John",  
  lastName: "Doe"  
}  
  
person.fullName.apply(person1, ["Oslo", "Norway"]);

Use this Reference link: <https://www.geeksforgeeks.org/explain-call-and-apply-methods-in-javascript/>

**Call() Method:**The call method is basically used to invoke the function with different**this** object. In JavaScript, **this** refers to an object. It depends on how we are calling a particular function. In the global scope, **this** refers to the global object **window**. Inside function also **this** refers to the global object **window.**

In strict mode, when we use any function then **this** refers to **undefined.**In functions like call, this could refer to a different object. With the help of the **call** method, we can invoke a particular function with different objects.

**Syntax**

object.objectMethod.call( objectInstance, arguments )

**Example:single person data**

const obj = {

        firstName: "First\_name",

        lastName: "Last\_name",

        printName: function () {

            console.log(this.firstName

                + " " + this.lastName);

        }

    };

    obj.printName();

Example:multiple persons data

const person = {  
  **fullName**: function() {  
    return this.firstName + " " + this.lastName;  
  }  
}  
const person1 = {  
  firstName:"John",  
  lastName: "Doe"  
}  
const person2 = {  
  firstName:"Mary",  
  lastName: "Doe"  
}  
  
// This will return "John Doe":  
person.fullName.call(**person1**);

**Call():passing arguments:**

printname:function(city,man){

console.log(this.fistname+""+this.lastname+","+city+","+man);

}

}

const person1={

fistname:"manjudasd",

lastname:"devarala"

}

const person2={

fistname:"manjula",

lastname:"devarala"

}

person.printname.call(person1,"city","man");

**Apply() method:**Just like the call method we can also bind the function to any object. Using apply( ) method also we can invoke a given function with different objects.

**Syntax:**

object.objectMethod.apply(objectInstance, arrayOfArguments)

**Example:**

 const obj1 = {

        firstName: "First\_name",

        lastName: "Last\_name"

    };

    const obj2 = {

        firstName: "Sachin",

        lastName: "Tendulkar"

    };

    function printName() {

        console.log(this.firstName + " " + this.lastName);

    }

    printName.apply(obj2);

**Example:apply()**

const person={

printname:function(city,man){

console.log(this.fistname+""+this.lastname+","+city+","+man);

}

}

const person1={

fistname:"manjudasd",

lastname:"devarala"

}

const person2={

fistname:"manjula",

lastname:"devarala"

}

person.printname.apply(person2,["city","man"]);

**output:”**manjuladevarala,city,man”

**Bind():**

With the bind() method, an object can borrow a method from another object.

The example below creates 2 objects (person and member).

The member object borrows the fullname method from the person object:

**Example:**

const person = {  
  firstName:"John",  
  lastName: "Doe",  
  fullName: function () {  
    return this.firstName + " " + this.lastName;  
  }  
}  
  
const member = {  
  firstName:"Hege",  
  lastName: "Nilsen",  
}  
  
let fullName = person.fullName.bind(member);

fullName();

**Example:**

const person ={

fistname:"manjudasd",

lastname:"devarala",

fullName:function(){

console.log(this.fistname+""+this.lastname);

}

}

const member ={

fistname:"manjula",

lastname:"devarala"

}

/\* let printname=person.printname.bind(person2); \*/

fullName = person.fullName.bind(member);

fullName();

Math.max():

Math.max(1,2,3);  // Will return 3

Math.max.apply(null, [1,2,3]); // Will also return 3

[https://www.geeksforgeeks.org/javascript-function-binding/](https://www.geeksforgeeks.org/javascript-function-binding/" \t "_blank" \o "https://www.geeksforgeeks.org/javascript-function-binding/)

HTTP request methods are **GET, POST, PUT, PATCH, and DELETE**

Closures, Hoisting

Difference between Let and const

Map, Filter, Reduce – Methods

**37. What are different DOM methods?**

A) HTML DOM getElementsByName() Method. HTML DOM getElementsByTagName() Method. HTML DOM query Selector () Method. HTML DOM querySelectorAll () Method.

**38. Difference between Undefined and not defined?**

A) The main difference between "undefined" and "not defined" is that **"undefined" is a value that can be assigned to a variable, while "not defined" indicates that a variable does not exist**

**39. Call-backs?**

A) A call back is **a function passed as an argument to another function**. This technique allows a function to call another function. A call back function can run after another function has finished.

**40. What is Propagation?**

A) Propagation refers to **how events travel through the Document Object Model (DOM) tree**.

**41.Event bubbling**?

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

const body = document.getElementsByTagName("body")[0]

const div = document.getElementsByTagName("div")[0]

const span = document.getElementsByTagName("span")[0]

const button = document.getElementsByTagName("button")[0]

body.addEventListener('click', () => {

console.log("body was clicked")

})

div.addEventListener('click', () => {

console.log("div was clicked")

})

span.addEventListener('click', () => {

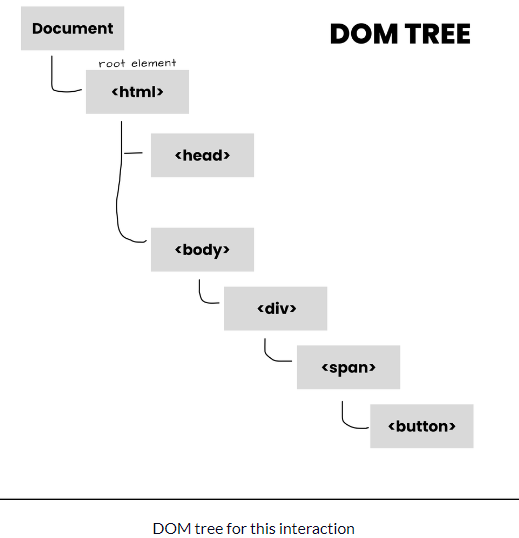
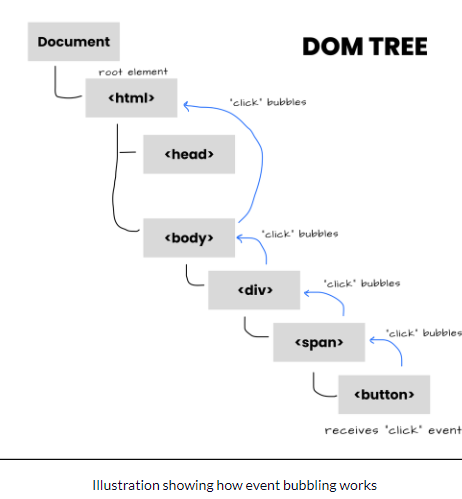
console.log("span was clicked")

})

button.addEventListener('click', () => {

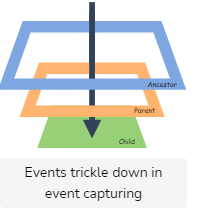
console.log("button was clicked")

})

**42.event Capturing?**

A) **Event capturing** is one of two ways to do event propagation in the HTML DOM. In event capturing, an event propagates from the outermost element to the target element. It is the opposite of **[event bubbling](https://www.educative.io/edpresso/javascript-event-bubbling" \t "_blank)**, where events propagate outwards from the target to the outer elements.



<html>

  <head></head>

  <body>

    <article id="ancestor" >

      article element

      <div id="parent" >

        div element

        <p id="child" >

          p element

        </p>

      </div>

    </article>

  </body>

</html>

<script>

  // Script to click event handler to capture on each element

  for(let elem of document.querySelectorAll('\*')) {

    elem.addEventListener("click", e => console.log("Capturing:", elem.tagName), true);

  }

</script>



**43. What is Currying?**

A) Currying is a technique in functional programming that performs the transformation of a function with multiple arguments into several functions containing a single argument in a sequence.

function curry(f) { // curry(f) does the currying transform

return function(a) {

return function(b) {

return f(a, b);

};

};

}

**44.**  <script>

        console.log('10'+10+10);

        console.log(10+10+'10');

    </script>

**Ans)** **101010**

**2010**

**45.** **<div>**

**<div class="name">**

**</div>**

**</div>**

**Ans)** **div {**

**display:flex;**

**justify-content:center;**

**align-self:center;**

**align-items:center;**

**height:100vh;**

**}**

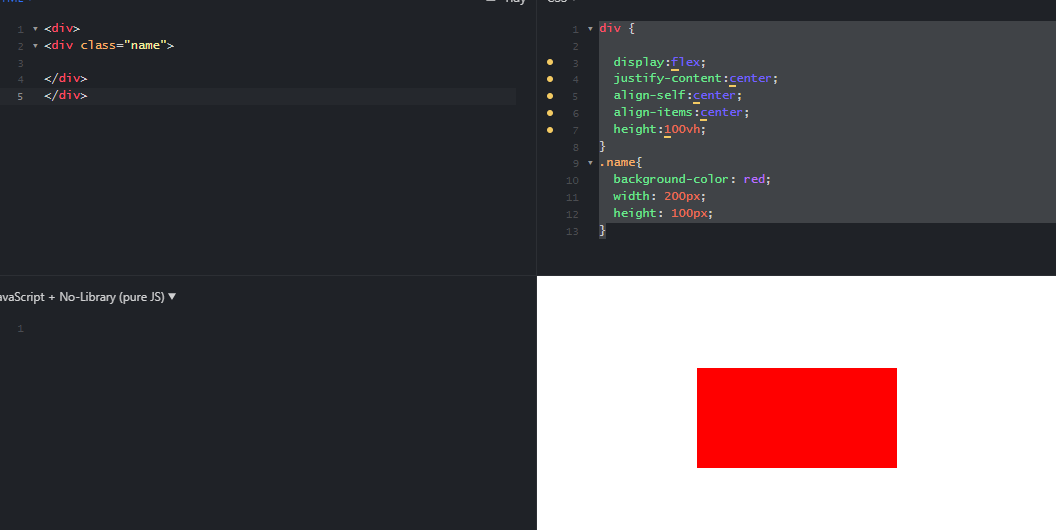
**.name{**

**background-color: red;**

**width: 200px;**

**height: 100px;**

**}**

****

**46.**   const num = [1,2,3,4,5];

         for(let i in num){

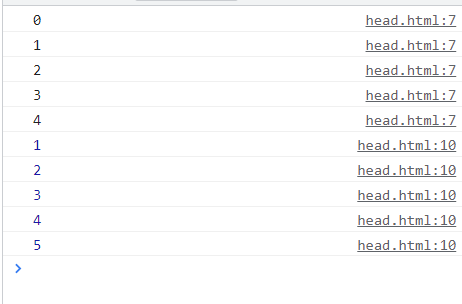
            console.log(i);

         }

         for(let i of num){

            console.log(i);

         }

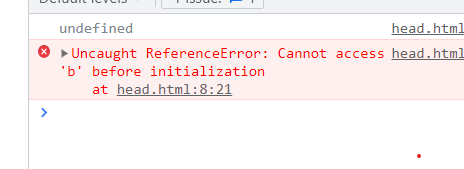
**Ans)** ****

 console.log(a);

        var a=5;

        console.log(b);

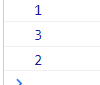
        let b =10;

Ans) 

**47.**  console.log(1);

        setTimeout(() => console.log(2),0);

        console.log(3);

**Ans)** ****

**48.**  const obj = {a:5,b:10};

        let obj1 = obj;

        obj1.c =15;

        console.log(obj.c);

**Ans)15**

**49. What is VCS?**

A)Vcs Defines Version control system tool and called as Git. Which means it store the history whatever changes we have done In Git

Denouncing and throttling?

If, else, if else, switch conditions

**50.** <html>

    <footer>2</footer>

    <header>1</header>

</html>

### Ans) 2

1

### **51.The Object Datatype**

The object data type can contain:

1. An object  
2. An array  
3. A date

**52.Javascript Onclick functionality**

<h2>My First JavaScript</h2>

<button type="button" onclick="document.getElementById('demo').innerHTML = Date()">

Click me to display Date and Time.</button>

<p id="demo"></p>

**53.What is the javascript event flow?**

**Event Target**

**Event Bubbling**

**Event Capturing**

**54. Why we Use JavaScript?**

JavaScript is used by programmers across the world **to create dynamic and interactive web content like applications and browsers,**

JavaScript is **commonly used for creating web pages**.

**55.What is JavaScript Throttling?**

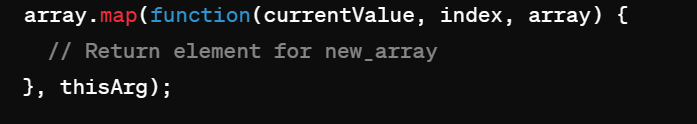
Throttling is used to call a function after every millisecond or a particular interval of time only the first click is executed immediately.

56.call ,apply ,bind javascript?

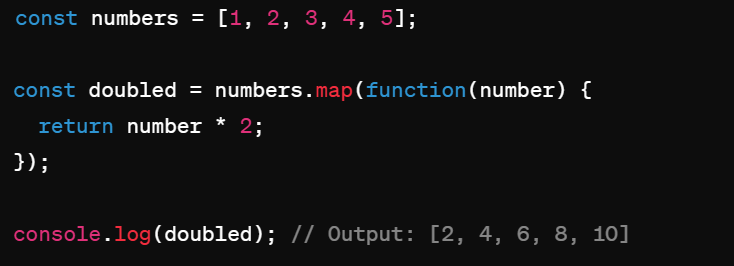
57.let ,var,const?

58.map?

Ans)  
The **map()** method in JavaScript is used to create a new array by applying a function to each element of an existing array. It doesn't change the original array; instead, it returns a new array with the results of applying the function to each element.

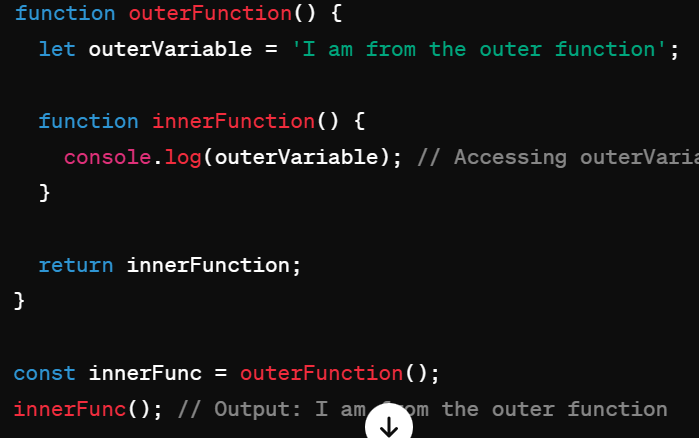


* **function(currentValue, index, array)**: A function to be called for each element in the array.
  + **currentValue**: The current element being processed in the array.
  + **index** (Optional): The index of the current element being processed in the array.
  + **array** (Optional): The array **map** was called upon.
* **thisArg** (Optional): A value to be passed to the function to be used as its "this" value.



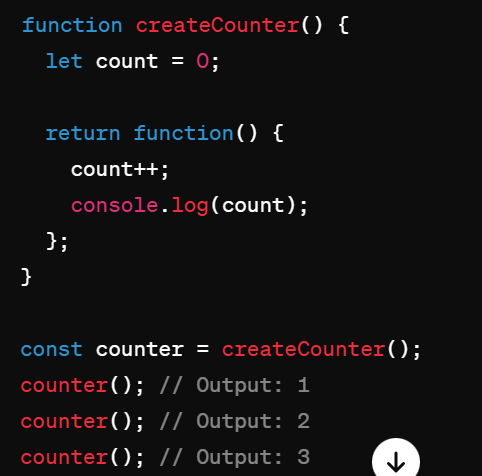
59.hosting?

60.closures?

Ans)  
A closure is a fundamental concept in JavaScript that allows a function to remember and access its lexical scope even when it's executed outside that scope. In simpler terms, a closure gives you access to an outer function's scope from an inner function.

In this example, **innerFunction** is a closure because it has access to the **outerVariable** defined in the **outerFunction**, even after **outerFunction** has finished executing.

Closures are powerful because they allow you to encapsulate data and behavior and use them in a controlled way. They are commonly used to create private variables and methods in JavaScript.

Ans)

61.undefined ?

Ans)  
In JavaScript, **undefined** and **not defined** are two different concepts:

**undefined**:

* + **undefined** is a special value in JavaScript that indicates that a variable has been declared but has not been assigned a value.
  + If a variable is declared but not initialized, or a function doesn't return a value, the variable or expression evaluates to **undefined**.

Example:

javascript

Copy code

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

**not defined**:

* + If you try to access a variable that has not been declared at all, JavaScript will throw a ReferenceError, indicating that the variable is **not defined**.

Example:

javascript

Copy code

console.log(y); // ReferenceError: y is not defined

In summary:

* **undefined** means a variable has been declared but has not been assigned a value.
* **not defined** means a variable has not been declared at all.

let x; console.log(x); // Output: undefined console.log(y); // ReferenceError: y is not defined

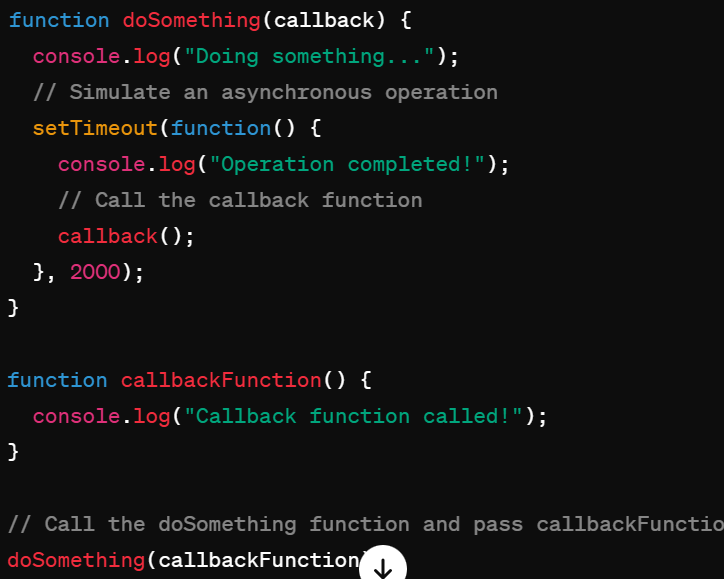
62.Not defined ?

63.memozation?

Ans)Memoization is an optimization technique used in computing to speed up the execution of functions by caching the results of expensive function calls and returning the cached result when the same inputs occur again.

In JavaScript, you can implement memoization using closures. Here's a simple example of memoization:

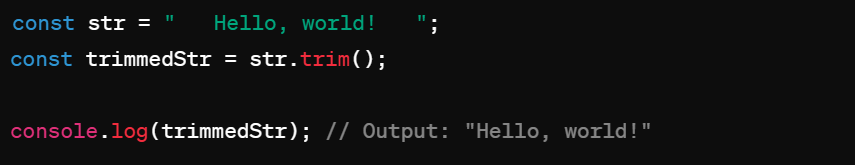
64.callback function?

Ans)  
A callback function is a function that is passed as an argument to another function and is executed after a particular event or task is completed. Callback functions are a fundamental concept in JavaScript, especially when dealing with asynchronous operations such as AJAX requests, event handling, and setTimeout.

65.trim()

Ans)The **trim()** method in JavaScript is used to remove whitespace from both ends of a string. Whitespace in this context includes spaces, tabs, and newlines.

Here's how you can use the **trim()** method:



In this example, the **trim()** method removes the leading and trailing spaces from the **str** string, leaving only the text **"Hello, world!"**.

66.positions differnce

* Ans)

1. ****Static Positioning (**position: static;**)****:
   * This is the default positioning value.
   * Elements are positioned according to the normal flow of the document.
   * The **top**, **right**, **bottom**, **left**, and **z-index** properties have no effect on statically positioned elements.

* ****Static****: Element is positioned according to the normal flow of the document.
* ****Relative****: Element is positioned relative to its normal position.
* ****Absolute****: Element is positioned relative to its nearest positioned ancestor.
* ****Fixed****: Element is positioned relative to the browser window.
* ****Sticky****: Element is positioned relative to the viewport and becomes fixed when a specified scroll position is reached.

67.flex properties

Ans)

Flexbox is a layout model in CSS that allows you to design flexible and responsive layouts more easily. To use Flexbox, you set the **display** property of an element to **flex**, and then you can use various properties to control the layout of its children.

Here are some of the most commonly used Flexbox properties:

**display: flex;**: This property is applied to the parent container to create a flex container.

**flex-direction: row | row-reverse | column | column-reverse;**: Specifies the direction of the flex container's main axis.

**flex-wrap: nowrap | wrap | wrap-reverse;**: Specifies whether the flex items should wrap or not if they overflow the flex container along the main axis.

**flex-flow: <flex-direction> || <flex-wrap>;**: A shorthand for setting both **flex-direction** and **flex-wrap** properties.

**justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly;**: Defines how flex items are aligned along the main axis of the flex container.

**align-items: stretch | flex-start | flex-end | center | baseline;**: Defines how flex items are aligned along the cross axis of the flex container.

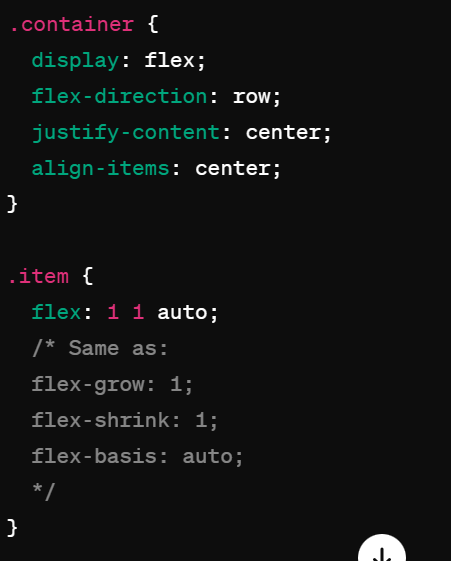
**align-content: flex-start | flex-end | center | space-between | space-around | stretch;**: Defines the alignment of flex lines within the flex container when there is extra space in the cross axis.

**flex-grow: <number>;**: Specifies how much a flex item will grow relative to the rest of the flex items inside the same container.

**flex-shrink: <number>;**: Specifies how much a flex item will shrink relative to the rest of the flex items inside the same container.

**flex-basis: <length> | auto;**: Specifies the initial size of the flex item before any available space is distributed according to the **flex-grow** and **flex-shrink** properties.

**flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]**: A shorthand for setting **flex-grow**, **flex-shrink**, and **flex-basis** properties at once.



68.meta tags

Ans)

**<meta charset="UTF-8">**: Specifies the character encoding for the HTML document.

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Sets the viewport width to the device width and sets the initial zoom level to 1. This is crucial for responsive web design.

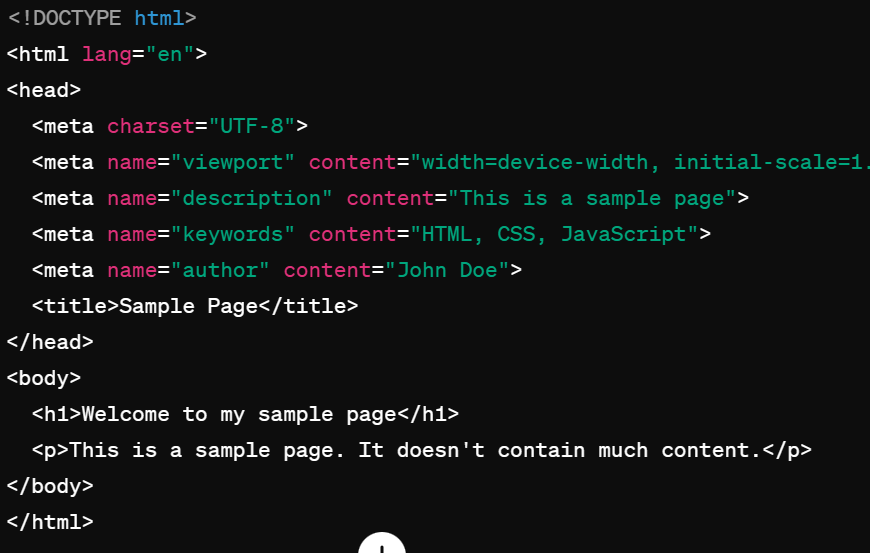
**<meta name="description" content="Description of the page">**: Provides a brief description of the page. This content is often displayed in search engine results.

**<meta name="keywords" content="keyword1, keyword2, keyword3, ...">**: Specifies keywords relevant to the page's content. Although not as important for search engine rankings as they once were, they can still be helpful.

**<meta name="author" content="Author Name">**: Specifies the author of the page.

**<meta name="robots" content="index, follow">**: Instructs search engine

crawlers to index the page and follow its links. Other common directives include "noindex" and "nofollow".

