SHERYIANS CODING SCHOOL

PRESENTS

Bible of JavaScript

Whoever follows this blindly achieves the greatness and unbeatable wisdom in JavaScript.

Interview Bible

Interview Bible

NOW, WHAT YOU'RE GOING TO READ ARE THE QUESTIONS USUALLY ASKED IN THE INTERVIEWS, THE ONLY PERSPECTIVE WE HAD WHILE WRITING THE ANSWERS WAS,

"KEEPING THEM TO THE POINT AND SHORT"

INTERVIEW QUESTIONS

WHAT IS THE DIFFERENCE BETWEEN LET, VAR, CONST ?

let & cons originate from new js version es6, and **var** is from older version of js, they both exist in current JavaScript and can be used, but they behave differently, if you create a variable with var keyword, the variable can be accesed in whole function, let and const exists in the curly braces { }, vars are attached to window object but let and const are not.

CAN YOU EXPLAIN DIFFERENCE BETWEEN = = AND = = = ?

== operator is called equality operator while === operator is called strict equality operator.
== only checks for value and doesn't checks for type
=== checks for both value and type.

CAN YOU EXPLAIN WHAT IS HIGHER ORDER FUNCTIONS ?

Higher Order Functions Are The Functions Which Accept A Function In A Parameter Or Return A Function Or Both.

Imp For Example : ForEach Method Always Takes Another Function Inside It, So ForEach Is A Higher Order Function

INTERVIEW QUESTIONS

WHAT IS FIRST CLASS FUNCTIONS ?

A Language Is Said To Have First Class Functions When The Functions In That Language Are Treated As Normal Values Or Like Variables, You Can Save Them, You Can Pass Them As Arguments To Another Functions.

WHAT ARE CONSTRUCTOR FUNCTIONS ?

Any Normal Function In Js Which Whenever Called With "New" Keyword, Returns An Object, If We Use "This" Keyword Inside That Function, It Returns An Object With All Of The Properties And Methods Mentioned Inside That Function With This Keyword, Such Function Is Called Constructor Function.

Exmp

```
function abcd( ){
    this.name = "harsh";
}
```

constructor function

```
var person1 = new abcd( );
```

new keyword infront of function call makes a new blank object and returns to the person1 variable.

INTERVIEW QUESTIONS

WHAT IS THE NEW KEYWORD, HOW YOU'LL EXPLAIN IT ?

In JavaScript, The New Keyword Is Used To Create An Instance Of An Object Based On A Constructor Function. The New Keyword Creates A New Empty Object And Sets The This Keyword To Point To The New Object.

In Order To Understand New Keyword, Do This:

Whenever You Encounter A New Keyword Always Imagine A Blank Pair Of Curly Braces { } Which Means A Blank Object And Now Move Inside The Function Which Is Called Just After The New Keyword, Inside That Function All Of The This Keyword Instances Will Add Properties And Methods Inside Your Blank Object Created By The New Keyword.

Exmp

```
function abcd( ){
     this.name = “harsh”;
}
var person1 = new abcd( );
```

2. put { } in place of this keyword we add name property in object

```
{
   name: “harsh”
}
```

1. new keyword make a blank { }

INTERVIEW QUESTIONS

WHAT IS IIFE ?

IIFE Stands For Immediately Invoked Function Expression. It Is A Way To Create A Function And Immediately Execute It Without Needing To Call It Later. Here's An Example:

Exmp

Normal Function

```
function abcd() {
  // some code
}
```

IIFE

```
(function() {
  // wrap function with () and then call it ()
})();
```

we straightaway executed it

IIFEs Are Commonly Used To Create A Private Scope For Your Code, So That Variables And Functions Defined Inside The IIFE Are Not Accessible From Outside The IIFE.

Code

```
let globalVariable = (function() {
  let privateVariable = 0;
})();
```

this is a private variable we can not access it outside or via console

INTERVIEW QUESTIONS

EXPLAIN MAP, FILTER REDUCE.