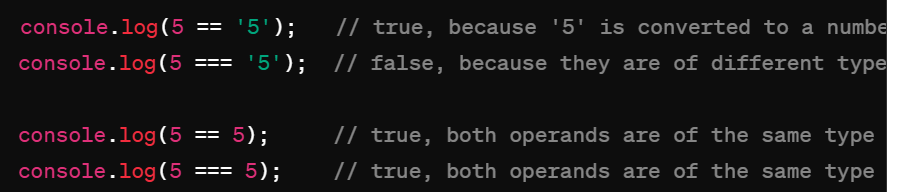


69.Diff between “==” and “===”’ ? Which is true?

In JavaScript, both **==** and **===** are comparison operators used to compare values.

**==** (Equality Operator): It compares the equality of two values after type conversion. If the values are of different types, JavaScript attempts to convert them to the same type before making the comparison.

**===** (Strict Equality Operator): It also compares the equality of two values, but without type conversion. The values must be of the same type to be considered equal.



70.What is throttling?

A) Throttling is **a mechanism in Intel® Processors to reduce the clock speed when the temperature in the system reaches above TJ Max (or Tcase)**. This is to protect the processor and to indicate to the user that there is an overheating issue in their system that they need to monitor.

Throttling is a technique used in web development to control the rate at which a function is executed. It limits the number of times a function can be called over a specified period. Throttling is often used to optimize performance and prevent excessive function calls, especially in situations like event handling and API requests.

There are two main approaches to throttling:

****Time-based throttling****: This approach ensures that a function is called at most once in a given time interval. If the function is called multiple times within the specified interval, only the first call is executed, and subsequent calls are ignored until the interval has elapsed.

****Request-based throttling****: This approach limits the number of times a function can be called within a certain number of requests. Once the limit is reached, further function calls are ignored until the request count is reset.

Here's a simple example of time-based throttling using JavaScript:



In this example, the **throttle** function takes another function (**func**) and a delay time (**delay**) as parameters. It returns a new function that wraps the original function. This new function can only be called once in the specified delay period. If it is called multiple times within the delay, subsequent calls are ignored until the delay has elapsed.

1. why we use javascript?

Ans)JavaScript is a versatile programming language primarily used for web development, though it has applications beyond just web pages. Here are several reasons why JavaScript is commonly used:

****Client-Side Interactivity****: JavaScript enables dynamic, interactive elements on websites, allowing for enhanced user experience. It can respond to user actions like clicks, scrolls, and keyboard input without needing to reload the entire page.

****Cross-Browser Compatibility****: JavaScript is supported by all major web browsers, making it a reliable choice for web development.

****Rich Interfaces****: JavaScript, along with HTML and CSS, is used to create modern, feature-rich user interfaces, including animations, transitions, and complex visual effects.

****Asynchronous Communication****: JavaScript allows for asynchronous communication with a web server using AJAX (Asynchronous JavaScript and XML), enabling web pages to request and receive data from servers without needing to reload the entire page.

****Server-Side Development****: With the introduction of Node.js, JavaScript can now be used for server-side development as well. Node.js allows developers to build scalable and high-performance server-side applications using JavaScript.

****Libraries and Frameworks****: JavaScript has a vast ecosystem of libraries and frameworks like React.js, Angular, and Vue.js, which simplify and speed up the development process by providing pre-built components and functionalities.

****Versatility****: JavaScript is a versatile language that can be used for a wide range of tasks, including web development, mobile app development (using frameworks like React Native and Ionic), desktop application development (using frameworks like Electron), game development, and more.

****Community and Support****: JavaScript has a large and active community of developers, which means that there are plenty of resources, tutorials, and libraries available to help developers build and maintain their projects.

Overall, JavaScript is an essential tool for web developers, allowing them to create dynamic, interactive, and feature-rich web applications.

1. Javascript event flow?

Ans)JavaScript event flow refers to the order in which events are triggered and handled in the Document Object Model (DOM) hierarchy. There are two main phases in the JavaScript event flow:

****Event Capturing Phase****:

* + During this phase, the event is captured by the outermost element and propagates through the DOM hierarchy down to the target element.
  + Event handlers attached to elements during this phase will execute before the event reaches its target.

****Event Bubbling Phase****:

* + During this phase, the event bubbles up from the target element through the DOM hierarchy to the outermost element.
  + Event handlers attached to elements during this phase will execute after the event reaches its target.

73.javascript json?

**JSON** is a format for storing and transporting data. **JSON** is often used when data is sent from a server to a web page.

74.version

75.angular use

76.commands

75.pure and impure pipes?

Ans)A **pure pipe** is only called when Angular detects a change in the value or the parameters passed to a pipe. An **impure pipe** is called for every change detection cycle no matter whether the value or parameter(s) changes.

76.immutable?

77.annotions?

Ans) An annotation is extra information associated with a particular point in a document or other piece of information. It can be a note that includes a comment or explanation. Annotations are sometimes presented in the margin of book pages

**78.** annotations in angular?

Ans) Annotation is **a block of text that can be displayed over a node or connector**. Annotation is used to textually represent an object with a string that can be edited at runtime. Multiple annotations can be added to a node/connector.

79.semantic tags?

* <header>: t defines a header for a web page.
* <nav>: It defines a container for navigation links.
* <section>: This defines a section in a web page.
* <article>: This element contains the main part, containing information about the web page.
* <aside>: The <aside> content is often placed as a sidebar in a document.
* <footer>: It defines a footer for a document or a section.

Ans) Examples of **non-semantic** elements: <div> and <span> - Tells nothing about its content.

Examples of **semantic** elements: <form>, <table>, and <article> - Clearly defines its content.

* <article>
* <aside>
* <details>
* <figcaption>
* <figure>
* <footer>
* <header>
* <main>
* <mark>
* <nav>
* <section>
* <summary>
* <time>

## 80. Nesting <article> in <section> or Vice Versa?

The <article> element specifies independent, self-contained content.

The <section> element defines section in a document.

Can we use the definitions to decide how to nest those elements? No, we cannot!

So, you will find HTML pages with <section> elements containing <article> elements, and <article> elements containing <section> elements.

**81. What are mixins in CSS?**

The @mixin directive **lets you create CSS code that is to be reused throughout the website**. The @include directive is created to let you use (include) the mixin.

### **[Sass @mixin and @include - W3Schools](https://www.w3schools.com/sass/sass_mixin_include.php)**

**82. event bubbling?**

**<https://javascript.info/bubbling-and-capturing>**

**83.box model?**

     .cleardu{

                width: 100px;

                height: 100px;

                padding:10px;

                box-sizing: border-box;

                border:2px solid #000;

                background-color: rgb(66, 194, 7);

            }

            .cleardu1{

                width: 100px;

                height: 100px;

                padding:10px;

                box-sizing: content-box;

                border: 2px solid #000;

                background-color: aqua;

            }

            .cleardu2{

                width: 100px;

                height: 100px;

                padding:10px;

                border: 2px solid #000;

                background-color: rgb(177, 58, 58);

            }

        </style>

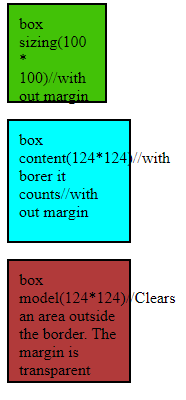
    </head>

    <body>

        <p class="cleardu">box sizing(100 \* 100)//with out margin</p>

        <p class="cleardu1">box content(124\*124)//with borer it counts//with out margin</p>

        <p class="cleardu2">box model(124\*124)//Clears an area outside the border. The margin is transparent</p>

****

**84.box sizing?**

# 85. **CSS Pseudo-classes?**

Ans) A pseudo-class is used to define a special state of an element.

/\* unvisited link \*/  
a:link {  
  color: #FF0000;  
}  
  
/\* visited link \*/  
a:visited {  
  color: #00FF00;  
}  
  
/\* mouse over link \*/  
a:hover {  
  color: #FF00FF;  
}  
  
/\* selected link \*/  
a:active {  
  color: #0000FF;  
}

## Pseudo-classes and HTML Classes

a.highlight:hover {  
  color: #ff0000;  
}

## CSS - The :first-child Pseudo-class

p:first-child {  
  color: blue;  
}

## CSS - The :lang Pseudo-class

q:lang(no) {  
  quotes: "~" "~";  
}  
</style>  
</head>  
<body>  
  
<p>Some text <q lang="no">A quote in a paragraph</q> Some text.</p>  
  
</body>

## 86. What are Pseudo-Elements?

## Ans) A CSS pseudo-element is used to style specified parts of an element.

## 87. What are Pseudo-Elements?

## Ans) selector::pseudo-element {   property: value; }

## The ::first-line Pseudo-element

## p::first-line {   color: #ff0000;   font-variant: small-caps; }

## p::first-letter {   color: #ff0000;   font-size: xx-large; }

## h1::before {   content: url(smiley.gif); }

## h1::after {   content: url(smiley.gif); }

## ::selection {   color: red;   background: yellow; }

## 88.px ,%, vw/vh ,rem, em?

## Px(Absolute size)

## % ( Realated to parent width/height)

## Vw/vh (Realated to sreensize)

## Rem ( related to root element font)

## Em ( related to its font size)

**89.** **The flex container properties are:**

* [flex-direction](https://www.w3schools.com/css/css3_flexbox_container.asp" \l "flex-direction)
* [flex-wrap](https://www.w3schools.com/css/css3_flexbox_container.asp" \l "flex-wrap)
* [flex-flow](https://www.w3schools.com/css/css3_flexbox_container.asp" \l "flex-flow)
* [justify-content](https://www.w3schools.com/css/css3_flexbox_container.asp" \l "justify-content)
* [align-items](https://www.w3schools.com/css/css3_flexbox_container.asp" \l "align-items)
* [align-content](https://www.w3schools.com/css/css3_flexbox_container.asp" \l "align-content)

## 90. conditional ternary operator?

## Ans) The conditional (ternary) operator is the only JavaScript operator that takes three operands: a condition followed by a question mark ( ? ), then an expression to execute if the condition is truthy followed by a colon ( : ), and finally the expression to execute if the condition is falsy.

**91.Data types?**

**JavaScript Data Types** · 1. String · 2. Number · 3. Bigint · 4. Boolean · 5. Undefined · 6. Null · 7. Symbol · 8. Object ...

**92. Doctype?**

### **[HTML doctype declaration - W3Schools](https://www.w3schools.com/tags/tag_doctype.asp)**

[https://www.w3schools.com › tags › tag\_doctype](https://www.w3schools.com/tags/tag_doctype.asp)

All **HTML** documents must start with a <!**DOCTYPE**> declaration. The declaration is not an **HTML** tag. It is an "information" to the browser about what document type ...

**93.What is <! DOCTYPE with example?**

DOCTYPE> tag is used to inform the browser about the version of HTML used in the document. It is called as the document type declaration (DTD).  
...  
Syntax.

|  |  |
| --- | --- |
| **Display** | **None** |
| Start tag/End tag | Start tag only |
| Usage | Structural |

## 94. ****What are some of the common lists that can be used when designing a page?****

**Ans)** You can insert any or a combination of the following list types:  
– ordered list  
– unordered list  
– definition list  
– menu list  
– directory list

– ordered list

 <ul>

        <li>C</li>

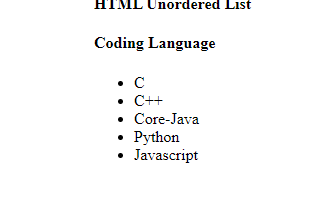
        <li>C++</li>

        <li>Core-Java</li>

        <li>Python</li>

        <li>Javascript</li>

    </ul>



<h4>Coding Language</h4>

    <ul style="list-style-type:square;">

        <li>C</li>

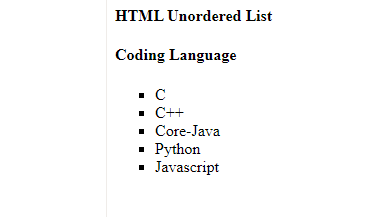
        <li>C++</li>

        <li>Core-Java</li>

        <li>Python</li>

        <li>Javascript</li>

    </ul>



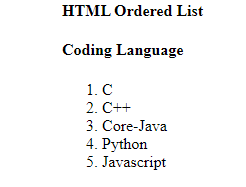
<ol>

<li>Item1</li>

<li>Item2</li>

<li>Item3</li>

</ol>



 <ol type="I">

        <li>C</li>

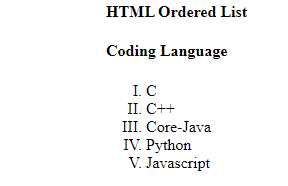
        <li>C++</li>

        <li>Core-Java</li>

        <li>Python</li>

        <li>Javascript</li>

    </ol>



 <ol start="12">

        <li>C</li>

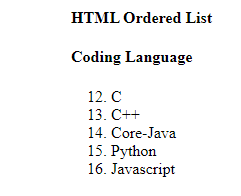
        <li>C++</li>

        <li>Core-Java</li>

        <li>Python</li>

        <li>Javascript</li>

    </ol>



**Description List:**A description list is a type of list where each item has a description. It is also known as a definition list. The <dl> tag is used to create description list, the <dt> tag defines the item, and the <dd> tag describes each item in list.

**Syntax:**

<dl> Contents... </dl>

The HTML definition list contains following 3 tags:

* **[<dl>](https://www.geeksforgeeks.org/html-dl-tag/)**: It defines the start of the list.
* **[<dt>](https://www.geeksforgeeks.org/html-dt-tag/)**: It defines a item.
* **[<dd>](https://www.geeksforgeeks.org/html-dd-tag/):** It defines the description of each item.

 <h4>Description List</h4>

    <strong>Developers Life</strong>

    <dl> <dt>Code</dt>

        <dd>- Code all day!</dd>

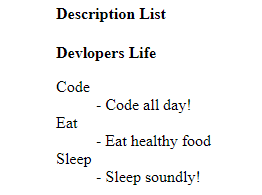
            <dt>Eat</dt>

        <dd>- Eat healthy food</dd>

            <dt>Sleep</dt>

        <dd>- Sleep soundly!</dd>

    </dl>



**95.** function cName(){

            return;

            {

                mes: "javascript"

            }

        }

        console.log(cName());

**Ans)undefined (y because return after semicolon )**

**96.**  function cName(){

            return{

                mes: "javascript"

            }

        }

        console.log(cName());

**Ans)** ****

**97.**  function cName(){

            return

{//if I use flower bracket is in down

                mes: "javascript"

            }

        }

        console.log(cName());

**Ans)undefined (y because**

**98.**  const numbers =[

            1,

            [3, [2, 8, [12]], 9]

            [5],

            [12 [[5]]],

            [100, [23, 45]]

        ];

**Ans)[1, 3, 2, 8, 12, 9, 5, 12, 5, 100, 23, 45]**

**99.what is Software**

**Ans)**Software is a program or collection of programs which will solve particular problem.

Types of software:

1.standalone app –in our system only login done (means it will work only our system ).

2.client-server – some of the application run to another system is called client ap

**100.** **Definition of HTTP**

**Ans)** (**HyperText Transfer Protocol**) The communications protocol used to connect to Web servers on the Internet or on a local network (intranet).

**101.** Bootstrap CDN

If you don't want to download and host Bootstrap yourself, you can include it from a CDN (**Content Delivery Network**).

**102. Top 10 Features of ES6**

* let and const keywords :
* Arrow Functions.
* Multi-line Strings.
* Default Parameters.
* Template Literals.
* Destructuring Assignment.
* Enhanced Object Literals.
* Promises.

**103.Type infernce**

Ans) The type of the x variable is inferred to be number. This kind of inference takes place when initializing variables and members, setting parameter default values, and determining function return types.

In most cases, type inference is straightforward.

Example1

let x = 3;

let x: number

Example2

let x = [0, 1, null];

let x: (number | null)[]

**104. What are annotations in Angular?**

Annotation is **a block of text that can be displayed over a node or connector**. Annotation is used to textually represent an object with a string that can be edited at runtime. Multiple annotations can be added to a node/connector.

Example:

@ComponentAnnotation

import {

  ComponentAnnotation as Component,

} from '@angular/core';

export class ComponentAnnotation extends DirectiveMetadata {

  constructor() {

  }

}

**105.enum**

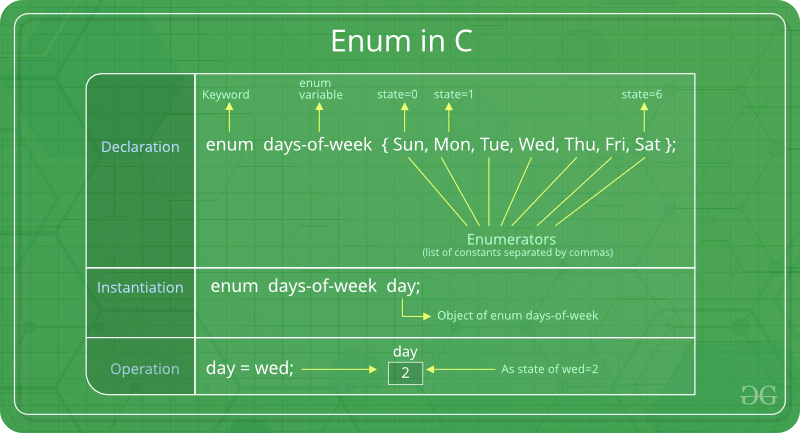
enum emp\_typ { designer=92,developer=94,tester=95,architecture=97,sysadmin=98,netadmin=99};

        var myDes=emp\_typ.netadmin;

        console.log(myDes);

        console.log(emp\_typ[95]);//output tester:95

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.



**106.function passeing in annotation?**

Ans)

function add(x:number,y:number,z?:number):number{

        if(z!=undefined){

            return x+y+z;

        }

        else{

        return x+y;

        }

    }

    var res = add(10,20);

    console.log(res);

    var res2:number;//output:30

    res2=add(45,56,55);

    console.log(res2)//output:156

107. What is void in Angular?

Ans)Similar to languages like Java, void is **used where there is no data**. For example, if a function does not return any value then you can specify void as return type. There is no meaning to assign void to a variable, as only null or undefined is assignable to void.

# **108. TypeScript Data Type – Void**

Ans) Similar to languages like Java, void is used where there is no data. For example, if a function does not return any value then you can specify void as return type.

Example:

function sayHi(): void {

console.log('Hi!')

}

let speech: void = sayHi();

console.log(speech); //Output: undefined

There is no meaning to assign void to a variable, as only null or undefined is assignable to void.

Exampple:

let nothing: void = undefined;

let num: void = 1; // Error

**109.**

//suppose if i give more parameters is called rest paramaters

    function add(...x){//three dots 3 called spread operators

        console.log(x);//output (3) [false, 34, 55]

    }

    add(12, 34, 55);

    add(11,22,33,44,55,66,77);

110.spread operator compare to normal hard code If I use es6 property spread operator easy

 //very hard code

    var x=[11,22,33];

    var y=[44,55,66];

    var z=[x,y];

    console.log(z);//output[ [11, 22, 33], [ 44, 55, 66] ]

 //(2) [Array(3), Array(3)]

    0: (3) [11, 22, 33]

    1: (3) [44, 55, 66]

    //

    //hard code

    var x=[11,22,33];

    var y=[44,55,66];

    var z=[x[0],x[1],x[2],y[0],y[1],y[2]];

    console.log(z);//output[ [11, 22, 33, 44, 55, 66] ]

//

    //

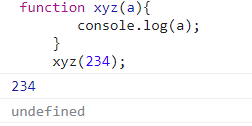
    //pure spread operator(very simple)

    var w = [...x,...y];

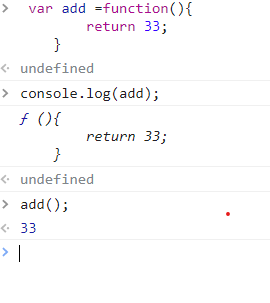
    console.log(w);//output[ [11, 22, 33, 44, 55, 66] ]

//(7) [11, 22, 33, 44, 55, 66, 77]

111.



112.



113.call back function?

Def:a callback function is a function passed into another function as an argument, which is then invoked inside the outer function come kind of router or action.

  //we can pass functions to another function and exceuted

     var add =function(){

        consle.log("hello")

     }

     function xyz(a){

        a();//is called call back function

     }

     xyz(add);

     //output:hello

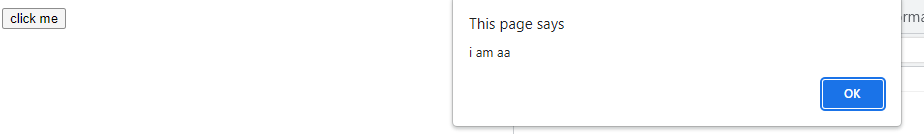
114. callback function examples

Ans) when I click alert mssg will come

Example1: function aa(){

        alert("i am aa");

     }



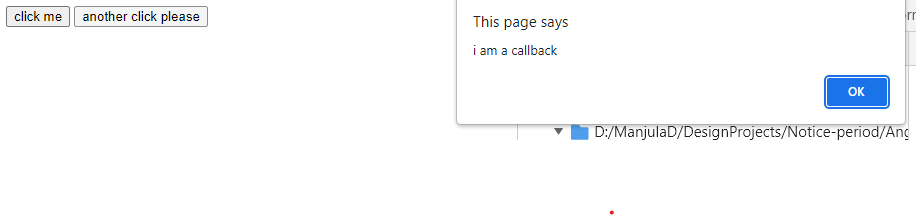
Example:2

document.getElementById('b1').addEventListener('click',function(){

        alert("i am a callback");

     });

Output:



Example3: passing arguments to callback function

var x=[10,20,30,40,50,60];

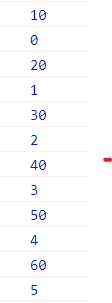
     var y=x.map(function(a,b){

        console.log(a);

        console.log(b);

     })

Output:



Example :using map in callback function

Ans)

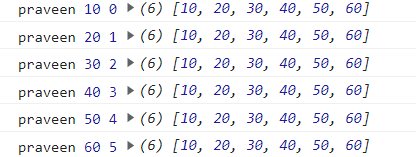
 var x=[10,20,30,40,50,60];

     var y=x.map(function(a,b,c){

        console.log("praveen",a,b,c);

         })

Output:



115.Higher order functions in javascript?

Ans) **a function which takes another function as an argument or returns a function** is known as a higher order function. Let's deep dive a bit to see both types of implementation, that is: Passing a function as an argument to another function. Returning a function from another function.

1)every()

2)filter()

3)ForEach()

4)reduce()

5)map()

6) setTimeout()var

**a)every():**Checks if every element in an array pass a test

Notes: for every value of x this function will called.

every function based on return type it desides iterate or not iterate alo desided

## Definition and Usage

The every() method executes a function for each array element.

The every() method returns true if the function returns true for all elements.

The every() method returns false if the function returns false for one element.

The every() method does not execute the function for empty elements.

The every() method does not change the original array

116. var x=[12,23,34,45,56,67,78]

     var k=x.every(function(){

        console.log('hello');

     })

     //output: hello

117.     var x=[12,23,34,45,56,67,78]

     var k=x.every(function(){

        console.log('hello');

        return true;//7 times its executed that is y 7 hello will get

     })

     //output: 7hello

118.

     var x=[12,23,34,45,56,67,78]

     var k=x.every(function(a,b,c){

        console.log('hello',a,b,c);

        return true;

     })

     console.log(k);//true

119.  var x=[12,23,34,45,56,67,78]

     var k=x.every(function(a,b,c){

        if(a==34){

            return false;

        }

        return true;

     })

     console.log(k);//false

120. var x=[12,23,34,45,56,67,78]

     var k=x.every(function(a,b,c){

        if(a==NaN){

            return false;

        }

        return true;

     })

     console.log(k);//true

121.

var x=[12,23,34,'pp',45,56,67,78]

     var k=x.every(function(a,b,c){

        if(isNaN(a)){

            return false;

        }

        return true;

     })

     console.log(k);//false pp

**b) forEach() :** Calls a function for each array element

 1) var x=[12,23,34,'pp',45,56,67,78]

     var k=x.forEach(function(a,b,c){

       console.log(a);

     })

Output:



122.  var x=[12,23,34,'pp',45,56,67,78]

     var k=x.forEach(function(a,b,c){

       console.log(a);

       return true;

     })

Output:



**c)filter():** Creates a new array with every element in an array that pass a test

123.  var x=[12,23,34,'pp',45,56,67,78]

     var k=x.filter(function(a,b,c){

        if(isNaN(a)){

            return false;

        }

        return true;

     })

     console.log(k);

output:

2)

 var x=[12,23,34,'pp',45,56,67,78]

     var k=x.filter(a=>isNaN(a)?false:true);

     console.log(k);

output:



**c)reduce():** Reduce the values of an array to a single value (going left-to-right)

## Definition and Usage

The reduce() method executes a reducer function for array element.

The reduce() method returns a single value: the function's accumulated result.

The reduce() method does not execute the function for empty array elements.

The reduce() method does not change the original array.