Map : Suppose You Have To Perform A Particular Task On Every Member Of The Array, Multiply Every Element Of The Array With 2 And Then Place The Answers In The New Array And Eventually Return That New Array, And That's Exactly What Map Does

Exmp

```
var numbers = [1, 2, 3, 4, 5];
var doubledNumbers = numbers.map(function(value){
     return value*2;
});
```

map always returns something, if you don't return anything that would give error.

Key Higlights

i) map looks very similar to forEach loop, but inside it there's something always returned, and whatever is returned gets placed in the resultant array.

INTERVIEW QUESTIONS

EXPLAIN MAP, FILTER REDUCE.

Filter : Suppose You Have An Array And You Want To Filter Out (Not Accept) Elements In New Array, That's Where Filter Comes In, Let's Say Array Contains Many Numbers We Want To Extract Only Those Numbers Which Are Greater Than 5 That's Where Filter Is Used.

Exmp

```
var numbers = [1, 2, 3, 4, 5];
var filteredNums = numbers.filter(function(value){
    return value>5;
});
```

filter always expects something which returns true or false, value>5 can either be true or false

Key Higlights

i) filter looks very similar to map, but inside it whatever it returns should always be boolean which means true or false, if true is returned, that particular array value is accepted in new array and otherwise it's not.

INTERVIEW QUESTIONS

EXPLAIN MAP, FILTER REDUCE.

Reduce : Ek Array Ki Saari Value Par Kuchh Perform Karke Ek Value Banane Ke Liye We Use Reduce, Example : Add All Values Of Array, When We Add All Values, It Gives Us The Sum Which Is A Single Value, Any Such Case Where We Need To Convert Array Into A Single Value, That's Where Reduce Is Used.

Exmp

```
const myArray = [1, 2, 3, 4, 5];
const result = myArray.reduce((acc, val) => {
    return acc = acc+val;
});
```

WHAT IS ACC ?

acc is accumulator it contains the build up answer, example if we add 1,2 acc will contain 3 and now when we add 3 on previous sum acc will contain 6 which means the buildup answer is acc

INTERVIEW QUESTIONS

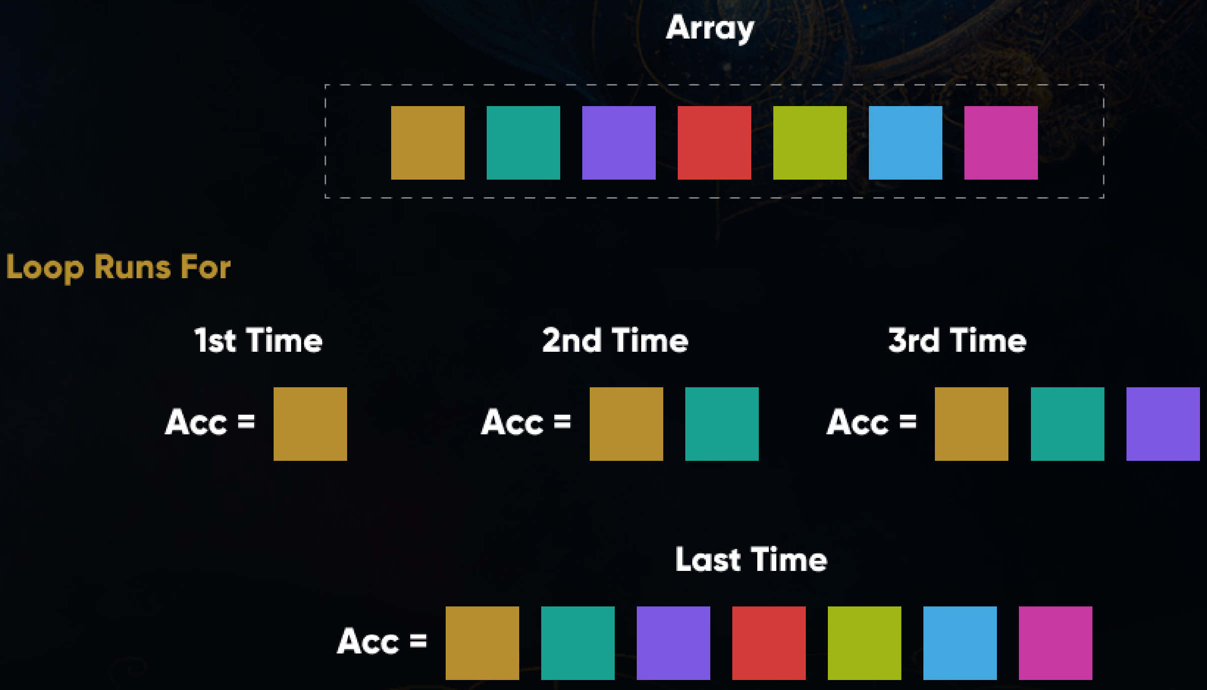
EXPLAIN MAP, FILTER REDUCE.

Exmp

```
const myArray = [1, 2, 3, 4, 5];
const result = myArray.reduce((acc, val) => {
    return acc = acc+val;
});
```

WHAT IS VAL ?

val is every next value of the array just like foreach takes every next value of array



INTERVIEW QUESTIONS

HOW MANY WAYS TO CREATE OBJECT IN JS ?

```
i) var object = new Object();

ii) var object = Object.create(null);

iii) var object = { name: "Sheryians", age: 7 };

iv) function Individual(username) {
          this.username = username;
          this.location = “Bhopal”;
     }
     var object = new Individual("Sheryians");

v) class Individual {
    constructor(username) {
       this.username = username;
    }
   }
   var object = new Individual("Sheryians");
```

INTERVIEW QUESTIONS

ACCESSING OBJECTS PROPERTIES TWO WAYS.

```
var obj = {
    name: “harsh”
}
```

`obj.name`   `obj[‘name’]`

both gives same answer

DELETE OBJECT PROPERTY

```
var obj = {
    name: “harsh”
}
```

`delete obj.name`

deletes the property “name” of object “obj”

INTERVIEW QUESTIONS

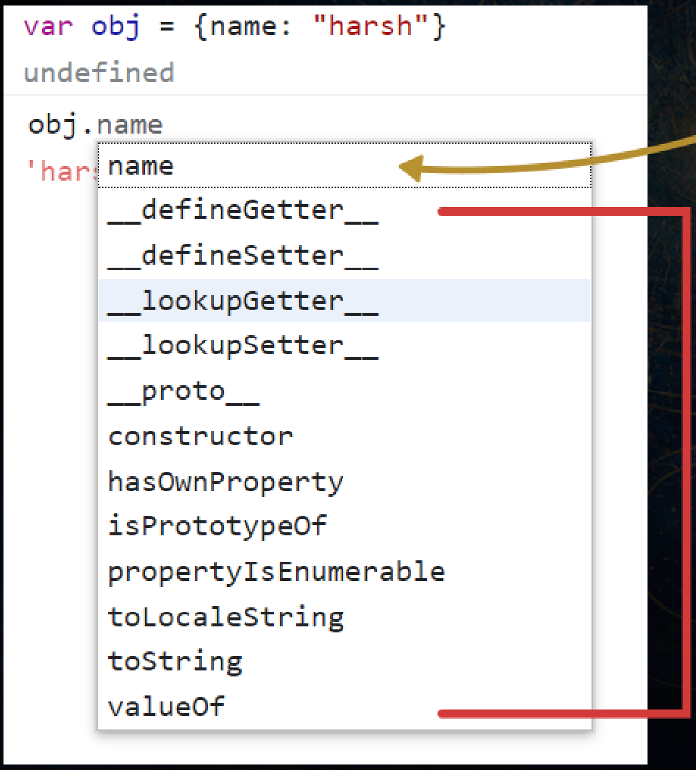
UNDERSTANDING PROTOTYPE.