124.  var x=[12,23,34,45,56,67,78];

    var k=x.reduce(function(a){

        console.log(a);

    })

Output:



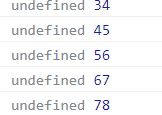
125.  var x=[12,23,34,45,56,67,78];

    var k=x.reduce(function(a,b){

        console.log(a,b);

    });

Output:



126. var x=[12,23,34,45,56,67,78];

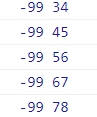
    var k=x.reduce(function(a,b){

        console.log(a,b);

        return -99;

    });

Output:



127.var x=[12,23,34,45,56,67,78];

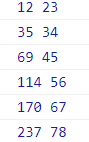
    var k=x.reduce(function(a,b){

        console.log(a,b);

        return a+b;

    });

Output:



128. var x=[12,23,34,45,56,67,78];

    var k=x.reduce(function(a,b){

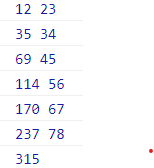
        console.log(a,b);

        return a+b;

    });

    console.log(k);

output:



129.  var w = x.reduce((a,b)=>a+b);

    console.log(w);

output:315

**lamda function:**

var x = [12,23,45,67,88];

var k = x.reduce((a,b)=>{

console.log(a,b);

return a+b;

})

**Example:**

var x = [12,23,45,56,67]

var k = x.reduce((a,b)=>a+b);

console.log(k)//203

**d)map():** Creates a new array with the result of calling a function for each array element

important notes:

a)it will be stored in new array

b)Map returns an array

1) var x=[12,23,34,45,56,67,78];

    var k=x.map(function(a,b){

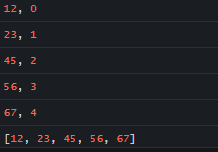
        console.log(a,b);

        return a;

    });

    console.log(k);

output:



130.  var x=[12,23,34,45,56,67,78];

    var k=x.map(function(a,b){

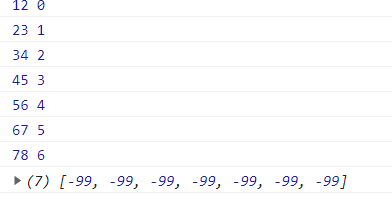
        console.log(a,b);

        return -99;

    });

    console.log(k);

Output:



131. var x=[12,23,34,45,56,67,78];

    var k=x.map(function(a,b){

        console.log(a,b);

        return a+4;

    });

    console.log(k);

output:

4)

e)setTimeout also a higher order

1) setTimeout(function abc(){

        alert("hello");

    },5000

    )

2)  setInterval(function(){

        alert("hello");

    },5000

    )

3)    console.log("hello")

   setTimeout(function(){

    console.log("hellooooooooooooooo")

   },2300);

   var si=setTimeout(function(){

    console.log("example of setinterval")

   },4000);

   document.getElementById("b1").addEventListener("click",function(){

    clearInterval(si);

   });

**132.normal function?**

Ans)  var f1 = function(x,y){

            var z;

            z=x+y;

            return z;

        }

        var res=f1(12,23);

        console.log(res);

output:35

**133.arrow function?**

 var f1 = (x,y)=>{

            var z;

            z=x+y;

            return z;

        }

        var res=f1(12,23);

        console.log(res);

output:35

134.

  var add = (x,y)=>(return x+y;);

        var add1 = (x,y)=>x+y;

        var res2 = add(10,20);

        console.log(res2);//30

        console.log(add1(23,34));//57

        var inc = x=>++x;

        console.log(inc(12));//13

        var inc2 =  function(x){

            return x++;

        }

**135.higher order function?**

A function which can writen function then we called as high order function

var f1 = function(){

                var f11=function(){

                    console.log("i am f11");

                }

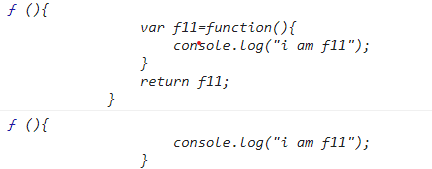
                return f11;

            }

            console.log(f1);

  console.log(f1());

**output:**

****

**136.**    var f1 = function(){

                var f11=function(){

                    console.log("i am f11");

                }

                return f11();

            }

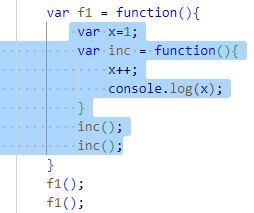
            console.log(f1);

            console.log(f1());

**Output:**

****

**137.clousers?**



Marked blue background called clouser

 /\* starts here\*/

            var x=1;

            f1 = function(){

                x++;

                console.log(x)

            }

            f1();//2

            f1();//3

            /\*ends here\*/

  138./\* starts here\*/

            f1 = function(){

                var x=1;

                x++;

                console.log(x)//every time x is reintialzed

            }

            f1();//2

            f1();//2

            /\*ends here\*/

 139./\* starts here\*/

             var x=1;

            f1 = function(){

                x++;

                console.log(x)

            }

            f1();//2

            f1();//3

            f2= function(){

                x++;

                console.log(x);

            }

            f2();//4

            /\*ends here\*/

**140.enhancing our components with pipes and Directives**

1. Directives in angular
2. Form related
3. Routing directives

**Directives in angular**

* + Ngclss
  + Ngstyle
  + Ngif
  + Ngfor
  + Ngswitch

**141.** onclick button textshould be hide and show?

<div id="m1">

    <h1 class="">angular26 works!</h1>

    <button (click)="chg()">click here</button>

    <h2 \*ngIf="flag">some text</h2>

</div>

**.ts**

 flag=false;

chg(){

    this.flag=!this.flag;

  }

**\*ngIf:**

<!--step1 starts here-->

<div id="m1">

  <h1>my heading</h1>

  <button (click)="chg()">click</button>

  <h1 \*ngIf="flag">Some text</h1>

</div>

<!--step1 eds here-->

**.css**

#m1 {

  border: 2px solid;

  padding: 10px;

  margin: 15px;

  background-color: blueviolet;

}

**.ts**

flag;

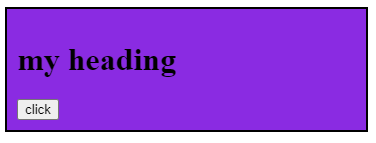
chg() {

    this.flag = !this.flag;

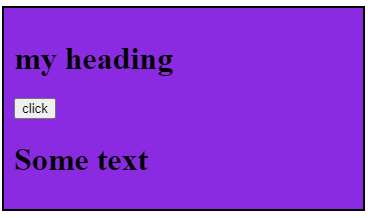
  }

**Output:**

**before clilck**

****

**After clicking**

****

<!--step2 starts here-->

<div class="m1">

  <input type="checkbox" [(ngModel)]="acc" />I agree<br />

  {{ acc }}

  <button>click to know</button>

</div>

<!--step2 eds here-->

**.ts**

 acc;

**output:**

****

  <!--step3 starts here-->

  <div class="m1">

    <input type="checkbox" [(ngModel)]="acc" />I agree<br />

    {{ acc }}

    <button [disabled]="acc" (click)="abc(acc.value)">click to know</button>

  </div>

  <!--step3 eds here-->

**.ts**

  acc;

****

<!--step4 starts here-->

  <div class="m1">

    <div>

        <input type="text" [(ngModel)]="x">

        <h1 \*ngIf="x%2==0">Its an even number</h1>

        <h1 \*ngIf="x%2!=0">Its an odd number</h1>

    </div>

</div>

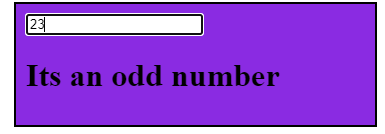
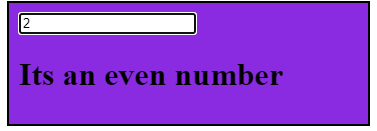
<!--step4 ends here-->

**.ts**

 abc(x: any){

    alert(x);

  }

**** ****

**Array Methods:**

**142.Splice():**

  //1) splice methods

            var ar1 = [10,20,30,40,50];

            ar1.splice(3,2,66);

            console.log(ar1);

            //output:(4) [10, 20, 30, 66]

  //Example:

            var ar2=[10,20,30,40,50,60]

            ar2.splice(3,2,0);

            console.log(ar2);

            //output:[10, 20, 30, ,0, 60]

            //example:

            var ar2=[10,20,30,40,50,60]

            ar2.splice(1);

            console.log(ar2);

            //output:[10]

//Example:

            var ar2=[10,20,30,40,50,60]

            ar2.splice(1,0);

            console.log(ar2);

            //output:[10, 20, 30, 40, 50, 60]

//Example:

            var ar2=[10,20,30,40,50,60]

            ar2.splice(1,3);

            console.log(ar2);

            //output:[10,50,60]

**143.Push():**

   //2) push methods

            var ar2 = [10,20,30,40,50,70];

            ar2.push(60);// after last index adding 60

            console.log(ar2);

             //output:(7) [10, 20, 30, 40, 50, 70, 60]

**144.Pop():**

 //3) pop methods

            var ar3 = [10,20,30,40,50,60,70];

            ar3.pop();//remove last index 70

            console.log(ar3);

             //output:(6) [10, 20, 30, 40, 50, 60]

**145.Shift():**

//4) shift methods

            var ar4 = [10,20,30,40,50];

            ar4.shift();//remove first element

            console.log(ar4);

             //output:(4) [20, 30, 40, 50]

**146.Unshift():**

//5) unshift methods

            var ar4 = [10,20,30,40,50,60];

            ar4.unshift(33);//adding 33 to first element

            console.log(ar4);

             //output:(7) [33, 10, 20, 30, 40, 50, 60]

notes:

            //push:adding last element(60)

            //pop:remove last element

            //shift:remove first element

            //unshift:adding first element(33)

**147.forEach():function for each array element**

**Notes:it wont check return true or false**

//6 forEach method

            //step1

            var ar6 = [10,20,30,40,50]

            ar6.forEach(function(){

                console.log("i am ar2");

            })

            //output:

            /\*"i am ar2"

            "i am ar2"

            "i am ar2"

            "i am ar2"

            "i am ar2"\*/

  //step2 passing arguments

            var ar6 = [10,20,30,40,50]

            ar6.forEach(function(a){

                console.log("i am ar2",a);

            })

            //output:

            /\*

            "i am ar2", 10

            "i am ar2", 20

            "i am ar2", 30

            "i am ar2", 40

            "i am ar2", 50

            \*/

//step3 passing arguments

             var ar6 = [10,20,30,40,50]

            ar6.forEach(function(a,b,c){

                console.log("i am ar2",a,b,c);

            })

            //output:

            /\*

            "i am ar2", 10, 0, [10, 20, 30, 40, 50]

            "i am ar2", 20, 1, [10, 20, 30, 40, 50]

            "i am ar2", 30, 2, [10, 20, 30, 40, 50]

            "i am ar2", 40, 3, [10, 20, 30, 40, 50]

            "i am ar2", 50, 4, [10, 20, 30, 40, 50]

            \*/

**Important Notes:**

/forEach is called callback function (with out using for loop we can process entire data)

//print even numbers

             var ar6 = [12,23,34,45,56,67,78,89,90]

            ar6.forEach(function(a,b,c){

                if(a%2==0){

                console.log(a);

                }

            })

            //output:12 34 56 78 90

//normal processs to print even numbers or this pure functioal programing language

            var ar6 = [12,23,34,45,56,67,78,89,90]

            for(i=0;i<ar6.length;i++){

                if(ar6[i]%2==0){

                console.log(ar6[i]);

                }

            }

            //output:12 34 56 78 90

**148.Every():it will check return true or false**

  /\* starts here\*/

     var x=[12,23,34,45,56,67,78]

     var k=x.every(function(){

        console.log('hello');

        return true;

     })

     //output: hello

     /\*ends here\*/

     var x=[12,23,34,45,56,67,78]

     var k=x.every(function(){

        console.log('hello');

        return true;

     })

     //output: 7hello

     /\*ends here\*/

    /\* starts here\*/

     var x=[12,23,34,45,56,67,78]

     var k=x.every(function(a,b,c){

        console.log('hello',a,b,c);

        return true;

     })

     console.log(k);//true

    /\*ends here\*/

     /\* starts here\*/

    var x=[12,23,34,45,56,67,78];

     var k=x.every(function(a,b,c){

        if(a==34){

            return false;

        }

        return true;

     })

     console.log(k);//false

     /\*ends here\*/

    /\* starts here\*/

    var x=[12,23,34,45,56,67,78]

     var k=x.every(function(a,b,c){

        if(a==NaN){

            return false;

        }

        return true;

     })

     console.log(k);//true

     /\*ends here\*/

     /\* starts here\*/

     var x=[12,23,34,'pp',45,56,67,78]

     var k=x.every(function(a,b,c){

        if(isNaN(a)){

            return false;

        }

        return true;

     })

     console.log(k);//false pp

/\*ends here\*/

**149.Filter():for**

//filter() array methods starts here

            /\*starts here\*/

            var x=[12,23,34,'pp',45,56,67,78]

            var k=x.filter(function(a,b,c){

                if(isNaN(a)){

                    return false;

                }

                return true;

            })

            console.log(k);

            /\*Ends here \*/

            /\*starts here\*/

            var x=[12,23,34,'pp',45,56,67,78]

            var k=x.filter(a=>isNaN(a)?false:true);

            console.log(k);

            /\*Ends here \*/

            //filter() array methods ends here

**150.Reduce():**

//reduce() array methods starts here

            /\*starst here\*/

            var x=[12,23,34,45,56,67,78];

                var k=x.reduce(function(a){

                    console.log(a);

                });

            /\*ends here\*/

            /\*starst here\*/

            var x=[12,23,34,45,56,67,78];

                var k=x.reduce(function(a,b){

                    console.log(a,b);

                });

            /\*ends here\*/

            /\*starst here\*/

            var x=[12,23,34,45,56,67,78];

                var k=x.reduce(function(a,b){

                    console.log(a,b);

                    return -99;

                });

            /\*ends here\*/

                /\*starst here\*/

            var x=[12,23,34,45,56,67,78];

                var k=x.reduce(function(a,b){

                    console.log(a,b);

                    return a+b;

                });

            /\*ends here\*/

                /\*starst here\*/

                var x=[12,23,34,45,56,67,78];

                var k=x.reduce(function(a,b){

                    console.log(a,b);

                    return a+b;

                });

                console.log(k);

            /\*ends here\*/

                /\*starst here\*/

                var x=[12,23,34,45,56,67,78];

                var k=x.reduce(function(a,b){

                    console.log(a,b);

                    return a+b;

                });

                console.log(k);

                var w = x.reduce((a,b)=>a+b);

                console.log(w);//output:315

            /\*ends here\*/

              /\* starts here\*/

                var x=[12,23,34,45,56,67,78]

                var k=x.every(function(){

                    console.log('hello');

                    return true;

                })

                //output: hello

                /\*ends here\*/

                var x=[12,23,34,45,56,67,78]

                var k=x.every(function(){

                    console.log('hello');

                    return true;

                })

                //output: 7hello

                /\*ends here\*/

                /\* starts here\*/

                var x=[12,23,34,45,56,67,78]

                var k=x.every(function(a,b,c){

                    console.log('hello',a,b,c);

                return true;

            })

            console.log(k);//true

            /\*ends here\*/

            //reduce() ends here

**151.Map():**

//map() array methods starts here

            /\*starts here \*/

            var x=[12,23,34,45,56,67,78];

            var k=x.map(function(a,b){

                console.log(a,b);

                return a;

            });

            console.log(k);

            /\*ends here\*/

            /\*starts here \*/

            var x=[12,23,34,45,56,67,78];

            var k=x.map(function(a,b){

                console.log(a,b);

                return -99;

            });

            console.log(k);

            /\*ends here\*/

            /\*starts here \*/

            var x=[12,23,34,45,56,67,78];

            var k=x.map(function(a,b){

                console.log(a,b);

                return a+4;

            });

            console.log(k);

            /\*ends here\*/

            //map() array methods ends here

**152.Length:**

//7 length method

             var ar7 = [10,20,30,40,50,60,70];

            console.log(ar7.length);

            //output:6

**Important Notes: length only length method not length()**

**153.toString():**

//8 toString() methods

            var ar8 = [10,20,30,40,50,60,70];

            //The toString() method returns an array as a comma separated string:

            console.log(ar8.toString());

            //output::(7) “10, 20, 30, 40, 50, 60, 70”

**154.Join():**

    //9 join() methods

            var ar9 = [10,20,30,40,50,60,70];

           //The join() method returns an array as a \* separated string:

            console.log(ar9.join("\*"));

//output:10\*20\*30\*40\*50\*60\*70

**155.concat():**

  //10 concat() starts here

            const mygirls = ["manju", "sravanthi", "anju", "abanthi"];

            const myboys = ["ashok", "mallesh", "venky", "malli"];

            const mychildren = mygirls.concat(myboys);

            console.log(mychildren);

//output:manju,sravabthi,anju,abanthi,ashok,mallesh,veky,malli

**156.Constructor:**

//12 constructor method starts here

            const fruits1 = ["Banana", "Orange", "Apple", "Mango"];

            let text=fruits1.constructor;

            console.log(text);

            //function Array() { [native code] }

**157.copyWithin():**

 //13 copyWithin() method starts here

            const ar13 = ["Banana", "Orange", "Apple", "Mango","hjhjdjh","alkak"];

            ar13.copyWithin(2,0);

            console.log(ar13);

            //output:["Banana", "Orange", "Banana", "Orange", "Apple", "Mango"]

**158.Entries():**

  //14 entires() method starts here

            const ar14 = ["Banana", "Orange", "Apple", "Mango","hjhjdjh"];

            const f= ar14.entries();

            for(let x of f){

            console.log(x);

            }

            //

            /\*

            [0, "Banana"]

            [1, "Orange"]

            [2, "Apple"]

            [3, "Mango"]

            [4, "hjhjdjh"]

            \*/

**159.Fill():**

 //fill()

        const ar19=[10,20,30];

        ar19.fill("kiwi");

        console.log(ar19);

        //output:["kiwi", "kiwi", "kiwi"]

**160.Find():**

## Definition and Usage

The find() method returns the value of the first element that passes a test.

The find() method executes a function for each array element.

The find() method returns undefined if no elements are found.

The find() method does not execute the function for empty elements.

The find() method does not change the original array.

//find() method starts here

            const ar20=[10,20,30];

            console.log(ar20.find(checkage));

            function checkage(ar20){

                return ar20>18;

            }//output:20

            //find() method ends here

**161.findIndex():**

    //findIndex() method starts here

            const ar21=[10,20,30];

            console.log(ar21.find(checkage));

            function checkage(ar20){

                return ar21>18;

            }//output:1

            //find() method ends here

**162.From():**

  //from() starts here

           let textd = "ABCDEFG"

            const myArr = Array.from(textd);

            console.log(myArr)

            //output:["A", "B", "C", "D", "E", "F", "G"]

            //from() ends here

**163.Includes():**

//includes() starts here

            const ar22=[10,20,30,40,50];

            console.log(ar22.includes(10));

            //includes() ends here

Output:true

**164.Array.isArray():**

 //Array.isArray() starts here

            const ar23=[10,20,30,40,50];

            let result=Array.isArray(ar23);

            console.log(result);

            //output:true

            //Array.isArray() ends here

**165.Keys():**

  //keys() method starts here

            const fruitskey = ["Banana", "Orange", "Apple", "Mango"];

            const keys = fruitskey.keys();

            let textkey = "";

            for (let x of keys) {

            textkey += x + "<br>";

            }

            document.getElementById("demo").innerHTML = textkey;

            //output

            /\*

            0

            1

            2

            3

            \*/

            //keys() method ends here

**166.lastIndexOf():**

 //lastIndexOf() method starts here

            const fruitsindex = ["Apple", "Orange", "Apple", "Mango"];

            let index = fruitsindex.lastIndexOf("Apple");

            console.log(index);

            //output:2

            //lastIndexOf() method ends here

**Authentication:**

checking I am a revalsys employee or not

**Authorization:**

checking I m having permisssions for all modules or not

handiling for asynchronous operations

I particular task

**promise:**

**167..Not lazy(Eager)**

const promise=new promise(()=>{

        reolve(100);

    });

**168..data emits single value**

 const promise=new promise(()=>{

        reolve(100);

    });

**169..Data is used using then method**

const promise=new promise(()=>{

        reolve(100);

    });

    promise.then((value)=>{

        console.log(value)

    });

**170.cannot cancel**

**171.unicast**

const promise=new promise(()=>{

        reolve(100);

    });

    promise.then((value)=>{

        console.log(value)

    });

    promise.then((value)=>{

        console.log(value)

    });

**Observable:**

**172.Lazy**

    obs=new observable((observer)=>{

        odserver.next(100);

    })

**If we subscribe the code then only obseravable work.**

**173..Emits multiple values over a period of time**

  obs=new observable((observer)=>{

        observer.next(100);

        observer.next(200);

        observer.next(300);

    })

**174.Data is used using subscriptions**

  obs=new observable((observer)=>{

        observer.next(100);

    })

    const sub=obs.subscribe(value)=>{

        console.log(value);

    }

**175.Can be cancelled**

**176.multicast**

const promise=new promise(()=>{

        reolve(100);

    });

    obs=new observable((observer)=>{

        odserver.next(100);

    })

    const sub1=obs.subscribe(value)=>{

        console.log(value);

    }

    const sub2=obs.subscribe(value)=>{

        console.log(value);

    }

**177.How to pass the data from onecomponent to another component how many ways?**

Ans: There are 4 ways

              1)@Input -- this decorator to share the data parent to child component

              2)@output --- this decorator to share the data child to parent component

              3)@viewchild --ViewChild is used to select an element from component's template@ view child decorator

              while ContentChild is used to select projected content

              4)@viewchilderen -- To get the refence of one component to another component.

\*\*\*\*\*

**178.What is lazy loading in angular?**

Ans: Lazy loading is a technique in Angular that allows you to load JavaScript components asynchronously when a specific route is activated.

     This can add some initial performance during the initial load, especially if you have many components with complex routing. Improve the performance of the Angular

\*\*\*\*\*

**179.Explain the importance of components & Modules?**

Ans: Components-consists of html,css,ts,services & pipes Modules-like a big container-which has one or more components

\*\*\*\*\*

**180.How to make HTTP calls using angular?**

Ans: For making the API calls, we use HTTP like get(),post(), put(),delete()

**181.Explain the need of angular pipes?**

Ans: Transform the data to expected result.Inthis we have 2 types-pure and impure.