Go To Browser Console And Create An Object :

```
var obj = {
    name: “Harsh”
}
```

& Now Type Object Name Followed With A Dot Operator :

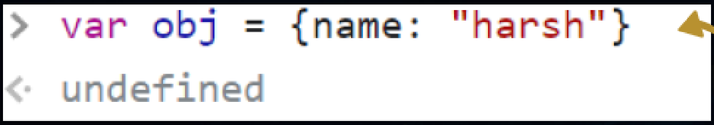
```
obj.
```



```
var obj = {name: "harsh"}
undefined
obj.name
'har
name
__defineGetter__
__defineSetter__
__lookupGetter__
__lookupSetter__
__proto__
constructor
hasOwnProperty
isPrototypeOf
propertyIsEnumerable
toLocaleString
toString
valueOf
```

so, if we didn’t created these properties where do they come from, that’s where the concept of prototype comes in, every created object gets a property called prototype, which means whenever you create an object it gets prototype property automatically

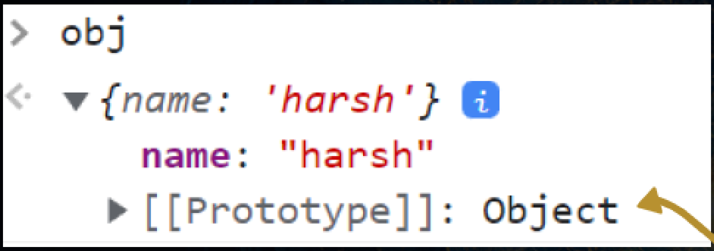
UNDERSTANDING PROTOTYPE.



```
> var obj = {name: "harsh"}
<· undefined
```

we just created obj with name

but when we check it on console what does it contains



```
> obj
<· ▼{name: 'harsh'} i
      name: "harsh"
    ▶ [[Prototype]]: Object
```

we didn’t created [[prototype]]

it contains an extra property called [[prototype]] so where does it come from and what does it contains.

WHERE IT CAME FROM ?

javascript by default adds a property called [[prototype]] to every object, so if you ever see any object, you can blindly say that object contains prototype, so now, what does it contains ?

INTERVIEW QUESTIONS

UNDERSTANDING PROTOTYPE.

WHAT DOES IT CONTAINS ?

[[prototype]] contains many helper properties and methods which we can use to complete our task, let’s say we create an array and we want to know length of it, what do we do, we use .length property on array, did we created .length on that array, no! but it still contains .length, the question is how ?

the answer is, many properties and methods are already available to use built by javascript creators inside prototype of every object.

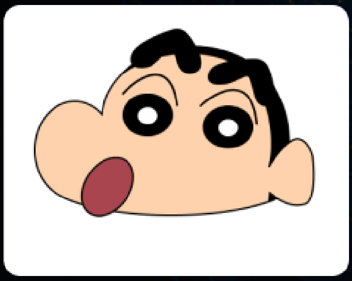
UNDERSTANDING PROTOTYPAL INHERITANCE

that’s shinchan ke papa

he’s human

he got a last name

he got round eyebrows



that’s shinchan

because shinchan is his papa’s son, he inherits or we can say contains properties of his papa, example, shinchan is also human, he also has same last name, and he also gets round eyebrows.

THIS IS CALLED INHERITANCE.

BUT, WHAT ABOUT PROTOTYPAL INHERITANCE ?

that’s exactly what we’re going to talk about now, inheritance is basically passing parent’s features or properties to their childrens, to do the same thing in javascript with the help of prototype (one extra property always given by javascript to every object) is called prototypal inheritance.

UNDERSTANDING PROTOTYPAL INHERITANCE

SO, HOW WE PERFORM PROTOTYPAL INHERITANCE ?

make an object called human and put properties like, canFly, canTalk, willDie

```
var Human = {
    name: “Harsh”,
    canFly: false,
    canTalk: true,
    willDie: true
}
```

make another object called sheryians student, he can do all things which a human can do but he can do few more things like, he can solve js questions and create modern websites, so we create extra two props which normal humans can’t do in that object and rest properties we will inherit from human.

```
var SheryiansStudent = {
    solveJsQuestion: true,
    createModernWebsite: true
}
```

this line does the magic

```
SheryiansStudent.__proto__ = Human;
```

INTERVIEW QUESTIONS

UNDERSTANDING PROTOTYPAL INHERITANCE

```
SheryiansStudent.__proto__ = Human;
```

this line adds all the properties of human in our sheryians students object, so now sheryians student has his properties and it also contains properties of human object, so it inherits properties from parent object Human.

INTERVIEW QUESTIONS

UNDERSTANDING STRICT MODE

to use strict mode, just type “use strict” at top of your code.