**182.What is Routing ?**

Ans: navigating between pages-declaration in app-routing.modules.ts

**183.Patterns:**

**Maxvalue:**

var x=[10,20,30,40,50];

var max=x[0];

for(i=0;i<=5;i++){

    if(max<x[i]){

    max=x[i];

  }

}

console.log(max);

//output:50

for(let i=1;i<=5;i++){

    for(j=1;j<=5;j++){

     document.write(i);

    }

    document.write("<br>");

}

document.write("<br>");

document.write("<br>");

//output:

/\*

000000

111111

222222

333333

444444

555555

\*/

for(let i=1;i<=5;i++){

    for(j=1;j<=5;j++){

     document.write(j);

    }

    document.write("<br>");

}

document.write("<br>");

document.write("<br>");

//output:

/\*

12345

12345

12345

12345

12345

\*/

//for j=1 inside forloop exicuted 5times

for(let i=1;i<=5;i++){

    for(j=1;j<=5;j++){

        if(j<=i){

        document.write(j);

        }

    }

    document.write("<br>");

}

document.write("<br>");

document.write("<br>");

//output:

/\*

1

12

123

1234

12345

\*/

//ends here

//starts here

for(let i=1;i<=5;i++){

    for(j=1;j<=5;j++){

      if(j<=i){

        document.write(i);

        }

    }

    document.write("<br>");

}

document.write("<br>");

document.write("<br>");

//output

/\*

1

22

333

4444

55555

\*/

//ends here

**184.Constructor function:**

     function Employee(fn,ln,a,g,l,q){

            var firstname=fn;//variable

            var lastname=ln;

            this.age=a;//property

            this.location=l;//property

        }//method

        var e1 = new Employee("ashok","mancha",28,"male","guntur","masters");

        var e2 = new Employee("manjula","mancha",26,"female","nellore","masters");

        console.dir(e1);

        console.dir(e2);

//output:

{

age: 25,

qualification: "masters"

}

**186.Array constructor function and normal array:**

     var ar1 = [10,20,30,40,50]; //sort hand way of creating array //array literals

        var ar2 =  new Array(10,20,30,40,50); //creating array from array constructor function

        console.dir(ar1);

        console.dir(ar2);

        console.log(typeof(ar1));

        console.log(typeof(ar2));

console.log(typeof(document));//object

console.log(typeof(Document));//function

**185.http request:**

    //first start here

 function abc(){

        var http = new XMLHttpRequest();

        console.dir(http);

    }

    abc();

//first ends here

//second step starts here//

function abc(){

        var http = new XMLHttpRequest();

        http.open("get", "webdemo.html");

        console.dir(http);

    }

    abc();

//second step ends here//

//third step start here //

 function abc(){

        var http = new XMLHttpRequest();

        http.open("get", "webdemo.html");

        http.send();

        console.dir(http);

        http.onreadystatechange = function(){

            console.log("ready state changed", http.readyState)

        }

    }

    abc();

//third step ends here //

    //last step starts here

    function abc(){

        var http = new XMLHttpRequest();

        http.open("get", "webdemo.html");

        http.send();

        console.dir(http);

        http.onreadystatechange = function(){

            if(http.readyState==4){

                document.getElementById("edokati").innerHTML=http.respaonse;

            }

        }

    }

    abc();

    //last step ends here

         //start

         function abc(){

            var http = new XMLHttpRequest();

            console.log(http.readyState);

            console.dir(http);

            http.onreadystatechange = function(){

                console.log(http.readyState);

            }

            http.open("GET","http://isolveit.in/demos/angulardemo.html");

            http.send();

        }

        abc();

        //end

 <ul>

        <li>web server</li>

        <li>web client</li>

        <li>http</li>

        <p>Reloding of page chainging is called statelessness</p>

        <li>http request:

            <ul>

                <li>address bar</li>

                <li>anchor tag</li>

                <li>form</li>

                <li>javascripit:based on below request det 2 methods

                    <ul>

                        <li>XMLHttpRequest</li>

                        <li>fetch()</li>

                    </ul>

                </li>

            </ul>

            <li>http response</li>

        </li>

    </ul>

<p>Binding the data methods and properties together in a small unit is called encapsulation</p>

 <li><strong>Objects:</strong>object is collection of methods and proerties is called objects</li>

        <li><strong>Objects:</strong>To group the data in a set or block is called object literals</li>

**187.Arrow functions: decreasing lines of code**

**Local path:** D:\ManjulaD\DesignProjects\Notice-period\Angular-classes\Angular-23-arrow-funtions

 //normal funtion starts here

     var f1= function(x,y){

        var z;

        z=x+y;

        return z;

     }

     var res=f1(12,13);

     console.log(res);

     //output:25

      //normal functio ends here

//arrow funtions starts here

     var f1= (x,y)=>{

        var z;

        z=x+y;

        return z;

     }

     var res=f1(12,13);

     console.log(res);

     //output:25

      //arrow functions ends here

//minifined arrow funtions starts here

      var add=(x,y)=>{return x+y};

      var res2=add(10,20);

      console.log(res2);

       //output:30

      //minifined arrow functions ends here

//minifined arrow funtions starts here

      var add1=(x,y)=>

        x+y;

        console.log(add1(23,34));

        //output:57

      //minifined arrow functions ends here

    //minified arrow function starts

      var inc=(x)=>{

        return ++x;

      }

      console.log(inc(12));

      //output:13

//

      var inc=x=>

      ++x;

      console.log(inc(12));

      //output:12

    //minified arrow function ends

If suppose single parameter passing then no need to use bractes.

 //this is normal increament function starts here

   var inc2 = function(x){

        return x++;

    }

    console.log(inc2(12));

    //this is normal increament function starts here

188.Member functions:

 //member functions starts here

    var f1a=function(){

        this.x=100;

        var y=200;

        this.f2=function(){

            console.log("i am f11");

        }

        var f3=function(){

            console.log("i m f3");

        }

    }

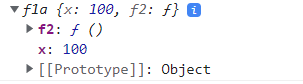
   var obj1=new f1a();

   console.log(obj1);

    //output:f1a {x: 100, f2: ƒ}

    //member functions ends here

Output:



 //member functions starts here

    var fun1=function(){

        this.x=100;//property

        var y=200;//variable

        this.f11=function(){

            console.log("i m f11");

        }//member function or method

        var f2=function(){

            console.log("i am f2");

        }

    }

    var ob1=new fun1();

    var ob2=new fun1();

    console.log(ob1.x);

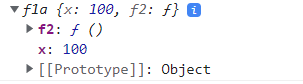
    console.log(ob2.x);

    //output:100

    //output:100

    //member functions ends here

Notes:Ob1 and ob2 same instance created like



Y is cant acceses in out side

Suppose we want acces inside only then this type is called private functions

//member funtion

    var fun1=function(){

        this.x=100;//property

        var y=200;//variable

        this.f11=function(){

            console.log("i m f11",this.x,y);

        }//member function or method

        var f2=function(){

            console.log("i am f2");

        }

    }

    var ob1=new fun1();

    ob1.f11();

    //output:i m f11 100 200

 //access in normal functions

    //note:i m accessing y but x i am not accessing y because normal functions wont access properties

    var fun1=function(){

        this.x=100;//property

        var y=200;//variable

        this.f11=function(){

            console.log("i m f11",this.x,y);

        }//member function or method

        var f2=function(){

            console.log("i am f2",this.x,y);

        }//itscreate own this

        f2();

    }

    //output:i am f2 undefined 200

//if i want to se arrow fuction i can access properies also.

    var fun1=function(){

        this.x=100;//property

        var y=200;//variable

        this.f11=function(){

            console.log("i m f11",this.x,y);

        }//member function or method

        var f2=()=>{

            console.log("i am f2",this.x,y);

        }//itscreate own this

        f2();

    }

    //output:i m f11 100 200

//if i want to se arrow fuction i can access properies also.

    var fun1=function(){

        this.x=100;//property

        var y=200;//variable

        this.f11=function(){

            console.log("i m f11",this.x,y);

        }//member function or method

        var f2=()=>{

            console.log("i am f2",this.x,y);

        }//function

        f2();

        setTimeout(function(){

            console.log("i am callback of settimeout",this.x);

        },2000);

    }

    var ob1=new fun1();

    ob1.f11();

    //output:i m f11 100 200

    //arrow-funtions.html:155 i am callback of settimeout undefined

  //if i want to se arrow fuction i can access properies also and methods.

    var fun1=function(){

        this.x=100;//property

        var y=200;//variable

        this.f11=function(){

            console.log("i m f11",this.x,y);

        }//member function or method

        var f2=()=>{

            console.log("i am f2",this.x,y);

        }//function

        f2();

        setTimeout(function(){

            console.log("i am callback of settimeout",this.x);

        },2000);

    }

    var ob1=new fun1();

    ob1.f11();

    //output:i m f11 100 200

    //arrow-funtions.html:155 i am callback of settimeout undefined

 //if i want to se arrow fuction i can access properies also.

    var fun1=function(){

        this.x=100;//property

        var y=200;//variable

        this.f11=function(){

            console.log("i m f11",this.x,y);

        }//member function or method

        var f2=()=>{

            console.log("i am f2",this.x,y);

        }//function

        f2();

        setTimeout(()=>{

            console.log("i am callback of settimeout",this.x);

        },2000);

    }

    var ob1=new fun1();

    ob1.f11();

    //output:html:175 i am callback of settimeout 100

189.Template strings:

//normal string print starts here

    var x=100;

    var y=200;

    console.log(x,"+",y,"=",x+y);//100 '+' 200 '=' 300

    //ends here

// normal string print starts here

    var x=100;

    var y=200;

    var s=x+"+"+y+"="+(x+y);

    console.log(s);

    //output:100+200=300

//ends here

 //using template strings starts here

    var x=100;

    var y=200;

    var k =`${x}+${y}=${x+y}`;

    console.log(k);

    //output:arrow-funtions.html:204 100+200=300

    //using template strings ewnds here

Important Notes: backtick charecters(` `) must and should use

//uisng normal

var x=100;

    var y=200;

    console.log(x,"+",y,"=",(x+y));

    //output:100, "+", 200, "=", 300

[

    //uisng normal

    var x=100;

    var y=201;

    console.log(x+"+"+y+"="+(x+y));

    //output:"100+201=301"

    //using template strings

    var x=100;

    var y=100;

    var z=`${x}+${y}=${x+y}`;

    console.log(z);

    //output:"100+100=200"

  //starts here

    var x=100;

    var y=100;

    console.log(`addition of ${x} and {y} is {x+y}`);

    //output:"addition of 100 and 200 is 300"

    //ends here

//starts here

    var e1='<h1>hello world</h1>';

    document.getElementById("d1").innerHTML=e1;

    //output:hello world

    //ends here

//multi lines strings starts here

    var e1="<h1>hello world</h1>

            <h2>manimumits</h2>";

    document.getElementById("d1").innerHTML=e1;

    output:error

    //multi lines strings ends her

    //multi lines strings starts here

    //note:suppose i use more lines of headings 1 line we cant use

    var e1="<h1>hello world</h1>    <h2>manimumits</h2>";

    document.getElementById("d1").innerHTML=e1;

    output:hello world

    manimumits

    //multi lines strings ends here

 //multi lines strings starts here

    //note:If i use template string backtick charecters if i use multiple lines also i will get output

    var e1=`<h1>hello world</h1>

     <h2>manimumits</h2>`;

    document.getElementById("d1").innerHTML=e1;

    //output:hello world

    //manimumits

    //multi lines strings ends here

**Classes in .ts:**

-----------------

    //class starts here

    class Employee{

        ename;

        eage;

        esal;

        egender;

        elocation

    }

    var e1= new Employee();

    console.log(e1);

    //output:arrow-funtions.html:286 Employee {ename: undefined, eage: undefined, esal: undefined, egender: undefined, elocation: undefined}

    //class ends here

 //class starts here

    class Employee1{

        ename;

        eage;

        esal;

        egender;

        elocation

    }

    var e1= new Employee1();

    e1.ename="praveen";

    e1.eage=35;

    console.log(e1);

    //output:arrow-funtions.html:301 Employee1 {ename: 'praveen', eage: 35, esal: undefined, egender: undefined, elocation: undefined}

    //class ends here

     //class starts here

     class Employee2{

        constructor(name,sal,age,gender,location){

        this.ename=name;

        this.eage=age;

        this.esal=sal;

        this.egender=gender;

        this.elocation=location

        }

    }

    var e1= new Employee2("praveen","1234","35","male","pune");

    console.log(e1);

    //output:arrow-funtions.html:317 Employee2 {ename: 'praveen', eage: '35', esal: '1234', egender: 'male', elocation: 'pune'}

    //class ends here

//class starts here

    //Notes:methods alsowe can access in class

       class Employee3{

        constructor(name,sal,age,gender,location){

        this.ename=name;

        this.eage=age;

        this.esal=sal;

        this.egender=gender;

        this.elocation=location;

        }

        ename;

        eage;

        esal;

        elocation;

        egender;

        getEmployeeGender(){

            return this.egender;

        }

    }

    var e1= new Employee3("praveen","1234","35","male","pune");

    console.log(e1.getEmployeeGender());

    //output:male

    //class ends here

**Import:**

Modules don’t create single ton but default modules create separate instance for every **import.**

**Decorator:**

**4 type of decorator:**

**1.class decorators**

**2.property decorators**

**3.Method decorators**

**4.parameter decorators**

**Function banana(target: Function): void{**

**target.prototyppe.banana = function(): void{**

**Console.log(‘We have banana!’);**

**)**

**}**

**@banana**

**Class Fruitbasket{**

**Constructor(){**

**//Implementation goes here…**

**}**

**}**

**Var basket = new Fruitbasket();**

**Basket.banana();//console will outpur ‘we have banana’**

**Important Notes:**

**1.Web application is nothing but single page application.**

**2.what class is to use tell us decorator**

**Decorator:decorator is used to tell us what type of template what class to apply it will tell**

**This is written in javascript or react.js:**

<div id="d1"></div>

           var Emloyees =  [{

        name: 'praveen',

        designation: 'angular teacher',

        descript: 'he is good teacher',

        img\_url: 'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg'

    },

    {

        name: 'raju',

        designation: 'angular ',

        descript: 'he is good teacher',

        img\_url: 'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg'

    },

    {

        name: 'ashok',

        designation: 'ammulu teacher',

        descript: 'he is good teacher',

        img\_url: 'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg'

    },

    {

        name: 'chiranjivi',

        designation: 'actor teacher',

        descript: 'he is good teacher',

        img\_url: 'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg'

    }

 ]

//  console.log(Emloyees[0].name)

for(let i=0;i<Emloyees.length;i++){

    var e11=document.createElement("h1");

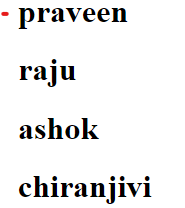
    var tn1=document.createTextNode(Emloyees[i].name);

    e11.appendChild(tn1);

    document.getElementById("d1").appendChild(e11);

}

//output:

****

           var Emloyees =  [{

        name: 'praveen',

        designation: 'angular teacher',

        descript: 'he is good teacher',

        img\_url: 'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg'

    },

    {

        name: 'raju',

        designation: 'angular ',

        descript: 'he is good teacher',

        img\_url: 'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg'

    },

    {

        name: 'ashok',

        designation: 'ammulu teacher',

        descript: 'he is good teacher',

        img\_url: 'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg'

    },

    {

        name: 'chiranjivi',

        designation: 'actor teacher',

        descript: 'he is good teacher',

        img\_url: 'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg'

    }

 ]

//  console.log(Emloyees[0].name)

for(let i=0;i<Emloyees.length;i++){

    var e11=document.createElement("h1");

    var tn1=document.createTextNode(Emloyees[i].name);

    e11.appendChild(tn1);

    var e12 = document.createElement("img");

    e12.setAttribute('src',Emloyees[i].img\_url);

    document.getElementById("d1").appendChild(e11);

    document.getElementById("d1").appendChild(e12);

}

//output:

****

**In Angular:**

**.html**

**.ts**

export class AppComponent {

  Emloyees = [

    {

      name: 'praveen',

      designation: 'angular teacher',

      descript: 'he is good teacher',

      img\_url:

        'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg',

    },

    {

      name: 'raju',

      designation: 'angular ',

      descript: 'he is good teacher',

      img\_url:

        'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg',

    },

    {

      name: 'ashok',

      designation: 'ammulu teacher',

      descript: 'he is good teacher',

      img\_url:

        'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg',

    },

    {

      name: 'chiranjivi',

      designation: 'actor teacher',

      descript: 'he is good teacher',

      img\_url:

        'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg',

    },

  ];

}

**Output:**

****

**If I want multiple employee data I want**

<div \*ngFor="let i of Emloyees">

  <h1>{{ Emloyees[0].name }}</h1>

  <h3>{{ Emloyees[0].designation }}</h3>

  <p>{{ Emloyees[0].descript }}</p>

  <img [src]="Emloyees[0].img\_url" />

</div>

**.ts**

export class AppComponent {

  Emloyees = [

    {

      name: 'praveen',

      designation: 'angular teacher',

      descript: 'he is good teacher',

      img\_url:

        'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg',

    },

    {

      name: 'raju',

      designation: 'angular ',

      descript: 'he is good teacher',

      img\_url:

        'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg',

    },

    {

      name: 'ashok',

      designation: 'ammulu teacher',

      descript: 'he is good teacher',

      img\_url:

        'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg',

    },

    {

      name: 'chiranjivi',

      designation: 'actor teacher',

      descript: 'he is good teacher',

      img\_url:

        'https://s3-ap-southeast-1.amazonaws.com/tv-prod/member/photo/1787281-medium190.jpg',

    },

  ];

}

**Output:**



**Template variables:**

<div id="m1">

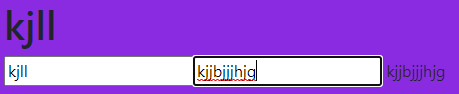
  <h1>{{x}}</h1>

  <input type="text" [(ngModel)]="x">

  <input type="text" [(ngModel)]="y">

  {{y}}

**Output:**

****

<span \*ngIf="10>3 then t1;else t2"></span>

  <ng-template #t1><div>praveen</div></ng-template>

  <ng-template #t2><div>praveen</div></ng-template>

</div>

<p>Template variables are binded to particular block</p>

**//output:** Praveen

**SASS:**

What is Sass? Sass stands for **Syntactically Awesome Stylesheet**. Sass is an extension to CSS. Sass is a CSS pre-processor. Sass is completely compatible with all versions of CSS.

Scss:

The term SCSS is an acronym for **Sassy Cascading Style Sheets**. It is basically a more advanced and evolved variant of the CSS language. Natalie Weizenbaum and Chris Eppstein created it, and Hampton Catlin designed it. It comes with more advanced features- thus often called Sassy CSS.

**The <strong> element is for content that is of greater importance, while the <b> element is used to draw attention to text without indicating that it's more important**.

**Properties:**

<img [src]="img\_url">

**.ts**

  img\_url="https://www.filmibeat.com/ph-big/2019/05/jr-ntr\_1558329150190.jpg";

**Output:**

**Bigger image**

**Staep2:**

<img [src]="img\_url" (click)="xyz()">

**.ts**

 xyz(e:any){

console.log("ABC called",e);

  }

**Output:**

**if click image then**

**I will get in console** ABC called

**Step3:**

<img [src]="img\_url" (click)="xyz(this)">

**.ts**

xyz(e:any){

console.log("ABC called",e);

  }

**Output:**

****

<img [src]="img\_url" (click)="xyz($event)">

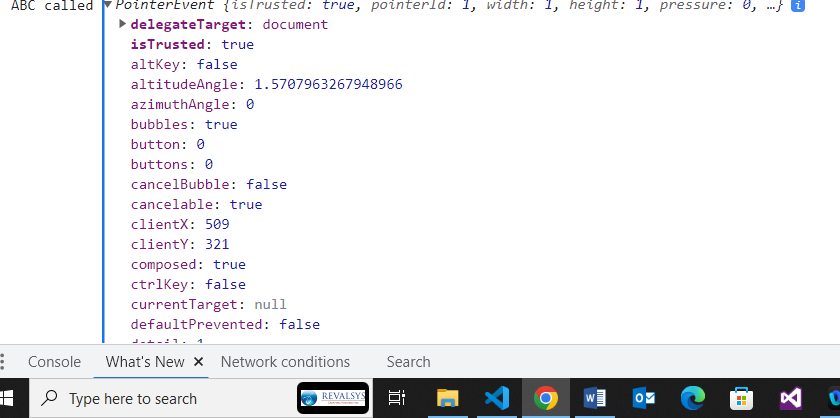
**.ts**

xyz(e:any){

console.log("ABC called",e);

  }

**Output**

****

Here I can get every thing from $event

Actually **this** and **event**s are global

Step3:

<img [src]="img\_url" (click)="xyz($event)">

 xyz(e:Event){

    e.target['style'].width=100px;

  }

Step4:

<div \*ngFor="let students of students" >

  <h1  (click)="exp($event)">{{ students.firstname }}</h1>

  <h2 style="display:none;">{{students.lastname}}</h2>

</div>

exp(a: any) {

    console.log(a);

    a.target.nextSibling.style.display=’block’;

  }

**For accordion**

**189.Finnaly accordion**

<!---->

<div \*ngFor="let students of students" >

  <h1  (click)="exp($event)">{{ students.firstname }}</h1>

  <h2 style="display:none;">{{students.lastname}}</h2>

</div>

<!---->

**.ts**

 students = [

    {

      firstname: 'ashok',

      lastname: 'mancha',

    },

    {

      firstname: 'manju',

      lastname: 'devarala',

    },

  ];

  exp(a: any) {

    console.log(a);

    var x=a.target.nextSibling.style.display='block';

    if(x=='none'){

      a.target.nextSibling.style.display='block';

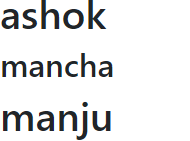
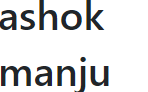
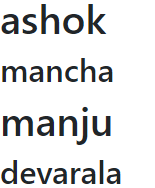
    }

    else{

      a.target.nextSibling.style.display='block'

    }

  }

**** ****

**190.This is one way data binding:**

<img src="{{img\_url}}"><!--interpolation to bind img-->

<img [src]="img\_url" width="100px" alt=""><!--property binding-->

<img [src]="img\_url" width="100px" alt="" (click)="pqr()"><!--event binding-->

<input type="text" [value]="name">

{{name}}

pqr(){

    console.log("i m pqr")

  }

name="angular";

img\_url="https://www.filmibeat.com/ph-big/2019/05/jr-ntr\_1558329150190.jpg";

**output:**

****

**Two way data binding:**

<!--this is 1 way data binidng strts here-->

<input type="text" [(ngModel)]="name">

{{name}}

<!--this is 1 way data binidng ends here-->

****

{{cost}}

<input type="text" [value]="cost" #c><!--#c is type of variables is called template variables-->

<button (click)="update(c.value)">click</button>

**.ts**

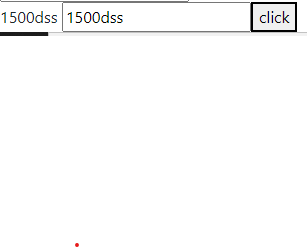
  cost=1500;

  update(t:any){

    this.cost=t;

  }

**When I enter input filed after clicking click button outside value changed**

****

**Keyup**

{{cost}}

<input type="text" [value]="cost" #c (keyup.enter)="update(c.value)"><!--#c is type of variables is called template variables-->

**Keyup:**

<input type="text" #d (keyup)="0">{{d.value}}

****

<input type="checkbox" #q (keyup)="0">{{q.value}}

<h1 #t1>manju</h1>

<h2 #t2>nfkd</h2>

<button (click)="edf(t1,t2)">lets click</button>

 edf(tv1: any,tv2: any){

    console.log(tv1,tv2)

  }

**Example1:**

<input type="radio" name="gender" value="male" [(ngModel)]="g">male<br>

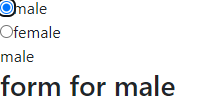
<input type="radio" name="gender" value="female" [(ngModel)]="g">female<br>

{{g}}

<h3 \*ngIf="g=='male'">form for male</h3>

<h3 \*ngIf="g=='female'" >form for female</h3>

 g:any;

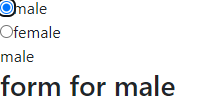
****

**Example2:**

<span \*ngIf="g=='m' then t1;else t2"></span>

<ng-template #t1>Male for form</ng-template>

<ng-template> #t2feMale for form</ng-template>

****

**How to Avoid Cross Browser Compatibility Issues?**

1. Validate HTML and CSS. ...
2. Maintain Layout Compatibility. ...
3. Use CSS Resets. ...
4. Provide Support for Basic Features of the Application. ...
5. Check JavaScript Issues to avoid the Cross-Browser compatibility issues. ...
6. Check DOCTYPE tag.

**200.**

**function outerFunc(outerParam) {**

**function innerFunc(innerParam) {**

**outerParam["b"] = innerParam;**

**}**

**return innerFunc;**

**} const obj = {a:1}**

**const example = outerFunc(obj);**

**const answer = example(2)**

**console.log(obj);**

**Ans)**

**{**

**a:1**

**b:2**

**}**

**201.**

**var y = 1;**

**if (function f() {}) {**

**y += typeof f;**

**}**

**console.log(y);**

**Ans)undefined**

**202.** **for (var i = 0; i < 4; i++) {**

**setTimeout(() => console.log(i), 0)**

**}**

**Ans)4**

**4**

**4**

**4**

**203. for(let i=0; i<10; i++){**

**setTimeout(function(){**

**console.log(i)**

**}, 1000)**

**}**

**Output:0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

## [The prototype chain](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes" \l "the_prototype_chain)

In the browser's console, try creating an object literal:

const myObject = {

city: "Madrid",

greet() {

console.log(`Greetings from ${this.city}`);

},

};

myObject.greet(); // Greetings from Madrid

**204.web application?**

Ans)response to same application and same pages

**205.web service?**

Ans)Response to the other appications also

**206.API**

Ans) api is a combinations which programs or package of programs which provides us some functionalityinteract with some other external resources.

Grid and flexbox. The basic difference between CSS Grid Layout and CSS Flexbox Layout is that **flexbox was designed for layout in one dimension - either a row or a column.** Grid was designed for two-dimensional layout - rows, and columns at the same time.

### **207. What are the different data types present in javascript?**

To know the type of a JavaScript variable, we can use the ****typeof****operator.

****1. Primitive types****

****String****- It represents a series of characters and is written with quotes. A string can be represented using a single or a double quote.

Example :

**var** str = "Vivek Singh Bisht"; //using double quotes**var** str2 = 'John Doe'; //using single quotes

* ****Number****- It represents a number and can be written with or without decimals.

Example :

**var** x = 3; //without decimal**var** y = 3.6; //with decimal

* ****BigInt****- This data type is used to store numbers which are above the limitation of the Number data type. It can store large integers and is represented by adding “n” to an integer literal.

Example :

**var** bigInteger = 234567890123456789012345678901234567890;

* ****Boolean****- It represents a logical entity and can have only two values : true or false. Booleans are generally used for conditional testing.

Example :

**var** a = 2;**var** b = 3;**var** c = 2;

(a == b) // returns false

(a == c) //returns true

* ****Undefined****- When a variable is declared but not assigned, it has the value of undefined and it’s type is also undefined.

Example :

**var** x; // value of x is undefined**var** y = undefined; // we can also set the value of a variable as undefined

* ****Null****- It represents a non-existent or a invalid value.

Example :

**var** z = null;

* ****Symbol****- It is a new data type introduced in the ES6 version of javascript. It is used to store an anonymous and unique value.

Example :

**var** symbol1 = Symbol('symbol');

* typeof ****of primitive types****:

**typeof** "John Doe" // Returns "string"**typeof** 3.14 // Returns "number"**typeof** true // Returns "boolean"**typeof** 234567890123456789012345678901234567890n // Returns bigint**typeof** undefined // Returns "undefined"**typeof** null // Returns "object" (kind of a bug in JavaScript)**typeof** Symbol('symbol') // Returns Symbol

****2. Non-primitive types****

* Primitive data types can store only a single value. To store multiple and complex values, non-primitive data types are used.
* Object - Used to store collection of data.
* Example:

// Collection of data in key-value pairs

**var** obj1 = {

x: 43,

y: "Hello world!",

z: **function**(){

**return** this.x;

}

}

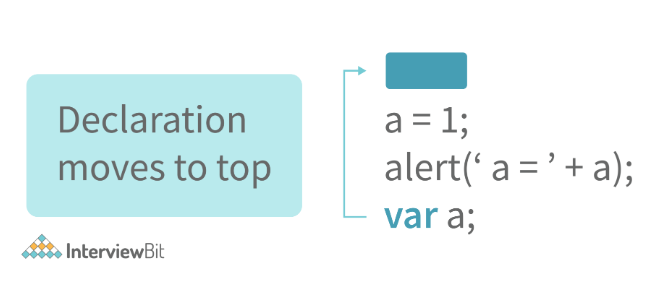
// Collection of data as an ordered list

**var** array1 = [5, "Hello", true, 4.1];

#### **Note- It is important to remember that any data type that is not a primitive data type, is of Object type in javascript.**

### **208.. Explain Hoisting in javascript.**

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



This means that irrespective of where the variables and functions are declared, they are moved on top of the scope. The scope can be both local and global.

****Example 1:****

hoistedVariable = 3;console.log(hoistedVariable); // outputs 3 even when the variable is declared after it is initialized **var** hoistedVariable;

****Example 2:****

hoistedFunction(); // Outputs " Hello world! " even when the function is declared after calling

**function** **hoistedFunction**(){

console.log(" Hello world! ");

}

****Example 3:****

// Hoisting takes place in the local scope as well**function** **doSomething**(){

x = 33;

console.log(x);

**var** x;

}

doSomething(); // Outputs 33 since the local variable “x” is hoisted inside the local scope

#### ****Note - Variable initializations are not hoisted, only variable declarations are hoisted:****

**var** x;console.log(x); // Outputs "undefined" since the initialization of "x" is not hoisted

x = 23;

#### ****Note - To avoid hoisting, you can run javascript in strict mode by using “use strict” on top of the code:****

"use strict";

x = 23; // Gives an error since 'x' is not declared**var** x;

### **209. Why do we use the word “debugger” in javascript?**

The debugger for the browser must be activated in order to debug the code. Built-in debuggers may be switched on and off, requiring the user to report faults. The remaining section of the code should stop execution before moving on to the next line while debugging.

### **210..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"

### **211.Difference between var and let keyword in javascript.**

Some differences are

1. From the very beginning, the 'var' keyword was used in JavaScript programming ****whereas the keyword****'let' was just added in 2015.
2. The keyword 'Var' has a function scope. Anywhere in the function, the variable specified using var is accessible but in ‘let’ the scope of a variable declared with the 'let' keyword is limited to the block in which it is declared. Let's start with a Block Scope.
3. In ECMAScript 2015, let and const are hoisted but not initialized. Referencing the variable in the block before the variable declaration results in a ReferenceError because the variable is in a "temporal dead zone" from the start of the block until the declaration is processed.

### **212. Explain Implicit Type Coercion in javascript.**

Implicit type coercion in javascript is the automatic conversion of value from one data type to another. It takes place when the operands of an expression are of different data types.

* ****String coercion****

String coercion takes place while using the ‘ + ‘ operator. When a number is added to a string, the number type is always converted to the string type.

Example 1:

**var** x = 3;**var** y = "3";

x + y // Returns "33"

Example 2:

**var** x = 24;**var** y = "Hello";

x + y // Returns "24Hello";

#### Note - ‘ + ‘ operator when used to add two numbers, outputs a number. The same ‘ + ‘ operator when used to add two strings, outputs the concatenated string:

**var** name = "Vivek";**var** surname = " Bisht";

name + surname // Returns "Vivek Bisht"

Let’s understand both the examples where we have added a number to a string,

When JavaScript sees that the operands of the expression x + y are of different types ( one being a number type and the other being a string type ), it converts the number type to the string type and then performs the operation. Since after conversion, both the variables are of string type, the ‘ + ‘ operator outputs the concatenated string “33” in the first example and “24Hello” in the second example.

#### Note - Type coercion also takes place when using the ‘ - ‘ operator, but the difference while using ‘ - ‘ operator is that, a string is converted to a number and then subtraction takes place.

**var** x = 3;

Var y = "3";

x - y //Returns 0 since the variable y (string type) is converted to a number type

* ****Boolean Coercion****

Boolean coercion takes place when using logical operators, ternary operators, if statements, and loop checks. To understand boolean coercion in if statements and operators, we need to understand truthy and falsy values.  
  
Truthy values are those which will be converted (coerced) to ****true****. Falsy values are those which will be converted to ****false****.  
  
All values except ****false, 0, 0n, -0, “”, null, undefined, and NaN****are truthy values.

****If statements:****

Example:

**var** x = 0;**var** y = 23;

**if**(x) { console.log(x) } // The code inside this block will not run since the value of x is 0(Falsy)

**if**(y) { console.log(y) } // The code inside this block will run since the value of y is 23 (Truthy)

* ****Logical operators:****

Logical operators in javascript, unlike operators in other programming languages, ****do not return true or false. They always return one of the operands.****  
  
****OR ( | | ) operator****- If the first value is truthy, then the first value is returned. Otherwise, always the second value gets returned.  
  
****AND ( && ) operator****- If both the values are truthy, always the second value is returned. If the first value is falsy then the first value is returned or if the second value is falsy then the second value is returned.  
  
Example:

**var** x = 220;**var** y = "Hello";**var** z = undefined;

x | | y // Returns 220 since the first value is truthy

x | | z // Returns 220 since the first value is truthy

x && y // Returns "Hello" since both the values are truthy

y && z // Returns undefined since the second value is falsy

**if**( x && y ){

console.log("Code runs" ); // This block runs because x && y returns "Hello" (Truthy)

}

**if**( x || z ){

console.log("Code runs"); // This block runs because x || y returns 220(Truthy)

}

* ****Equality Coercion****

Equality coercion takes place when using ‘ == ‘ operator. As we have stated before  
  
****The ‘ == ‘ operator compares values and not types.****  
  
While the above statement is a simple way to explain == operator, it’s not completely true  
  
The reality is that while using the ‘==’ operator, coercion takes place.  
  
The ‘==’ operator, converts both the operands to the same type and then compares them.  
  
Example:

**var** a = 12;**var** b = "12";

a == b // Returns true because both 'a' and 'b' are converted to the same type and then compared. Hence the operands are equal.

Coercion does not take place when using the ‘===’ operator. Both operands are not converted to the same type in the case of ‘===’ operator.

Example:

**var** a = 226;**var** b = "226";

a === b // Returns false because coercion does not take place and the operands are of different types. Hence they are not equal.

### **213. 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.****



Since javascript is a loosely(dynamically) typed language, variables in JS are not associated with any type. A variable can hold the value of any data type.

For example, a variable that is assigned a number type can be converted to a string type:

**var** a = 23;**var** a = "Hello World!";

### **214. 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.

isNaN("Hello") // Returns trueisNaN(345) // Returns falseisNaN('1') // Returns false, since '1' is converted to Number type which results in 0 ( a number) isNaN(true) // Returns false, since true converted to Number type results in 1 ( a number)isNaN(false) // Returns falseisNaN(undefined) // Returns true

### **215. Explain passed by value and passed by reference.**

****In JavaScript, primitive data types are passed by value and non-primitive data types are passed by reference.****  
  
For understanding passed by value and passed by reference, we need to understand what happens when we create a variable and assign a value to it,

**var** x = 2;

In the above example, we created a variable x and assigned it a value of “2”. In the background, the “=” (assign operator) allocates some space in the memory, stores the value “2” and returns the location of the allocated memory space. Therefore, the variable x in the above code points to the location of the memory space instead of pointing to the value 2 directly.

Assign operator behaves differently when dealing with primitive and non-primitive data types,  
  
****Assign operator dealing with primitive types:****



**var** y = 234;**var** z = y;

In the above example, the assign operator knows that the value assigned to y is a primitive type (number type in this case), so when the second line code executes, where the value of y is assigned to z, the assign operator takes the value of y (234) and allocates a new space in the memory and returns the address. Therefore, variable z is not pointing to the location of variable y, instead, it is pointing to a new location in the memory.

**var** y = #8454; // y pointing to address of the value 234

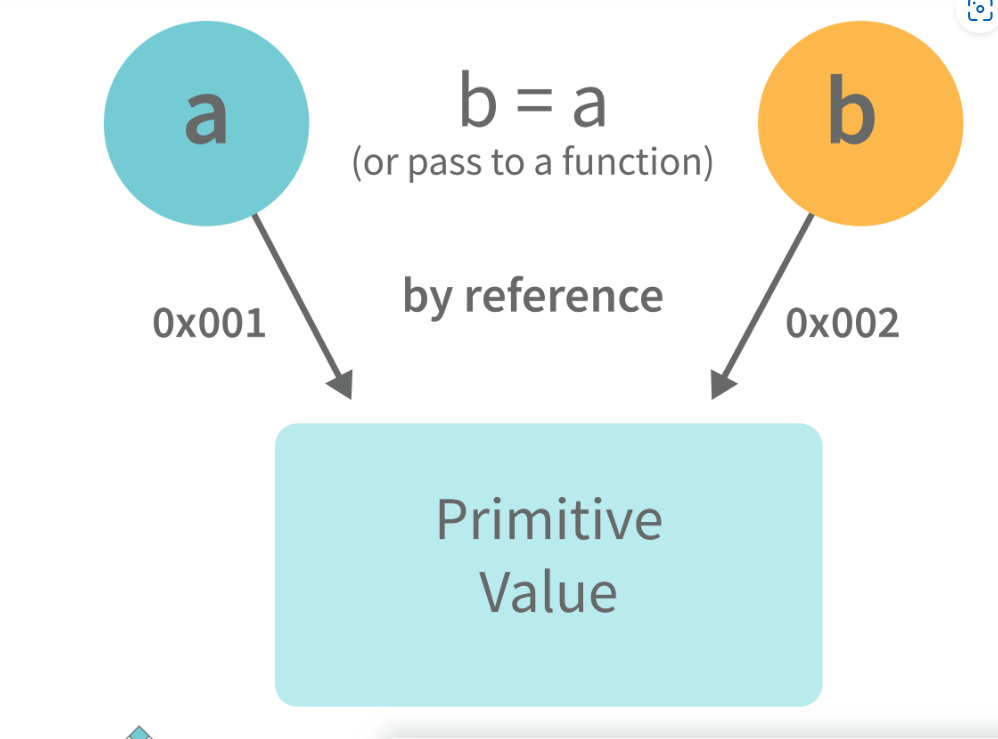
**var** z = y;

**var** z = #5411; // z pointing to a completely new address of the value 234

// Changing the value of y

y = 23;console.log(z); // Returns 234, since z points to a new address in the memory so changes in y will not effect z

From the above example, we can see that primitive data types when passed to another variable, are passed by value. Instead of just assigning the same address to another variable, the value is passed and new space of memory is created.  
  
****Assign operator dealing with non-primitive types:****



**var** obj = { name: "Vivek", surname: "Bisht" };**var** obj2 = obj;

In the above example, the assign operator directly passes the location of the variable obj to the variable obj2. In other words, the reference of the variable obj is passed to the variable obj2.

**var** obj = #8711; // obj pointing to address of { name: "Vivek", surname: "Bisht" }**var** obj2 = obj;

**var** obj2 = #8711; // obj2 pointing to the same address

// changing the value of obj1

obj.name = "Akki";console.log(obj2);

// Returns {name:"Akki", surname:"Bisht"} since both the variables are pointing to the same address.

From the above example, we can see that while passing non-primitive data types, the assigned operator directly passes the address (reference).  
  
Therefore, non-primitive data types are always ****passed by reference.****

### **216. What is an Immediately Invoked Function in JavaScript?**

****An Immediately Invoked Function ( known as IIFE and pronounced as IIFY) is a function that runs as soon as it is defined.****

Syntax of IIFE :

(**function**(){

// Do something;

})();

To understand IIFE, we need to understand the two sets of parentheses that are added while creating an IIFE :  
  
The first set of parenthesis:

(**function** (){

//Do something;

})

While executing javascript code, whenever the compiler sees the word “function”, it assumes that we are declaring a function in the code. Therefore, if we do not use the first set of parentheses, the compiler throws an error because it thinks we are declaring a function, and by the syntax of declaring a function, a function should always have a name.

**function**() {

//Do something;

}// Compiler gives an error since the syntax of declaring a function is wrong in the code above.

To remove this error, we add the first set of parenthesis that tells the compiler that the function is not a function declaration, instead, it’s a function expression.  
  
The second set of parenthesis:

(**function** (){

//Do something;

})();

From the definition of an IIFE, we know that our code should run as soon as it is defined. A function runs only when it is invoked. If we do not invoke the function, the function declaration is returned:

(**function** (){

// Do something;

})

// Returns the function declaration

****Therefore to invoke the function, we use the second set of parenthesis.****

### **217. What do you mean by strict mode in javascript and characteristics of javascript strict-mode?**

In ECMAScript 5, a new feature called JavaScript Strict Mode allows you to write a code or a function in a "strict" operational environment. In most cases, this language is 'not particularly severe' when it comes to throwing errors. In 'Strict mode,' however, all forms of errors, including silent errors, will be thrown. As a result, debugging becomes a lot simpler.  Thus programmer's chances of making an error are lowered.

Characteristics of strict mode in javascript

1. Duplicate arguments are not allowed by developers.
2. In strict mode, you won't be able to use the JavaScript keyword as a parameter or function name.
3. The 'use strict' keyword is used to define strict mode at the start of the script. Strict mode is supported by all browsers.

### 218**. Explain Higher Order Functions in javascript.**

****Functions that operate on other functions, either by taking them as arguments or by returning them, are called higher-order functions.****  
  
Higher-order functions are a result of functions being ****first-class citizens****in javascript.

Examples of higher-order functions:

**function** **higherOrder**(fn) {

fn();

}

higherOrder(**function**() { console.log("Hello world") });

**function** **higherOrder2**() {

**return** **function**() {

**return** "Do something";

}

} **var** x = higherOrder2();

x() // Returns "Do something"

### **219. Explain “this” keyword.**

****The “this” keyword refers to the object that the function is a property of.****  
  
****The value of the “this” keyword will always depend on the object that is invoking the function.\****

Confused? Let’s understand the above statements by examples:

**function** **doSomething**() {

console.log(this);

}

doSomething();

What do you think the output of the above code will be?

Note - Observe the line where we are invoking the function.

Check the definition again:

#### ****The “this” keyword refers to the object that the function is a property of.****

In the above code, the function is a property of which object?

Since the function is invoked in the global context, ****the function is a property of the global object.****

Therefore, the output of the above code will be ****the global object.****Since we ran the above code inside the browser, the global object is ****the window object.****

Example 2:

**var** obj = {

name: "vivek",

getName: **function**(){

console.log(this.name);

}

}

obj.getName();

In the above code, at the time of invocation, the getName function is a property of the object ****obj****, therefore, ****this****keyword will refer to the object ****obj****, and hence the output will be “vivek”.

Example 3:

**var** obj = {

name: "vivek",

getName: **function**(){

console.log(this.name);

}

}

**var** getName = obj.getName;

**var** obj2 = {name:"akshay", getName };

obj2.getName();

Can you guess the output here?

The output will be “akshay”.

Although the getName function is declared inside the object ****obj****, at the time of invocation, getName() is a property of ****obj2****, therefore the “this” keyword will refer to ****obj2****.

The silly way to understand the “****this”****keyword is, whenever the function is invoked, check the object before the ****dot****. The value of ****this****. keyword will always be the object before the ****dot****.

If there is no object before the dot-like in example1, the value of this keyword will be the global object.

Example 4:

**var** obj1 = {

address : "Mumbai,India",

getAddress: **function**(){

console.log(this.address);

}

}

**var** getAddress = obj1.getAddress;**var** obj2 = {name:"akshay"};

obj2.getAddress();

Can you guess the output?

****The output will be an error.****

Although in the code above, this keyword refers to the object ****obj2****, obj2 does not have the property “address”‘, hence the getAddress function throws an error.

### **220. What do you mean by Self Invoking Functions?**

Without being requested, a self-invoking expression is automatically invoked (initiated). If a function expression is followed by (), it will execute automatically. A function declaration cannot be invoked by itself.

Normally, we declare a function and call it, however, anonymous functions may be used to run a function automatically when it is described and will not be called again. And there is no name for these kinds of functions.

### **221. Explain call(), apply() and, bind() methods.**

****1. 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"]);

****2. 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: TS0122, Bullet

### **222. What is the difference between exec () and test () methods in javascript?**

* ****test ()**** and ****exec ()**** are RegExp expression methods used in javascript.
* We'll use ****exec ()**** to search a string for a specific pattern, and if it finds it, it'll return the pattern directly; else, it'll return an 'empty' result.
* We will use a****test ()**** to find a string for a specific pattern. It will return the Boolean value 'true' on finding the given text otherwise, it will return 'false'.

### **223. What is currying in JavaScript?**

****Currying is an advanced technique to transform a function of arguments n, to n functions of one or fewer arguments.****

Example of a curried function:

**function** **add** (a) {

**return** **function**(b){

**return** a + b;

}

}

add(3)(4)

For Example, if we have a function ****f(a,b)****, then the function after currying, will be transformed to ****f(a)(b).****  
  
By using the currying technique, we do not change the functionality of a function, we just change the way it is invoked.  
  
Let’s see currying in action:

**function** **multiply**(a,b){

**return** a\*b;

}

**function** **currying**(fn){

**return** **function**(a){

**return** **function**(b){

**return** fn(a,b);

}

}

}

**var** curriedMultiply = currying(multiply);

multiply(4, 3); // Returns 12

curriedMultiply(4)(3); // Also returns 12

As one can see in the code above, we have transformed the function ****multiply(a,b)****to a function ****curriedMultiply****, which takes in one parameter at a time.

### **224. 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

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

### **225. 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
* Local or Function Scope
* Block Scope

****Global Scope:****Variables or functions declared in the global namespace have global scope, which means all the variables and functions having global scope can be accessed from anywhere inside the code.

**var** globalVariable = "Hello world";

**function** **sendMessage**(){

**return** globalVariable; // can access globalVariable since it's written in global space

}**function** **sendMessage2**(){

**return** sendMessage(); // Can access sendMessage function since it's written in global space

}

sendMessage2(); // Returns “Hello world”

****Function Scope:****Any variables or functions declared inside a function have local/function scope, which means that all the variables and functions declared inside a function, can be accessed from within the function and not outside of it.

**function** **awesomeFunction**(){

**var** a = 2;

**var** multiplyBy2 = **function**(){

console.log(a\*2); // Can access variable "a" since a and multiplyBy2 both are written inside the same function

}

}console.log(a); // Throws reference error since a is written in local scope and cannot be accessed outside

multiplyBy2(); // Throws reference error since multiplyBy2 is written in local scope

****Block Scope:****Block scope is related to the variables declared using let and const. Variables declared with var do not have block scope. Block scope tells us that any variable declared inside a block { }, can be accessed only inside that block and cannot be accessed outside of it.

{

**let** x = 45;

}

console.log(x); // Gives reference error since x cannot be accessed outside of the block

**for**(**let** i=0; i<2; i++){

// do something

}

console.log(i); // Gives reference error since i cannot be accessed outside of the for loop block

****Scope Chain:****JavaScript engine also uses Scope to find variables. Let’s understand that using an example:

**var** y = 24;

**function** **favFunction**(){

**var** x = 667;

**var** anotherFavFunction = **function**(){

console.log(x); // Does not find x inside anotherFavFunction, so looks for variable inside favFunction, outputs 667

}

**var** yetAnotherFavFunction = **function**(){

console.log(y); // Does not find y inside yetAnotherFavFunction, so looks for variable inside favFunction and does not find it, so looks for variable in global scope, finds it and outputs 24

}

anotherFavFunction();

yetAnotherFavFunction();

}

favFunction();

****As you can see in the code above, if the javascript engine does not find the variable in local scope, it tries to check for the variable in the outer scope. If the variable does not exist in the outer scope, it tries to find the variable in the global scope.****

If the variable is not found in the global space as well, a reference error is thrown.

### **226. Explain Closures in JavaScript.**

Closures are an ability of a function to remember the variables and functions that are declared in its outer scope.

**var** Person = **function**(pName){

**var** name = pName;

this.getName = **function**(){

**return** name;

}

}

**var** person = **new** Person("Neelesh");console.log(person.getName());

Let’s understand closures by example:

**function** **randomFunc**(){

**var** obj1 = {name:"Vivian", age:45};

**return** **function**(){

console.log(obj1.name + " is "+ "awesome"); // Has access to obj1 even when the randomFunc function is executed

}

}

**var** initialiseClosure = randomFunc(); // Returns a function

initialiseClosure();

Let’s understand the code above,  
  
The function randomFunc() gets executed and returns a function when we assign it to a variable:

**var** initialiseClosure = randomFunc();

The returned function is then executed when we invoke initialiseClosure:

initialiseClosure();

The line of code above outputs “Vivian is awesome” and this is possible because of closure.

console.log(obj1.name + " is "+ "awesome");

When the function randomFunc() runs, it seems that the returning function is using the variable obj1 inside it:

Therefore randomFunc(), instead of destroying the value of obj1 after execution, ****saves the value in the memory for further reference.**** This is the reason why the returning function is able to use the variable declared in the outer scope even after the function is already executed.  
  
****This ability of a function to store a variable for further reference even after it is executed is called Closure.****

### **227. Mention some advantages of javascript.**

There are many advantages of javascript. Some of them are

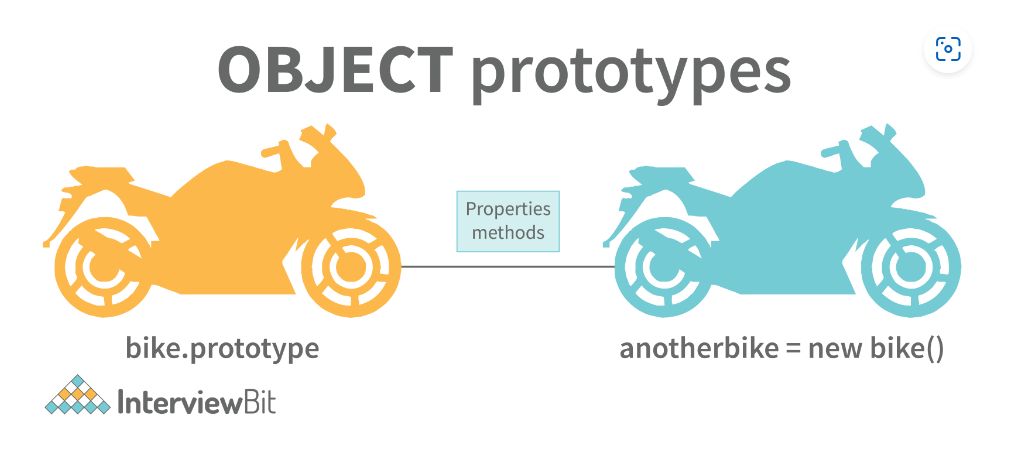
1. Javascript is executed on the client-side as well as server-side also. There are a variety of Frontend Frameworks that you may study and utilize. However, if you want to use JavaScript on the backend, you'll need to learn NodeJS. It is currently the only JavaScript framework that may be used on the backend.
2. Javascript is a simple language to learn.
3. Web pages now have more functionality because of Javascript.
4. To the end-user, Javascript is quite quick.

### **228.. What are object prototypes?**

All javascript objects inherit properties from a prototype. For example,

* Date objects inherit properties from the Date prototype
* Math objects inherit properties from the Math prototype
* Array objects inherit properties from the Array prototype.
* On top of the chain is ****Object.prototype.****Every prototype inherits properties and methods from the Object.prototype.
* ****A prototype is a blueprint of an object. The prototype**** allows us to use properties and methods on an object even if the properties and methods do not exist on the current object.