IN NORMAL MODE

are bro itna to chalta hai !

party jab khatam ho ghar aajana

baby, I am understanding, you can talk to everyone.

technical examples

```
x = 12;
```
works perfectly.

```
var x = 3.14;
delete x;
```
no errors & doesn’t deletes

```
function fnc(a, a) {};
```
no errors.

IN STRICT MODE

sirf aise hi chalega.

10 se pahle ghar par aajana

why you’re talking to so many girls ?

technical examples

```
x = 12;
```
gives error you should declare it first.

```
var x = 3.14;
delete x;
```
error

```
"use strict";
function fnc(a, a) {};
```
error for same param ‘a’

INTERVIEW QUESTIONS

UNDERSTANDING !! DOUBLE EXCLAMATION

!! does just one thing whatever you write after it will be converted into it’s truthy or falsy state, for example if we write !!-1 it will give us true

!!-1	gives	true
!![1,2,3]	gives	true
!!{name: “harsh”}	gives	true
!!”Hello Sheryians”	gives	true
!!0	gives	false
!!null	gives	false

that means if you write !! infront of anything it will give you either true or false depending on it’s truthy or falsy.

INTERVIEW QUESTIONS

UNDERSTANDING THIS KEYWORD.

this keyword is a special keyword in JavaScript which changes it’s value in different context.

LET’S SEE “THIS” KEYWORD IN DIFFERENT CONTEXT :

in global scope

```
    console.log(this);    gives    window
```

in function scope

```
    function abcd(){
        console.log(this);    gives    window
    }
```

in method scope

```
    var obj = {
       name: “harsh”,
       someMethod: function(){
        console.log(this);         gives    object obj
       }
    }
```

IMPORTANT

in any method, “this” keyword always refers to parent object

INTERVIEW QUESTIONS

UNDERSTANDING THIS KEYWORD.

this keyword is a special keyword in JavaScript which changes it's value in different context.

LET'S SEE "THIS" KEYWORD IN DIFFERENT CONTEXT :

in global scope

```
    console.log(this);    gives    window
```

in function scope

```
    function abcd(){
        console.log(this);    gives    window
    }
```

in method scope

```
    var obj = {
       name: "harsh",
       someMethod: function(){
        console.log(this);          gives    object obj
       }
    }
```

IMPORTANT

in any method, "this" keyword always refers to parent object

INTERVIEW QUESTIONS

UNDERSTANDING THIS KEYWORD.

event listeners

```
var button = document.querySelector("button");

button.addEventListener("click", function(){
    console.log(this);
})
```

this keyword is equal to whatever written before addEventListener, in this case button.

UNDERSTANDING CALL

to change function's this value to some object of our choice we can us call apply & bind.

make a function and check this keyword value :

```
function abcd(){
     console.log(this)
}
```

this = window

but we want to change this keyword inside function from window to some other object,

so we can use call :

```
function abcd(){
     console.log(this)
}

var obj = {
   name: "harsh"
}

abcd.call(obj)
```

when we call function abcd with .call we can pass this keyword's value of our choice.

INTERVIEW QUESTIONS

UNDERSTANDING APPLY

apply also does same thing which call does but if function takes parameter, then apply takes function arguments in an array.

```
function abcd(a,b,c,d){
     console.log(this)
}

var obj = {
   name: “harsh”
}

abcd.apply(obj, [1,2,3,4])
```

apply takes second argument always as array, all value in array are arguments for the parameter of abcd function.

INTERVIEW QUESTIONS

UNDERSTANDING BIND

bind is very similar like call just that it doesn’t calls the function straightaway but returns the function to call it later whenever we want.

```
function abcd(){
     console.log(this)
}

var obj = {
   name: “harsh”
}

var newfnc = abcd.bind(obj)
```

this newfnc variable now contains a function which we can run in future,

INTERVIEW QUESTIONS

UNDERSTANDING PURE FUNCTIONS

Pure function is any function which has these 2 features :

i) it should always return same output for same input

ii) it will never change/update the value of a global variable.

PURE FUNCTION

```
function calc(val){
   return val+2;
}
```

always same answer if you pass same value for ‘val’ argument, hence this function is pure function.