Let’s see prototypes help us use methods and properties:



**var** arr = [];

arr.push(2);

console.log(arr); // Outputs [2]

In the code above, as one can see, we have not defined any property or method called push on the array “arr” but the javascript engine does not throw an error.

The reason is the use of prototypes. As we discussed before, Array objects inherit properties from the Array prototype.

The javascript engine sees that the method push does not exist on the current array object and therefore, looks for the method push inside the Array prototype and it finds the method.

Whenever the property or method is not found on the current object, the javascript engine will always try to look in its prototype and if it still does not exist, it looks inside the prototype's prototype and so on.

### **229. 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

* In the code above, we are performing mathematical operations on the sum of two numbers. The operationOnSum function takes 3 arguments, the first number, the second number, and the operation that is to be performed on their sum (callback).
* Both divideByHalf and multiplyBy2 functions are used as callback functions in the code above.
* These callback functions will be executed only after the function operationOnSum is executed.
* Therefore, a callback is a function that will be executed after another function gets executed.

### **24. What are the types of errors in javascript?**

There are two types of errors in javascript.

1. ****Syntax error****: Syntax errors are mistakes or spelling problems in the code that cause the program to not execute at all or to stop running halfway through. Error messages are usually supplied as well.
2. ****Logical error****: Reasoning mistakes occur when the syntax is proper but the logic or program is incorrect. The application executes without problems in this case. However, the output findings are inaccurate. These are sometimes more difficult to correct than syntax issues since these applications do not display error signals for logic faults.

### **230. 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.  
Let’s understand memoization, by converting a simple function to a memoized function:

Note- Memoization is used for expensive function calls but in the following example, we are considering a simple function for understanding the concept of memoization better.

Consider the following function:

**function** **addTo256**(num){

**return** num + 256;

}

addTo256(20); // Returns 276

addTo256(40); // Returns 296

addTo256(20); // Returns 276

In the code above, we have written a function that adds the parameter to 256 and returns it.  
  
When we are calling the function addTo256 again with the same parameter (“20” in the case above), we are computing the result again for the same parameter.  
  
Computing the result with the same parameter, again and again, is not a big deal in the above case, but imagine if the function does some heavy-duty work, then, computing the result again and again with the same parameter will lead to wastage of time.

This is where memoization comes in, by using memoization we can store(cache) the computed results based on the parameters. If the same parameter is used again while invoking the function, instead of computing the result, we directly return the stored (cached) value.

Let’s convert the above function addTo256, to a memoized function:

**function** **memoizedAddTo256**(){

**var** cache = {};

**return** **function**(num){

**if**(num **in** cache){

console.log("cached value");

**return** cache[num]

}

**else**{

cache[num] = num + 256;

**return** cache[num];

}

}

}**var** memoizedFunc = memoizedAddTo256();

memoizedFunc(20); // Normal return

memoizedFunc(20); // Cached return

In the code above, if we run the memoizedFunc function with the same parameter, instead of computing the result again, it returns the cached result.

Note- Although using memoization saves time, it results in larger consumption of memory since we are storing all the computed results.

### **231. What is recursion in a programming language?**

Recursion is a technique to iterate over an operation by having a function call itself repeatedly until it arrives at a result.

function add(number) {

if (number <= 0) {

return 0;

} else {

return number + add(number - 1);

}

}

add(3) => 3 + add(2)

3 + 2 + add(1)

3 + 2 + 1 + add(0)

3 + 2 + 1 + 0 = 6

Example of a recursive function:  
  
The following function calculates the sum of all the elements in an array by using recursion:

function computeSum(arr){

if(arr.length === 1){

return arr[0];

}

else{

return arr.pop() + computeSum(arr);

}

}

computeSum([7, 8, 9, 99]); // Returns 123

### **27. What is the use of a constructor function in javascript?**

Constructor functions are used to create objects in javascript.

When do we use constructor functions?

If we want to create multiple objects having similar properties and methods, constructor functions are used.

#### ****Note- The name of a constructor function should always be written in Pascal Notation: every word should start with a capital letter.****

Example:

**function** **Person**(name,age,gender){

this.name = name;

this.age = age;

this.gender = gender;

}

**var** person1 = **new** Person("Vivek", 76, "male");console.log(person1);

**var** person2 = **new** Person("Courtney", 34, "female");console.log(person2);

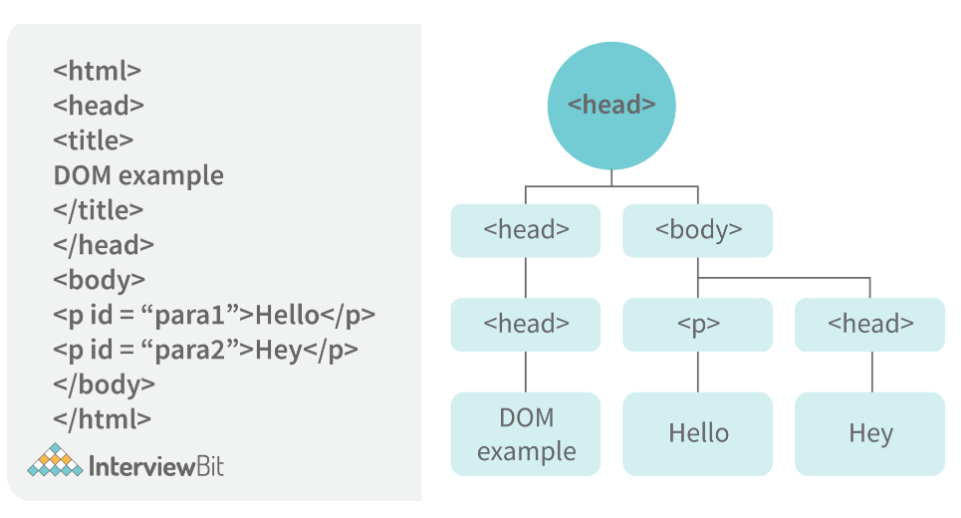
In the code above, we have created a constructor function named Person. Whenever we want to create a new object of the type Person, We need to create it using the new keyword:

**var** person3 = **new** Person("Lilly", 17, "female");

The above line of code will create a new object of the type Person. Constructor functions allow us to group similar objects.

### **232. 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:



### **233. Which method is used to retrieve a character from a certain index?**

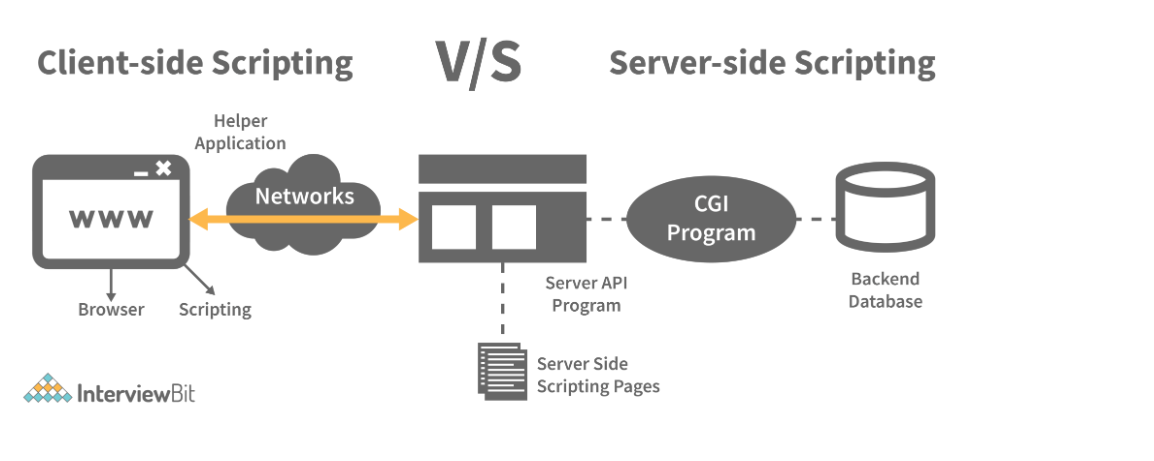
The charAt() function of the JavaScript string finds a char element at the supplied index. The index number begins at 0 and continues up to n-1, Here n is the string length. The index value must be positive, higher than, or the same as the string length.

### **234. What do you mean by BOM?**

Browser Object Model is known as BOM. It allows users to interact with the browser. A browser's initial object is a window. As a result, you may call all of the window's functions directly or by referencing the window. The document, history, screen, navigator, location, and other attributes are available in the window object.

### **235. What is the distinction between client-side and server-side JavaScript?**

Client-side JavaScript is made up of two parts, a fundamental language and predefined objects for performing JavaScript in a browser. JavaScript for the client is automatically included in the HTML pages. At runtime, the browser understands this script.



Server-side JavaScript, involves the execution of JavaScript code on a server in response to client requests. It handles these requests and delivers the relevant response to the client, which may include client-side JavaScript for subsequent execution within the browser.

### **236. 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.

// Traditional function expression**var** multiplyBy2 = **function**(num){

**return** num \* 2;

}// Arrow function expression**var** arrowMultiplyBy2 = num => num \* 2;

If the function takes in only one argument, then the parenthesis () around the parameter can be omitted as shown in the code above.

**var** obj1 = {

valueOfThis: **function**(){

**return** this;

}

}**var** obj2 = {

valueOfThis: ()=>{

**return** this;

}

}

obj1.valueOfThis(); // Will return the object obj1

obj2.valueOfThis(); // Will return window/global object

The biggest difference between the traditional function expression and the arrow function is the handling of ****this****keyword. By general definition, ****this****keyword always refers to the object that is calling the function. As you can see in the code above, ****obj1.valueOfThis()****returns obj1 since ****this****keyword refers to the object calling the function.

In the arrow functions, there is no binding of ****this****keyword. Thiskeyword inside an arrow function does not refer to the object calling it. It rather inherits its value from the parent scope which is the window object in this case. Therefore, in the code above, ****obj2.valueOfThis()****returns the window object.

### **237. What do mean by prototype design pattern?**

The Prototype Pattern produces different objects, but instead of returning uninitialized objects, it produces objects that have values replicated from a template – or sample – object. Also known as the Properties pattern, the Prototype pattern is used to create prototypes.

The introduction of business objects with parameters that match the database's default settings is a good example of where the Prototype pattern comes in handy. The default settings for a newly generated business object are stored in the prototype object.

The Prototype pattern is hardly used in traditional languages, however, it is used in the development of new objects and templates in JavaScript, which is a prototypal language.

### **238. Differences between declaring variables using var, let and const.**

Before the ES6 version of javascript, only the keyword var was used to declare variables. With the ES6 Version, keywords let and const were introduced to declare variables.

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

****Let’s understand the differences with examples:****

**var** variable1 = 23;

**let** variable2 = 89;

**function** **catchValues**(){

console.log(variable1);

console.log(variable2);

// Both the variables can be accessed anywhere since they are declared in the global scope

}

window.variable1; // Returns the value 23

window.variable2; // Returns undefined

* The variables declared with the let keyword in the global scope behave just like the variable declared with the var keyword in the global scope.
* Variables declared in the global scope with var and let keywords can be accessed from anywhere in the code.
* But, there is one difference! Variables that are declared with the var keyword in the global scope are added to the window/global object. Therefore, they can be accessed using window.variableName.  
  Whereas, the variables declared with the let keyword are not added to the global object, therefore, trying to access such variables using window.variableName results in an error.

****var vs let in functional scope****

**function** **varVsLetFunction**(){

**let** awesomeCar1 = "Audi";

**var** awesomeCar2 = "Mercedes";

}

console.log(awesomeCar1); // Throws an errorconsole.log(awesomeCar2); // Throws an error

Variables are declared in a functional/local scope using ****var****and ****let****keywords behave exactly the same, meaning, they cannot be accessed from outside of the scope.

{

**var** variable3 = [1, 2, 3, 4];

}

console.log(variable3); // Outputs [1,2,3,4]

{

**let** variable4 = [6, 55, -1, 2];

}

console.log(variable4); // Throws error

**for**(**let** i = 0; i < 2; i++){

//Do something

}

console.log(i); // Throws error

**for**(**var** j = 0; j < 2; i++){

// Do something

}

console.log(j) // Outputs 2

* In javascript, a block means the code written inside the curly braces ****{}****.
* Variables declared with ****var****keyword do not have block scope. It means a variable declared in block scope ****{}****with the ****var****keyword is the same as declaring the variable in the global scope.
* Variables declared with ****let****keyword inside the block scope cannot be accessed from outside of the block.

****Const keyword****

* Variables with the ****const****keyword behave exactly like a variable declared with the let keyword with only one difference, ****any variable declared with the const keyword cannot be reassigned.****
* Example:

**const** x = {name:"Vivek"};

x = {address: "India"}; // Throws an error

x.name = "Nikhil"; // No error is thrown

**const** y = 23;

y = 44; // Throws an error

In the code above, although we can change the value of a property inside the variable declared with ****const****keyword, we cannot completely reassign the variable itself.

### **239. What is the rest parameter and spread operator?**

Both rest parameter and spread operator were introduced in the ES6 version of javascript.  
  
****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 3,4,5,6console.log(clonedArray1); // Outputs [3,4,5,6]

**let** obj1 = {x:'Hello', y:'Bye'};**let** clonedObj1 = {...obj1}; // Spreads and clones obj1console.log(obj1);

**let** obj2 = {z:'Yes', a:'No'};**let** mergedObj = {...obj1, ...obj2}; // Spreads both the objects and merges itconsole.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.

### **240. In JavaScript, how many different methods can you make an object?**

In JavaScript, there are several ways to declare or construct an object.

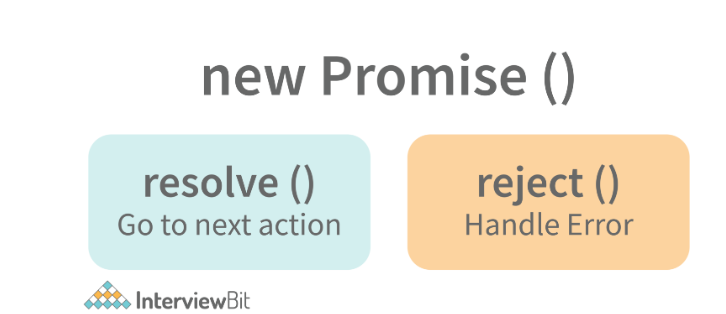
1. Object.
2. using Class.
3. create Method.
4. Object Literals.
5. using Function.
6. Object Constructor.

### **241. What is the use of promises in javascript?**

****Promises are used to handle asynchronous operations in javascript.****  
  
Before promises, callbacks were used to handle asynchronous operations. But due to the limited functionality of callbacks, using multiple callbacks to handle asynchronous code can lead to unmanageable code.  
  
Promise object has four states -

* Pending - Initial state of promise. This state represents that the promise has neither been fulfilled nor been rejected, it is in the pending state.
* Fulfilled - This state represents that the promise has been fulfilled, meaning the async operation is completed.
* Rejected - This state represents that the promise has been rejected for some reason, meaning the async operation has failed.
* Settled - This state represents that the promise has been either rejected or fulfilled.

A promise is created using the ****Promise****constructor which takes in a callback function with two parameters, ****resolve****and ****reject****respectively.



****reject****is a function that will be called, when the async operation fails or if some error occurs.  
  
Example of a promise:  
  
****Promises are used to handle asynchronous operations like server requests, for ease of understanding, we are using an operation to calculate the sum of three elements.****  
  
In the function below, we are returning a promise inside a function:

**function** **sumOfThreeElements**(...elements){

**return** **new** Promise((resolve,reject)=>{

**if**(elements.length > 3 ){

reject("Only three elements or less are allowed");

}

**else**{

**let** sum = 0;

**let** i = 0;

**while**(i < elements.length){

sum += elements[i];

i++;

}

resolve("Sum has been calculated: "+sum);

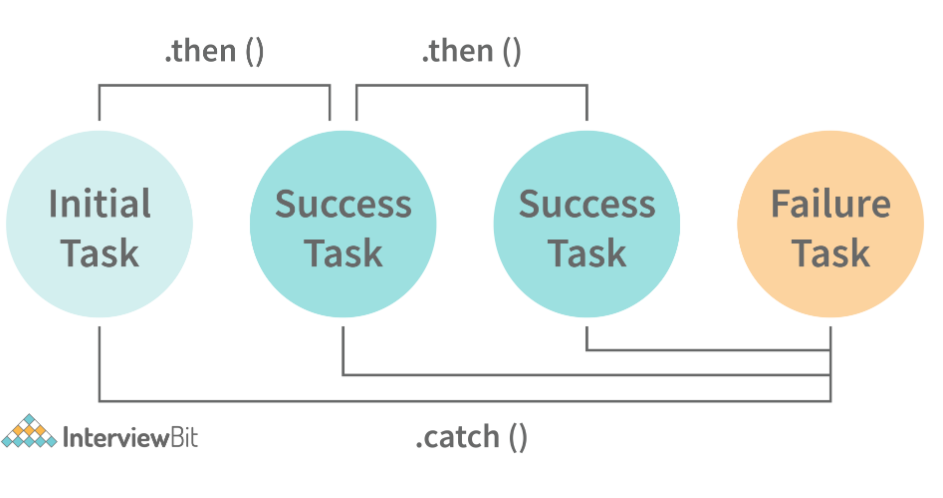
}

})

}

In the code above, we are calculating the sum of three elements, if the length of the elements array is more than 3, a promise is rejected, or else the promise is resolved and the sum is returned.

We can consume any promise by attaching then() and catch() methods to the consumer.



****then()****method is used to access the result when the promise is fulfilled.

****catch()****method is used to access the result/error when the promise is rejected. In the code below, we are consuming the promise:

sumOfThreeElements(4, 5, 6)

.then(result=> console.log(result))

.catch(error=> console.log(error));// In the code above, the promise is fulfilled so the then() method gets executed

sumOfThreeElements(7, 0, 33, 41)

.then(result => console.log(result))

.catch(error=> console.log(error));// In the code above, the promise is rejected hence the catch() method gets executed

### **242. What are classes in javascript?**

Introduced in the ES6 version, classes are nothing but syntactic sugars for constructor functions. They provide a new way of declaring constructor functions in javascript.  Below are the examples of how classes are declared and used:

// Before ES6 version, using constructor functions**function** **Student**(name,rollNumber,grade,section){

this.name = name;

this.rollNumber = rollNumber;

this.grade = grade;

this.section = section;

}

// Way to add methods to a constructor function

Student.prototype.getDetails = **function**(){

**return** 'Name: ${this.name}, Roll no: ${this.rollNumber}, Grade: ${this.grade}, Section:${this.section}';

}

**let** student1 = **new** Student("Vivek", 354, "6th", "A");

student1.getDetails();// Returns Name: Vivek, Roll no:354, Grade: 6th, Section:A

// ES6 version classes**class** **Student**{

**constructor**(name,rollNumber,grade,section){

this.name = name;

this.rollNumber = rollNumber;

this.grade = grade;

this.section = section;

}

// Methods can be directly added inside the class

**getDetails**(){

**return** 'Name: ${this.name}, Roll no: ${this.rollNumber}, Grade:${this.grade}, Section:${this.section}';

}

}

**let** student2 = **new** Student("Garry", 673, "7th", "C");

student2.getDetails();// Returns Name: Garry, Roll no:673, Grade: 7th, Section:C

Key points to remember about classes:

* Unlike functions, classes are not hoisted. A class cannot be used before it is declared.
* A class can inherit properties and methods from other classes by using the extend keyword.
* All the syntaxes inside the class must follow the strict mode(‘use strict’) of javascript. An error will be thrown if the strict mode rules are not followed.

### **243. What are generator functions?**

Introduced in the ES6 version, generator functions are a special class of functions.  
  
****They can be stopped midway and then continue from where they had stopped.****  
  
Generator functions are declared with the ****function\*****keyword instead of the normal ****function****keyword:

**function**\* **genFunc**(){

// Perform operation

}

In normal functions, we use the ****return****keyword to return a value and as soon as the return statement gets executed, the function execution stops:

**function** **normalFunc**(){

**return** 22;

console.log(2); // This line of code does not get executed

}

In the case of generator functions, when called, they do not execute the code, instead, they return a ****generator object****. This generator object handles the execution.

**function**\* **genFunc**(){

**yield** 3;

**yield** 4;

}

genFunc(); // Returns Object [Generator] {}

The generator object consists of a method called ****next()****, this method when called, executes the code until the nearest ****yield****statement, and returns the yield value.  
  
For example, if we run the next() method on the above code:

genFunc().next(); // Returns {value: 3, done:false}

As one can see the next method returns an object consisting of a ****value****and ****done****properties.  Value property represents the yielded value. Done property tells us whether the function code is finished or not. (Returns true if finished).

Generator functions are used to return iterators. Let’s see an example where an iterator is returned:

**function**\* **iteratorFunc**() {

**let** count = 0;

**for** (**let** i = 0; i < 2; i++) {

count++;

**yield** i;

}

**return** count;

}

**let** iterator = iteratorFunc();console.log(iterator.next()); // {value:0,done:false}console.log(iterator.next()); // {value:1,done:false}console.log(iterator.next()); // {value:2,done:true}

As you can see in the code above, the last line returns ****done:true****, since the code reaches the return statement.

### **244. Explain WeakSet in javascript.**

In javascript, a Set is a collection of unique and ordered elements. Just like Set, WeakSet is also a collection of unique and ordered elements with some key differences:

* Weakset contains only objects and no other type.
* An object inside the weakset is referenced weakly. This means, that if the object inside the weakset does not have a reference, it will be garbage collected.
* Unlike Set, WeakSet only has three methods, ****add()****, ****delete()****and ****has()****.

**const** newSet = **new** Set([4, 5, 6, 7]);console.log(newSet);// Outputs Set {4,5,6,7}

**const** newSet2 = **new** WeakSet([3, 4, 5]); //Throws an error

**let** obj1 = {message:"Hello world"};**const** newSet3 = **new** WeakSet([obj1]);console.log(newSet3.has(obj1)); // true

### **245. Why do we use callbacks?**

A callback function is a method that is sent as an input to another function (now let us name this other function "thisFunction"), and it is performed inside the thisFunction after the function has completed execution.

JavaScript is a scripting language that is based on events. Instead of waiting for a reply before continuing, JavaScript will continue to run while monitoring for additional events. Callbacks are a technique of ensuring that a particular code does not run until another code has completed its execution.

### **246. Explain WeakMap in javascript.**

In javascript, Map is used to store key-value pairs. The key-value pairs can be of both primitive and non-primitive types. WeakMap is similar to Map with key differences:

* The keys and values in weakmap should always be an object.
* If there are no references to the object, the object will be garbage collected.

**const** map1 = **new** Map();

map1.set('Value', 1);

**const** map2 = **new** WeakMap();

map2.set('Value', 2.3); // Throws an error

**let** obj = {name:"Vivek"};**const** map3 = **new** WeakMap();

map3.set(obj, {age:23});

### **247. What is Object Destructuring?**

Object destructuring is a new way to extract elements from an object or an array.

* ****Object destructuring:****Before ES6 version:

**const** classDetails = {

strength: 78,

benches: 39,

blackBoard:1

}

**const** classStrength = classDetails.strength;**const** classBenches = classDetails.benches;**const** classBlackBoard = classDetails.blackBoard;

The same example using object destructuring:

**const** classDetails = {

strength: 78,

benches: 39,

blackBoard:1

}

**const** {strength:classStrength, benches:classBenches,blackBoard:classBlackBoard} = classDetails;

console.log(classStrength); // Outputs 78console.log(classBenches); // Outputs 39console.log(classBlackBoard); // Outputs 1

As one can see, using object destructuring we have extracted all the elements inside an object in one line of code. If we want our new variable to have the same name as the property of an object we can remove the colon:

**const** {strength:strength} = classDetails;// The above line of code can be written as:**const** {strength} = classDetails;

* ****Array destructuring:****Before ES6 version:

**const** arr = [1, 2, 3, 4];**const** first = arr[0];**const** second = arr[1];**const** third = arr[2];**const** fourth = arr[3];

The same example using object destructuring:

**const** arr = [1, 2, 3, 4];**const** [first,second,third,fourth] = arr;console.log(first); // Outputs 1console.log(second); // Outputs 2console.log(third); // Outputs 3console.log(fourth); // Outputs 4

### **248. Difference between prototypal and classical inheritance**

Programers build objects, which are representations of real-time entities, in traditional OO programming. Classes and objects are the two sorts of abstractions. A class is a generalization of an object, whereas an object is an abstraction of an actual thing. A Vehicle, for example, is a specialization of a Car. As a result, automobiles (class) are descended from vehicles (object).

Classical inheritance differs from prototypal inheritance in that classical inheritance is confined to classes that inherit from those remaining classes, but prototypal inheritance allows any object to be cloned via an object linking method. Despite going into too many specifics, a prototype essentially serves as a template for those other objects, whether they extend the parent object or not.

### **249. What is a Temporal Dead Zone?**

Temporal Dead Zone is a behaviour that occurs with variables declared using ****let****and ****const****keywords. It is a behaviour where we try to access a variable before it is initialized. Examples of temporal dead zone:

x = 23; // Gives reference error

**let** x;

**function** **anotherRandomFunc**(){

message = "Hello"; // Throws a reference error

**let** message;

}

anotherRandomFunc();

In the code above, both in the global scope and functional scope, we are trying to access variables that have not been declared yet. This is called the ****Temporal Dead Zone****.

### **250. 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.

### **251. Is JavaScript a pass-by-reference or pass-by-value language?**

The variable's data is always a reference for objects, hence it's always pass by value. As a result, if you supply an object and alter its members inside the method, the changes continue outside of it. It appears to be pass by reference in this case. However, if you modify the values of the object variable, the change will not last, demonstrating that it is indeed passed by value.

### **252. Difference between Async/Await and Generators usage to achieve the same functionality.**

* Generator functions are run by their generator yield by yield which means one output at a time, whereas Async-await functions are executed sequentially one after another.
* Async/await provides a certain use case for Generators easier to execute.
* The output result of the Generator function is always value: X, done: Boolean, but the return value of the Async function is always an assurance or throws an error.

### **253. What are the primitive data types in JavaScript?**

A primitive is a data type that isn't composed of other data types. It's only capable of displaying one value at a time. By definition, every primitive is a built-in data type (the compiler must be knowledgeable of them) nevertheless, not all built-in datasets are primitives. In JavaScript, there are 5 different forms of basic data. The following values are available:

1. Boolean
2. Undefined
3. Null
4. Number
5. String

### **254. What is the role of deferred scripts in JavaScript?**

The processing of HTML code while the page loads are disabled by nature till the script hasn't halted. Your page will be affected if your network is a bit slow, or if the script is very hefty. When you use Deferred, the script waits for the HTML parser to finish before executing it. This reduces the time it takes for web pages to load, allowing them to appear more quickly.

### **255. What has to be done in order to put Lexical Scoping into practice?**

To support lexical scoping, a JavaScript function object's internal state must include not just the function's code but also a reference to the current scope chain.

### **256. What is the purpose of the following JavaScript code?**

**var** scope = "global scope";**function** **check**() {

**var** scope = "local scope";

**function** **f**()

{

**return** scope;

}

**return** f;

}

Every executing function, code block, and script as a whole in JavaScript has a related object known as the Lexical Environment. The preceding code line returns the value in scope.

****resolve****is a function that will be called when the async operation has been successfully completed.

### **257. Guess the outputs of the following codes:**

// Code 1:

**function** **func1**(){

setTimeout(()=>{

console.log(x);

console.log(y);

},3000);

**var** x = 2;

**let** y = 12;

}

func1();

// Code 2:

**function** **func2**(){

**for**(**var** i = 0; i < 3; i++){

setTimeout(()=> console.log(i),2000);

}

}

func2();

// Code 3:

(**function**(){

setTimeout(()=> console.log(1),2000);

console.log(2);

setTimeout(()=> console.log(3),0);

console.log(4);

})();

****Answers:****

* ****Code 1****- Outputs ****2****and ****12****. Since, even though ****let****variables are not hoisted, due to the async nature of javascript, the complete function code runs before the setTimeout function. Therefore, it has access to both x and y.
* ****Code 2****- Outputs ****3****, three times since variable declared with ****var****keyword does not have block scope. Also, inside the for loop, the variable i is incremented first and then checked.
* ****Code 3****- Output in the following order:

2431 // After two seconds

Even though the second timeout function has a waiting time of zero seconds, the javascript engine always evaluates the setTimeout function using the Web API, and therefore, the complete function executes before the setTimeout function can execute.

### **258. Guess the outputs of the following code:**

// Code 1:

**let** x= {}, y = {name:"Ronny"},z = {name:"John"};

x[y] = {name:"Vivek"};

x[z] = {name:"Akki"};console.log(x[y]);

// Code 2:

**function** **runFunc**(){

console.log("1" + 1);

console.log("A" - 1);

console.log(2 + "-2" + "2");

console.log("Hello" - "World" + 78);

console.log("Hello"+ "78");

}

runFunc();

// Code 3:

**let** a = 0;**let** b = false;console.log((a == b));console.log((a === b));

****Answers:****  
  
****Code 1****- Output will be ****{name: “Akki”}.****  
  
Adding objects as properties of another object should be done carefully.  
  
Writing ****x[y] = {name:”Vivek”}****, is same as writing ****x[‘object Object’] = {name:”Vivek”}****,  
  
While setting a property of an object, ****javascript coerces the parameter into a string.****  
  
Therefore, since ****y****is an object, it will be converted to ****‘object Object’.****  
  
Both x[y] and x[z] are referencing the same property.  
  
****Code 2****- Outputs in the following order:

11

Nan2-22NaN

Hello78

****Code 3****- Output in the following order due to equality coercion:

truefalse

### **259. Guess the output of the following code:**

**var** x = 23;

(**function**(){

**var** x = 43;

(**function** **random**(){

x++;

console.log(x);

**var** x = 21;

})();

})();

### **Answer:**

Output is ****NaN****.  
  
random() function has functional scope since x is declared and hoisted in the functional scope.  
  
Rewriting the random function will give a better idea about the output:

**function** **random**(){

**var** x; // x is hoisted

x++; // x is not a number since it is not initialized yet

console.log(x); // Outputs NaN

x = 21; // Initialization of x

}

### **260. Guess the outputs of the following code:**

// Code 1

let hero = {

powerLevel: 99,

getPower(){

return this.powerLevel;

}

}

let getPower = hero.getPower;

let hero2 = {powerLevel:42};

console.log(getPower());

console.log(getPower.apply(hero2));

// Code 2

const a = function(){

console.log(this);

const b = {

func1: function(){

console.log(this);

}

}

const c = {

func2: ()=>{

console.log(this);

}

}

b.func1();

c.func2();

}

a();

// Code 3

const b = {

name:"Vivek",

f: function(){

var self = this;

console.log(this.name);

(function(){

console.log(this.name);

console.log(self.name);

})();

}

}

b.f();

Answers:  
  
****Code 1****- Output in the following order:

undefined

42

Reason - The first output is ****undefined****since when the function is invoked, it is invoked referencing the global object:

window.getPower() = getPower();

****Code 2****- Outputs in the following order:

global/window object

object "b"

global/window object

Since we are using the arrow function inside ****func2, this****keyword refers to the global object.  
  
****Code 3****- Outputs in the following order:

"Vivek"

undefined

"Vivek"

Only in the IIFE inside the function ****f****, ****this****keyword refers to the global/window object.

### **261. Guess the outputs of the following code:**

#### ****\*\*Note - Code 2 and Code 3 require you to modify the code, instead of guessing the output.****

// Code 1

(**function**(a){

**return** (**function**(){

console.log(a);

a = 23;

})()

})(45);

// Code 2

// Each time bigFunc is called, an array of size 700 is being created,// Modify the code so that we don't create the same array again and again

**function** **bigFunc**(element){

**let** newArray = **new** Array(700).fill('♥');

**return** newArray[element];

}

console.log(bigFunc(599)); // Array is createdconsole.log(bigFunc(670)); // Array is created again

// Code 3

// The following code outputs 2 and 2 after waiting for one second// Modify the code to output 0 and 1 after one second.

**function** **randomFunc**(){

**for**(**var** i = 0; i < 2; i++){

setTimeout(()=> console.log(i),1000);

}

}

randomFunc();

****Answers -****  
  
****Code 1****- Outputs ****45****.  
  
Even though a is defined in the outer function, due to closure the inner functions have access to it.  
  
****Code 2****- This code can be modified by using closures,

**function** **bigFunc**(){

**let** newArray = **new** Array(700).fill('♥');

**return** (element) => newArray[element];

}

**let** getElement = bigFunc(); // Array is created only once

getElement(599);

getElement(670);

****Code 3****- Can be modified in two ways:  
  
Using ****let****keyword:

**function** **randomFunc**(){

**for**(**let** i = 0; i < 2; i++){

setTimeout(()=> console.log(i),1000);

}

}

randomFunc();

****Using closure:****

**function** **randomFunc**(){

**for**(**var** i = 0; i < 2; i++){

(**function**(i){

setTimeout(()=>console.log(i),1000);

})(i);

}

}

randomFunc();

### **6. Write a function that performs binary search on a sorted array.**

**function** **binarySearch**(arr,value,startPos,endPos){

**if**(startPos > endPos) **return** -1;

**let** middleIndex = Math.floor(startPos+endPos)/2;

**if**(arr[middleIndex] === value) **return** middleIndex;

**elsif**(arr[middleIndex] > value){

**return** binarySearch(arr,value,startPos,middleIndex-1);

}

**else**{

**return** binarySearch(arr,value,middleIndex+1,endPos);

}

}

### **262. Implement a function that returns an updated array with r right rotations on an array of integers a .**

****Example:****

Given the following array: ****[2,3,4,5,7]****  
Perform ****3****right rotations:  
First rotation : [7,2,3,4,5] , Second rotation : [5,7,2,3,4] and, Third rotation: [4,5,7,2,3]

return ****[4,5,7,2,3]****

****Answer:****

function rotateRight(arr,rotations){

if(rotations == 0) return arr;

for(let i = 0; i < rotations;i++){

let element = arr.pop();

arr.unshift(element);

}

return arr;

}

rotateRight([2, 3, 4, 5, 7], 3); // Return [4,5,7,2,3]

rotateRight([44, 1, 22, 111], 5); // Returns [111,44,1,22]

### **263. Write the code for dynamically inserting new components.**

<html>

<head>

<title>inserting new components dynamically</title>

<script type="text/javascript">

function addNode () { var newP = document. createElement("p");

var textNode = document.createTextNode(" This is other node");

newP.appendChild(textNode); document.getElementById("parent1").appendChild(newP); }

</script>

</head>

<body> <p id="parent1">firstP<p> </body>

</html>

### **264. Write the code given If two strings are anagrams of one another, then return true.**

**var** firstWord = "Deepak";**var** secondWord = "Aman";

isAnagram(wordOne, wordTwo); // true

**function** **isAnagram**(one, two) {

//Change both words to lowercase for case insensitivity..

**var** a = one.toLowerCase();

**var** b = two.toLowerCase();

// Sort the strings, then combine the array to a string. Examine the outcomes.

a = a.split("").sort().join("");

b = b.split("").sort().join("");

**return** a === b;

}

### **265. Write the code to find the vowels**

**const** findVowels = str => {

**let** count = 0

**const** vowels = ['a', 'e', 'i', 'o', 'u']

**for**(**let** char **of** str.toLowerCase()) {

**if**(vowels.includes(char)) {

count++

}

}

**return** count

}

### **266. In JavaScript, how do you turn an Object into an Array []?**

**let** obj = { id: "1", name: "user22", age: "26", work: "programmer" };

//Method 1: Convert the keys to Array using - Object.keys()console.log(Object.keys(obj));// ["id", "name", "age", "work"]

// Method 2 Converts the Values to Array using - Object.values()console.log(Object.values(obj));// ["1", "user22r", "26", "programmer"]

// Method 3 Converts both keys and values using - Object.entries()console.log(Object.entries(obj));//[["id", "1"],["name", "user22"],["age", "26"],["work", “programmer"]]

### **267. What is the output of the following code?**

**const** b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

**for** (**let** i = 0; i < 10; i++) {

setTimeout(() => console.log(b[i]), 1000);

}

**for** (**var** i = 0; i < 10; i++) {

setTimeout(() => console.log(b[i]), 1000);

}

****Ans.****

12345678910undefinedundefinedundefinedundefinedundefinedundefinedundefinedundefinedundefinedundefined

**Objects**

var x = 100;//number literals

        var firstname = "manjula";//string literals

        var latname = "devarala";

        var person = {

            firstname:"manjula",

            lastname:"devaral"

        }//object literals //JSON

        var marks = [10,20,30,40]; //Array literals

    //Example

        var person = {

            "firstname":"manjula",

            "lastname":"devarala",

            "dob":"12/05/1997",

            "salary":"30000",

            "location":"hyderabad",

            "gender":"female"

        }

        var x = [23,34,59,60,78];

        for(let i=0;i<=4;i++){

            console.log(x[i])

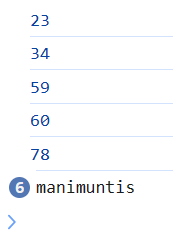
        }

        for (j in person)

        {

            console.log("manimuntis");

        }



   var person = {

            "firstname":"manjula",

            "lastname":"devarala",

            "dob":"12/05/1997",

            "salary":"30000",

            "location":"hyderabad",

            "gender":"female"

        }

        var x = [23,34,59,60,78];

        for(let i=0;i<=4;i++){

            console.log(x[i])

        }

        for (j in person)

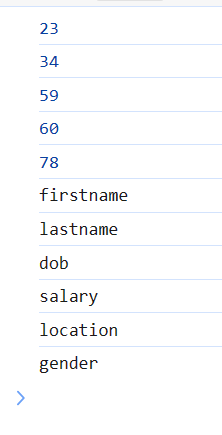
        {

            console.log("manimuntis");

            console.log(j);

            console.log(j,person[j]);

        }



 //Example

        var person = {

            "firstname":"manjula",

            "lastname":"devarala",

            "dob":"12/05/1997",

            "salary":"30000",

            "location":"hyderabad",

            "gender":"female"

        }

        var x = [23,34,59,60,78];

        for(let i=0;i<=4;i++){

            console.log(x[i])

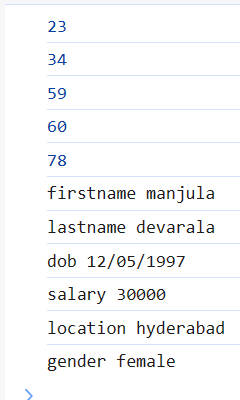
        }

        for (j in person)

        {

            console.log(j,person[j]);

        }



**Array Object**

 var persons = [

            {

                "firstname":"mallesh",

                "lastname":"devararal",

                "dob":"21/01/2001",

                "salary":"nothing",

                "location":"htderabad"

            },

            {

                "firstname":"malli",

                "lastname":"deva",

                "dob":"21/01/2000",

                "salary":"nothing",

                "location":"htderabad"

            },

            {

                "firstname":"maha",

                "lastname":"devara",

                "dob":"21/01/2002",

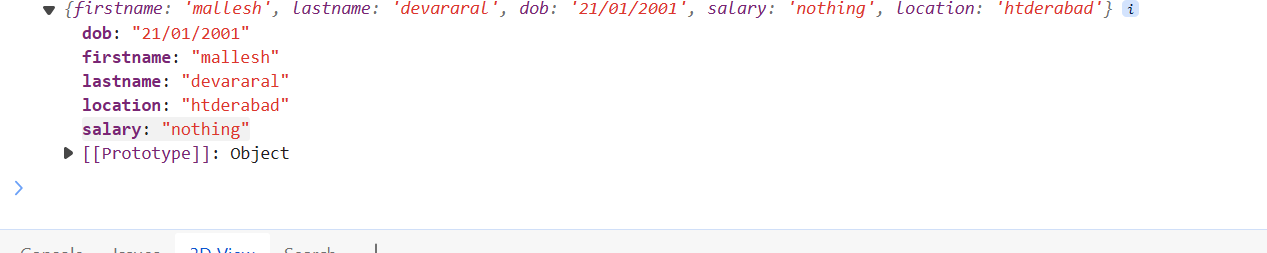
                "salary":"nothing",

                "location":"htderabad"

            }

        ]

        console.log(persons[0]);



 var persons = [

            {

                "firstname":"mallesh",

                "lastname":"devararal",

                "dob":"21/01/2001",

                "salary":"nothing",

                "location":"htderabad"

            },

            {

                "firstname":"malli",

                "lastname":"deva",

                "dob":"21/01/2000",

                "salary":"nothing",

                "location":"htderabad"

            },

            {

                "firstname":"maha",

                "lastname":"devara",

                "dob":"21/01/2002",

                "salary":"nothing",

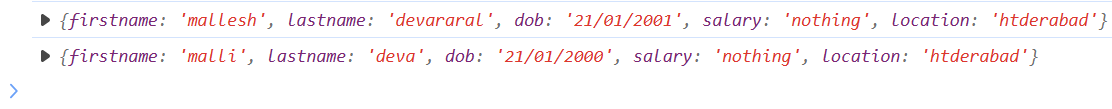
                "location":"htderabad"

            }

        ]

        console.log(persons[0]);

        console.log(persons[1]);



   var persons = [

            {

                "firstname":"mallesh",

                "lastname":"devararal",

                "dob":"21/01/2001",

                "salary":"nothing",

                "location":"htderabad"

            },

            {

                "firstname":"malli",

                "lastname":"deva",

                "dob":"21/01/2000",

                "salary":"nothing",

                "location":"htderabad"

            },

            {

                "firstname":"maha",

                "lastname":"devara",

                "dob":"21/01/2002",

                "salary":"nothing",

                "location":"htderabad"

            }

        ]

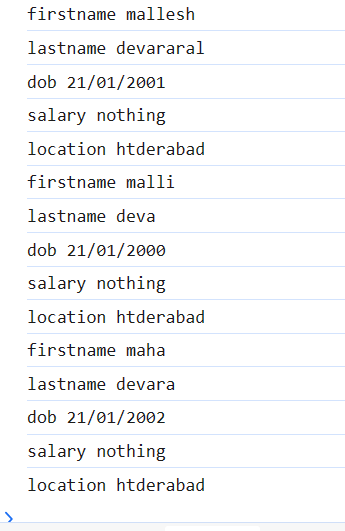
      for(var i=0;i<=3;i++){

        for (j in persons[i]){

            console.log(j,persons[i][j]);

        }

      }



<h2>Objects are 2</h2>

    <ul>

        <li>document</li>

        <li>window</li>

    </ul>

    <p>it is in browser</p>

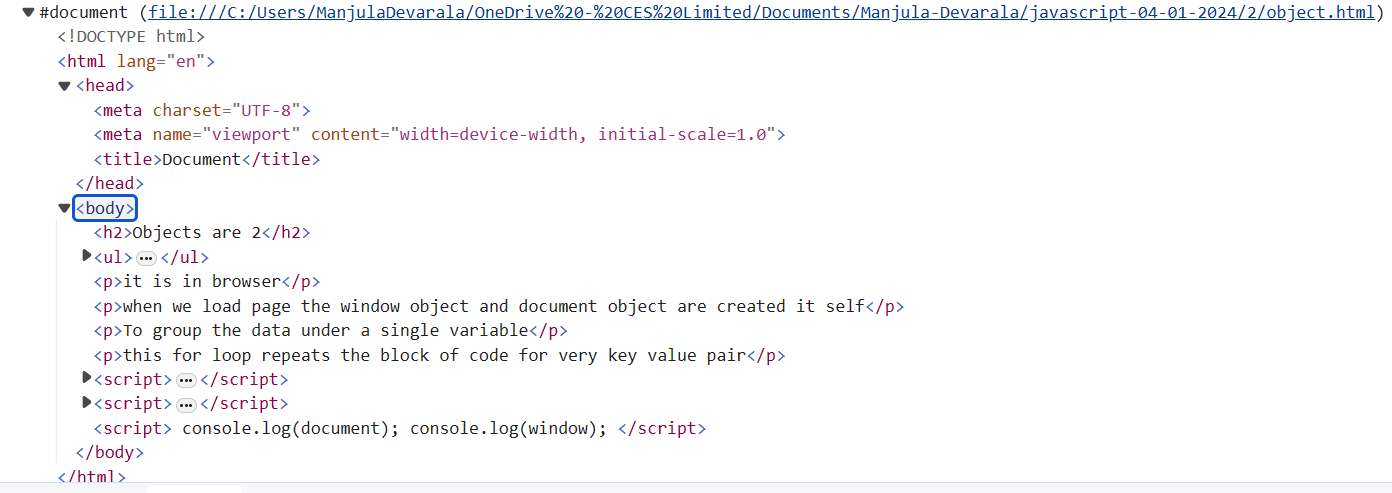
    <p>when we load page the window object and document object are created it self</p>

    <p>To group the data under a single variable</p>

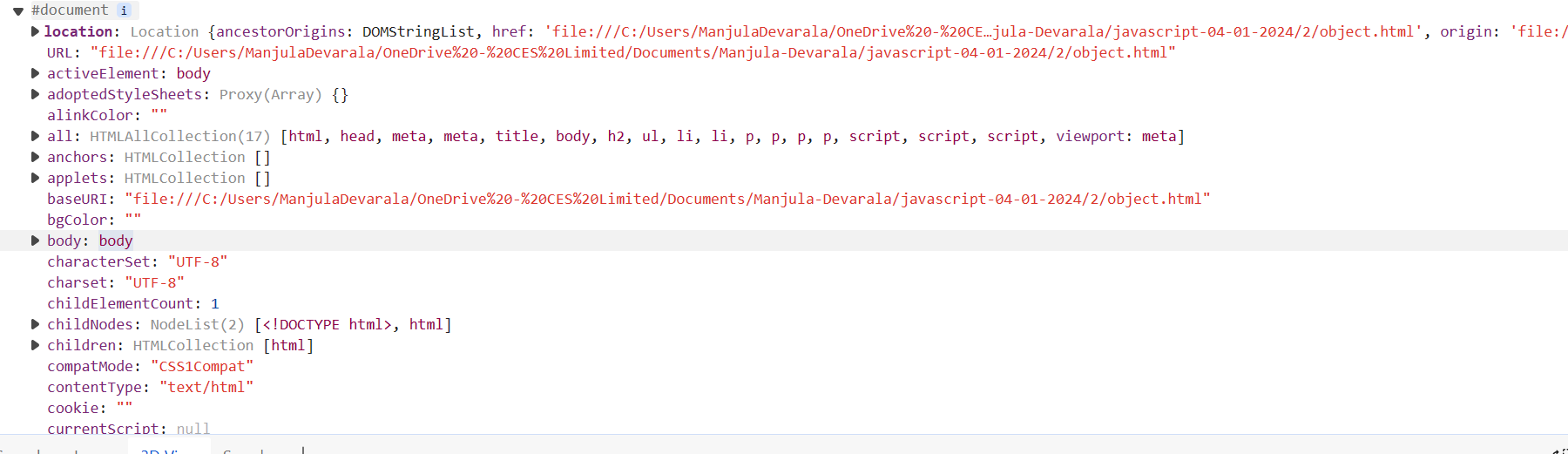
    <p>this for loop repeats the block of code for very key value pair</p>

 console.log(document);

Output:



 console.dir(document);



**Important Notes**

//array lo unde values ni itterate checukotaniki for loop undi

1.For iterating through array values, we can use a for loop.

2.For iterating object values, we can use a for-in loop.

**268.What is JavaScript?**

Ans)JavaScript it can control the style and content of the html .

**269.What is the difference between writing java-script in head and body?**

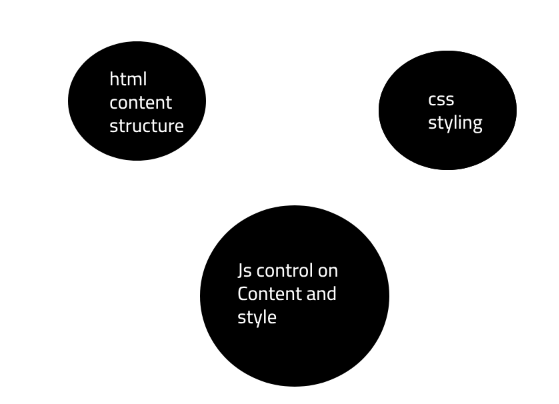
Ans)If I am writing JavaScript in head and body is

i) If I write java-script in HEAD section function are executed first after id is loaded so function doesn't work so no id is called .

Ii)If I write java-script in BODY section first id is loaded after function executed so output works properly.

**270.What is the purpose of Javascript?**

Ans)To create the logic and control the content of HTML structure and css



1.)1.hml and 1.js

console.log("hello world");

**Output:**hello world

2)

var x;

console.log(x);

**Output:**undefined

3)

var x=100;

console.log(x);

console.log(typeof(x));

Output:100

number

271.

x="manjula";

console.log(x);

console.log(typeof(x));

Output:manjula

string

**272.Var and let differnce**

Ans)Scope level difference

var x;

let y;

{

    var z=199;

    let w=133;

    console.log(z);

    console.log(w);

}

console.log(z);

console.log(w);

Output:199

133

199

1.js:28 Uncaught ReferenceError: w is not defined

at 1.js:28:13

**273.Block level scope?**

Ans)

for(let i=0;i<=10;i++)

{

    console.log(i);

}

console.log(i);

Output:1

2

3

4

5

6

7

8

9

10

1.js:23 Uncaught ReferenceError: i is not defined

at 1.js:23:13

/\*let example start\*/

if(10<30){

    let p=99;

}

console.log(p);

Output:error

/\*let example end\*/

/\*var example start\*/

if(10<30){

    varp=99;

}

console.log(p);

**Output**:99

/\*var example end\*/

**274.**

Ans)

for(var i=0;i<=10;i++)

{

    console.log(i)

}

console.log(i)

**Output:0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

**10**

**11**

**275.For loop**

Ans)for (initializer; condition; iterator) { *// statements* }

### 1) initializer

The for statement executes the initializer only once the loop starts. Typically, you declare and initialize a local loop variable in the initializer.

### 2) condition

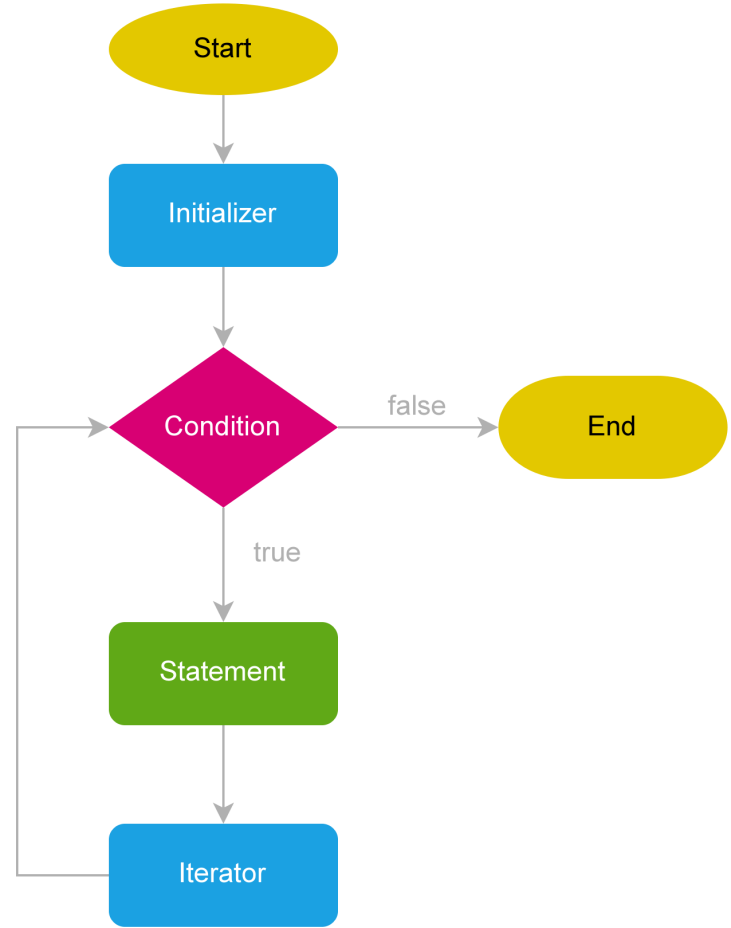
The condition is a boolean expression that determines whether the for should execute the next iteration.

The for statement evaluates the condition before each iteration. If the condition is true (or is not present), it executes the next iteration. Otherwise, it’ll end the loop.

### 3) iterator

The for statement executes the iterator after each iteration.

The following flowchart illustrates the for loop:

[](https://www.javascripttutorial.net/wp-content/uploads/2022/01/javascript-for-loop.svg)

**Example:** for (let i = 1; i < 5; i++) { console.log(i); }

**Output:**1 2 3 4

**Link:**

[JavaScript for Loop By Examples (javascripttutorial.net)](https://www.javascripttutorial.net/javascript-for-loop/)

**Const**

i)The const keyword creates a read-only reference to a value. The readonly reference cannot be reassigned but the value can be changed.

ii)The const declaration creates an immutable reference to a value. It does not mean the value it holds is immutable — just that the variable identifier cannot be reassigned. For instance, in the case where the content is an object, this means the object's contents (e.g., its properties) can be altered. You should understand const declarations as "create a variable whose identity remains constant", not "whose value remains constant" — or, "create immutable [bindings](https://developer.mozilla.org/en-US/docs/Glossary/Binding)", not "immutable values".

//Example

var a =200

const x = a;

console.log(x);

a = 900;

console.log(x);

Output:200

200

//x we cant change a value we can change

//Example

const o = {a:300,b:400};

o={k:"manjula"};

console.log(o);

//o is const a and b values we can change

Constants are block level scope.

//scope on const

if(10<90){

    const r = 88;

}

console.log(r);

Output:error

//error

Single line comments

Ans)Sinle line comments start with //

Any text between // and the end of the line will be igonred by javascript (will not executed).

Example :

//change heading

Notes:

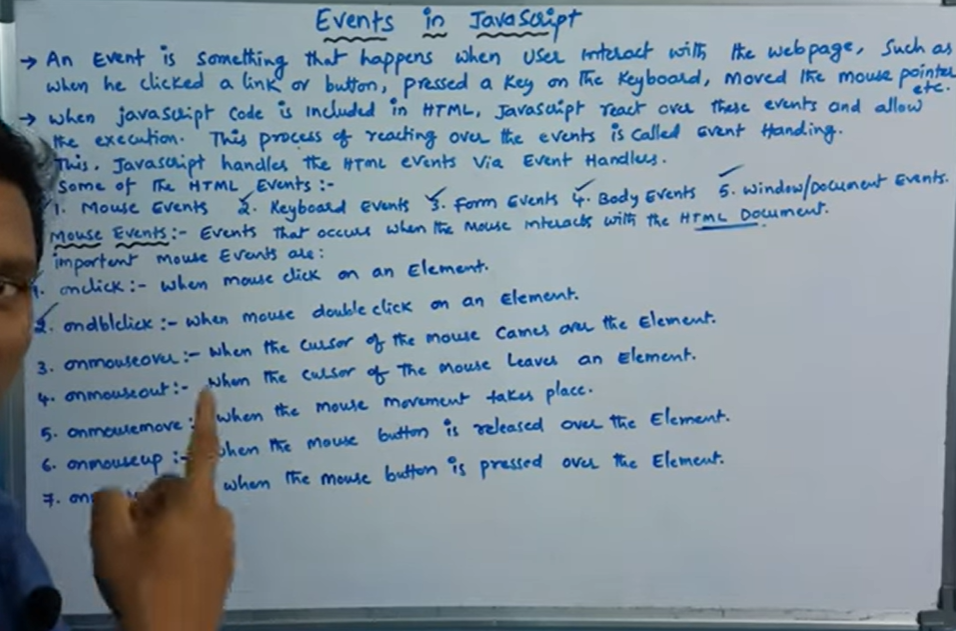
**id=”d1”**

1)onclick=”xxx()” based on above id we

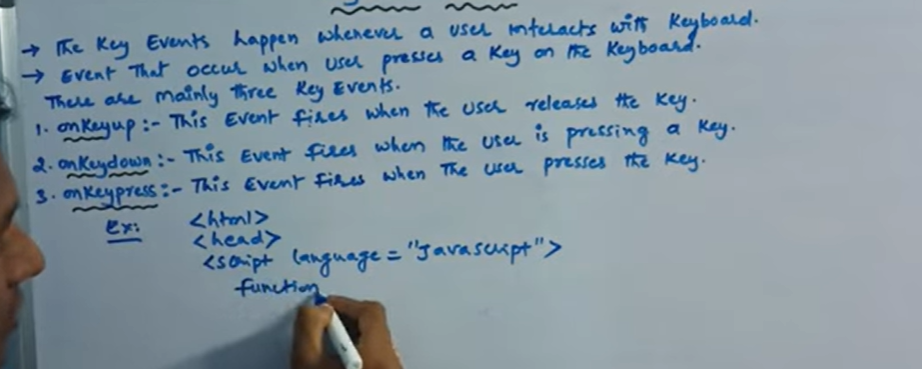
2)onmouseover=”xxx()”

Events in javascript

Mouse events



Keybaoard events



1. Software

Software is a program or collection of programs which will solve a problems

Software are 2 types

i)Standalone application

ii)Client-server

From webserve to webpage how to get for that we are using

1. web server
2. Address bar(url)
3. Link click
4. Form submition

Constructor functions are 2 types

1. Window
2. Document

**Encapsulation:**

Binding the data and methods together in a small unit is called encapsulation.

Wrapping data and member functions together in a particular unit is called encapsulation.

**What are the features of object-oriented principles**

1. Encapsulation
2. Obstartion
3. Inheritence
4. Polymorfision

**Data Obstarction**

Hiding the implementation details.

Function:

1. Write a function that can add 2 numbers.
2. Write a function which can display the factorial of a number

Javascript does not have classes,but we still use objects to develop our applications. Therefore, we definitely need to create some entities based on functions.

Every function can be a constructor function if it creates objects with properties and methods.

Every function is a normal function or constructor if it is able to create memory,properties, and methods.

Function functioname(parametres){

//code to be eaecuted

}

**Constructor function**:

Constructor functions are nothing but normal functions to which we can create objects

 function Employee(n,a,l,o, e, q){//arguments

            this.name = n,

            this.age =a,

            this.location = l,

            this.occupation = o,

            this.experience = e

        }//constructor functions

We are using **this** so memory will be allocated (age and location got memory). Firstname and lastname are normal variables. These are not getting memory; these are private variables.

  function Employee(fn,ln,a,g,l,q){

            var firstname = fn;

            var lastname = ln;

            this.age = a;//properties--memory created

            this.location = l;//properties--memory created

        }

        var e1 = new Employee("praveen","gubbala",35,"male","pune","masters");

        var e2 = new Employee("suresh","gubba",38,"male","hyderabad","btech");

        console.dir(e1);

        console.dir(e2);

**Arrays**

var x = [12,23,34,45];

        console.log(x[3]);

Output:45

X[0]=12

X[1]=23

X[2]=34

X[3]=45

 var y = [12,13,14,45,65,76,98]

        for(let i=0;i<=5;i++){

            console.log(y[i]);

        }

Output:

12

13

14

45

65

76

//maximum value from array

        var z = [98,87,76,65,78,89];

        var max = z[0];

        for(let  j=1;j>=5;j++)

        {

            if(max<x[j]){

                max=[j]

            }

        }

        console.log(max)

Output:98

278.What is the difference between JavaScript and JScript?

Ans)

JavaScript and JScript are both scripting languages, but there are some historical and technical differences between them:

****Origin and History:****

* + ****JavaScript:**** JavaScript was originally developed by Netscape in the mid-1990s. It was first introduced in Netscape Navigator 2.0 in 1995.
  + ****JScript:**** JScript is Microsoft's implementation of ECMAScript, which is the standard upon which JavaScript is based. JScript was introduced as a scripting language for Internet Explorer around the same time as JavaScript.

****Naming:****

* + ****JavaScript:**** The name "JavaScript" is a trademark of Oracle Corporation, which acquired Sun Microsystems (the original owner of JavaScript). However, the language is standardized as ECMAScript by the ECMA International organization.
  + ****JScript:**** JScript is the name used by Microsoft for its implementation of ECMAScript.

****Versions and Compatibility:****

* + ****JavaScript:**** The language has evolved over the years with new versions and features. It is widely supported in web browsers and has become a standard for client-side scripting on the web.
  + ****JScript:**** Microsoft's JScript has different versions that correspond to different versions of Internet Explorer. It was primarily used for client-side scripting in the context of web development for Internet Explorer.

****Usage:****

* + ****JavaScript:**** JavaScript is a versatile scripting language used not only in web development for client-side scripting but also for server-side development (Node.js), mobile app development, and more.
  + ****JScript:**** JScript was historically more tied to the Internet Explorer browser and Windows scripting, but its usage has diminished as Internet Explorer itself has been deprecated in favor of other browsers.

****Features and Differences:****

* + While the core language features are similar, there might be some differences in the implementation details and additional features provided by each in terms of browser or platform-specific functionality.

In summary, JavaScript is a more generic term referring to the scripting language standardized as ECMAScript, while JScript specifically refers to Microsoft's implementation of ECMAScript. JavaScript is more widely used and supported in various contexts beyond web development, while JScript has historically been associated with Internet Explorer and Windows scripting. With the decline of Internet Explorer and the rise of modern browsers, the use of JScript has decreased, and JavaScript has become the predominant scripting language for web development

1. Print number

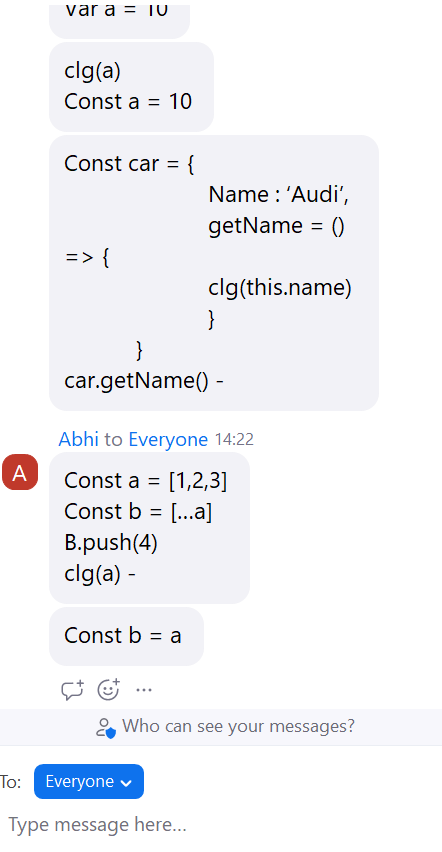
 for(let j=0;j<=10;j++){

            document.write(j);

        }

Output:01234567891011 //printed in in page level

278.Mobile first approach?

.

279.What is the use of promises

281.Why we use $ in jquery

Ans)

The $ symbol is a fundamental aspect of jQuery, serving as the shorthand alias for jQuery, the main function object of the jQuery library. Its use is central to jQuery's design, offering a concise and highly recognizable syntax that enhances the ease and readability of code. Here are several reasons why $ is used in jQuery:

**Convenience and Brevity**: $ is short, simple, and easy to type, which makes it very convenient for developers. It reduces the verbosity of code, making scripts cleaner and easier to read.

**Function Overloading:** The $ function in jQuery is overloaded to perform multiple tasks based on the type of argument passed to it. It can select elements, create elements, or run a function when the document is ready, among other things. Here are a few examples:

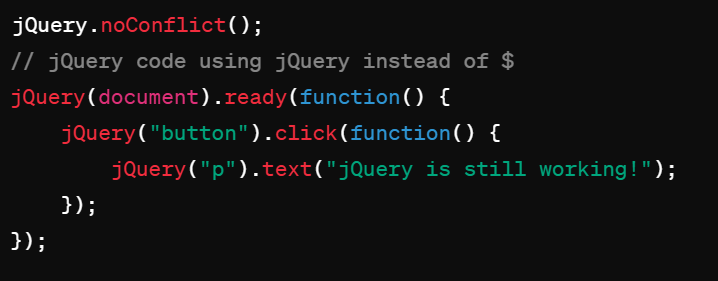
$('#id') - Selects an element with a specific ID.

$('<div>') - Creates a new <div> element.

$(function() {}) - A shorthand for $(document).ready(function() {}), which executes the function when the DOM is fully loaded.

**Consistency with Other Libraries:** The use of $ is not unique to jQuery. Other libraries, such as Prototype (which was more popular around the time jQuery was first released), also use the $ symbol. jQuery's choice allows it to be syntactically similar and somewhat interchangeable in contexts where another library may be swapped out.

**Namespacing and Conflict Resolution**: Although $ is used as the default alias for jQuery, it can lead to conflicts when multiple JavaScript libraries that use the $ symbol are included in the same project. jQuery provides a mechanism to relinquish control of the $ symbol via $.noConflict(). This function restores the $ back to its original value, which can help in integrating jQuery with other libraries that use the $ symbol.



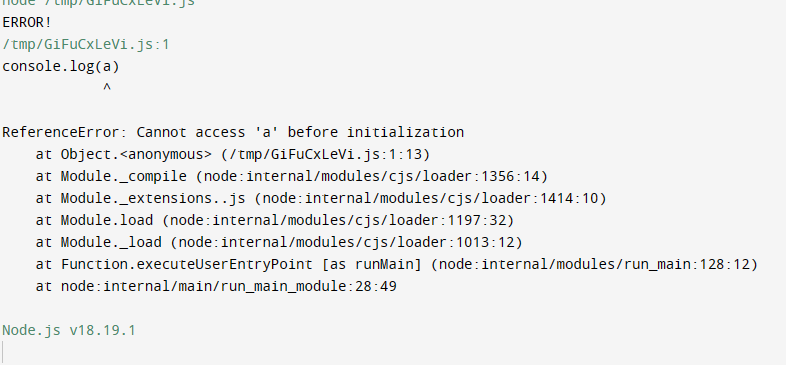
After calling $.noConflict(), you can still use jQuery itself by simply using the jQuery keyword instead of $.

In summary, the $ symbol is synonymous with jQuery and provides a quick, easily recognizable way to access jQuery's functionality. Its simplicity is a large part of what makes jQuery appealing, especially for those who are new to JavaScript or who need to write scripts quickly and efficiently.

282.

console.log(a)

const a = 10



283)

console.log(a)

var a = 10

**Output:**undefined

284)

const a =[1,2,3]

const b = [...a]

b.push(4)

console.log(a)

**Output:**[ 1, 2, 3 ]

285)

const a =[1,2,3]

const b = a

b.push(4)

console.log(a)

**Output:**[ 1, 2, 3, 4 ]

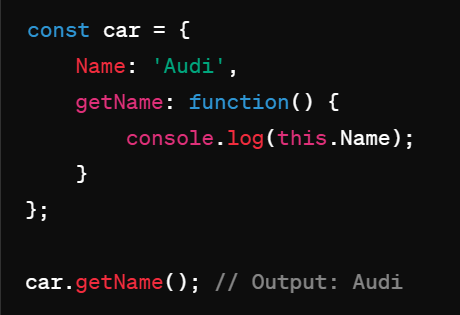
286)

### **Issues and Corrections**

1. ****Syntax Error****: The object definition uses an equals sign (**=**) for methods which is incorrect in JavaScript object literal syntax. Method definitions within an object should use the colon (**:**) followed by the function definition.
2. ****Arrow Function and** this**: Arrow functions do not have their own **this** context. Instead, they inherit **this** from the surrounding lexical scope. In the context of your example, when you define **getName** as an arrow function inside an object literal, **this** does not bind to the object itself but to the enclosing scope (likely the global scope or the scope of the module if in strict mode).

### **Corrected Code**

If you are aiming to access the **Name** property of the **car** object inside the **getName** method, you should use a regular function instead of an arrow function:

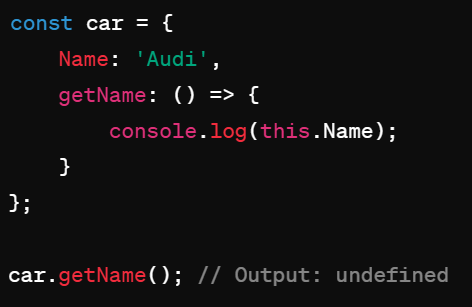


This will output **Audi** because **this** within the function now correctly refers to the **car** object, allowing access to its **Name** property.

### **Using an Arrow Function**

If you were to use an arrow function as initially intended but with the correct syntax, here's what it would look like:

287)



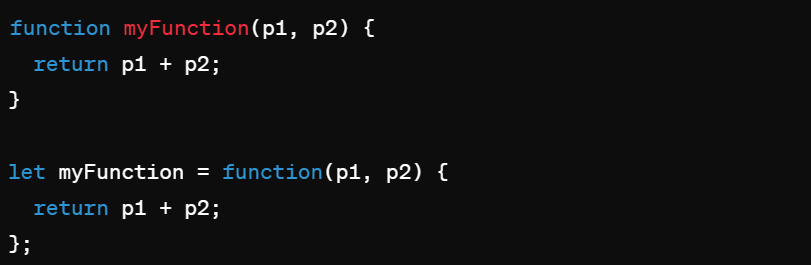
This will output undefined because this inside the arrow function does not point to the car object but to the enclosing scope's this, which does not have a Name property.

**Conclusion**

When defining methods in objects that need to access the object's properties using this, it's important to use function expressions or method definitions (ES6 shorthand syntax) instead of arrow functions to ensure that this behaves as expected.

288.what is the differnce between normal function and arrow function?

Ans)****Normal Function:****



****Arrow Function:****

* More concise, especially for single-expression functions.
* Does not have its own bindings to the **this**, **arguments**, **super**, or **new.target** keywords.



### **2. this Keyword Behavior**

****Normal Function:****

* The value of **this** is determined by how the function is called. It can be global, window (in a browser), the object that the function is a method of, or the new instance if the function is used as a constructor.

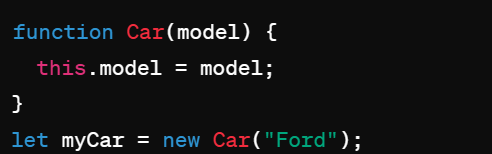
****Arrow Function:****

* Does not have its own **this** context; instead, it captures the **this** value of the enclosing context at the time it is created. This makes it ideal for use in callbacks and methods where you want to retain the context of the outer function.

### **3. Constructor Use**

****Normal Function:****

* Can be used as a constructor when used with the **new** keyword.



In JavaScript, the choice between using a normal (traditional) function and an arrow function often depends on the specific requirements of the code, especially concerning the behavior of the **this** keyword, succinctness, and function capabilities. Here’s a breakdown of the key differences between normal functions and arrow functions:

### **1. Syntax**

****Normal Function:****

* More verbose.
* Can be declared in various ways, including function declarations and function expressions.

javascript

Copy code

function myFunction(p1, p2) {

return p1 + p2;

}

let myFunction = function(p1, p2) {

return p1 + p2;

};

****Arrow Function:****

* More concise, especially for single-expression functions.
* Does not have its own bindings to the **this**, **arguments**, **super**, or **new.target** keywords.

javascript

Copy code

let myFunction = (p1, p2) => p1 + p2;

### **2. this Keyword Behavior**

****Normal Function:****

* The value of **this** is determined by how the function is called. It can be global, window (in a browser), the object that the function is a method of, or the new instance if the function is used as a constructor.

****Arrow Function:****

* Does not have its own **this** context; instead, it captures the **this** value of the enclosing context at the time it is created. This makes it ideal for use in callbacks and methods where you want to retain the context of the outer function.

### **3. Constructor Use**

****Normal Function:****

* Can be used as a constructor when used with the **new** keyword.

javascript

Copy code

function Car(model) {

this.model = model;

}let myCar = new Car("Ford");

****Arrow Function:****

* Cannot be used as a constructor and will throw an error if used with the **new** keyword.

### **4. arguments Object**

****Normal Function:****

* Has access to an **arguments** object, which is an array-like object corresponding to the arguments passed to the function.

****Arrow Function:****

* Does not have its own **arguments** object. Any reference to **arguments** within the arrow function will take its value from the enclosing function, if available.

### **5. Return Behavior**

****Normal Function:****

* Requires explicit return statements (unless used as a constructor).

****Arrow Function:****

* Allows a shorthand return when there’s a single expression (implicit return).



### **6. Method Definitions in Objects**

****Normal Function:****

* Suitable for method definitions in objects where **this** should refer to the object itself.

****Arrow Function:****

* Not suitable for method definitions if you need **this** to refer to the object.

### **Conclusion**

The choice between a normal function and an arrow function should be based on the specific needs related to **this** behavior, concise syntax, and whether the function needs to access the **arguments** object or be used as a constructor. Arrow functions are particularly useful for short callbacks or methods where preserving the lexical scope of **this** is necessary.

**289.What is output?**

let arr = [];

if(arr){ console.log("hello");

}

else{

console.log("bye")

}

**Output:**hello

**Note:In javascript** empty object or empty array is converted boolean value true it shows so will get output is hello .

**Undefined,null,NAN,0,””** -- false

**Empty array or any thing it will show** -- true

**290.**

function toBoolean(input) {

return Boolean(input);

}

console.log(toBoolean("")); **//output:** false

**291.General sibiling selector and adjacent sibiling selector differnce?**

Ans)

<div>

<h1>

</div>

div~p{

background-color:blue;

}

div+p{

background:blue;

}

**292.What is the output of below code?**

let arr = ["John", 25, "App"]

let obj = {...arr}

console.log(obj)

**//output:**{ '0': 'John', '1': 25, '2': 'App' }

**293.What is the output of below code?**

Ans)

const product = "Real me 8pro";

var price = 19000;

let ram ="16 gb";

{

const product ="Real me 8pro";

var price = 24000;

let ram ="8 gb";

}

console.log(product, price, ram)

**//output:**

Real me 8pro

24000

16 gb