IMPURE FUNCTION

```
let someval = 0;

function calc(x) {
  someval++;
}
```

changes a value of a global variable called someval

INTERVIEW QUESTIONS

UNDERSTANDING LAMBDA FUNCTIONS

Lambda functions are the easy way & shorter way to create functions in js.

OLDER WAY TO CREATE FUNCTIONS.

```
function calc(val){
   return val+2;
}
```

LAMBDA FUNCTIONS.

```
( ) => {

}
```

you put () and then arrow and open {} that’s a lambda function.

save it in a variable.

```
var abcd = ( ) => {

}
```

INTERVIEW QUESTIONS

UNDERSTANDING CURRYING.

if you have a function which takes multiple arguments, we can break down them into a series of function which takes one arguments each.

so, what we do is, we make a function which returns another function, and that returning function uses arguments of parent function too, so on running parent function we get a new function, which on run does some work with both of the arguments.

NORMAL FUNCTION

```
function add(val, val2){
   return val+val2;
}
```

CURRYING EXAMPLE

```
function add(val){
   return function(val2){
       return val+val2;
   }
}
```

```
var fnc2 = add(2)
```

on function call we receive another function

```
var ans = fnc2(5)
```

this gets the ans 7 in variable ans.

INTERVIEW QUESTIONS

UNDERSTANDING TEMPORAL DEAD ZONE

in JavaScript we got new ways to create variables and constants with the help of let and const, and they brought a new concept in picture known as TDZ (Temporal Dead Zone).

in this concept, if you try using a variable created with let or const keyword before declaring it, it will result into error, more specifically reference error.

```
console.log(a);

let a = 12;
```

reference error, using variable before declaring.

NOTE :

in previous days when we used to create variables with var keyword there was no such error because vars don't support TDZ, and we used to get undefined as the answer if we ever asked for a value of a variable before declaring it.

INTERVIEW QUESTIONS

UNDERSTANDING CLOSURES.

Everytime we have a function whcih returns another function, it creates something called closures.

in closures, there’s a parent function which might contain some data/variables which can be accessed/used by the child function present inside it, parent function always return child function in closures.

```
function parent(a){
var someval = a+2;
   return function(b){
        someval++;
   }
}
```

this returning function can access or change parent’s variable someval’s value.

INTERVIEW QUESTIONS

UNDERSTANDING ASYNC VS SYNC.

Synchronous = Line By Line Execution.

Asynchronous = This Code Is Moved To Side Stack, And Its Starts Executing When Whole Synchronous Code Is Executed And Main Stack Is Vacant.

in normal scenarios, code will execute line by line, which means first line executes first and then second line and so on, this way of code execution is called synchronous code execution.

but, js also supports something called asynchronous code execution, which means some code which is asynchronous, will get to side stack for execution and will run after all the synchronous code is finished,

let me explain : think like we have two type of code, sync and async, first, second & third line is sync and fourth and fifth line is async code, now how will they get executed, first, second and third line will move to main stack (main stack gets executed first) and fourth and fifth line will move to side stack, and code on side stack will wait until main stack is empty, then side stack code will move to main stack for execution.

INTERVIEW QUESTIONS

UNDERSTANDING ASYNC VS SYNC.

Synchronous = Line By Line Execution.

Asynchronous = Type Of Code Which Doesn’t Executes Straightaway But Is Moved To Side Stack, And Its Starts Executing When Whole Synchronous Code Is Executed And Main Stack Is Vacant.

in normal scenarios, code will execute line by line, which means first line executes first and then second line and so on, this way of code execution is called synchronous code execution.

but, js also supports something called asynchronous code execution, which means some code which is asynchronous, will get to side stack for execution and will run after all the synchronous code is finished,

let me explain : think like we have two type of code, sync and async, first, second & third line is sync and fourth and fifth line is async code, now how will they get executed, first, second and third line will move to main stack (main stack gets executed first) and fourth and fifth line will move to side stack, and code on side stack will wait until main stack is empty, then side stack code will move to main stack for execution.

UNDERSTANDING LOCAL STORAGE & SESSION STORAGE

local storage, it's like a tiny backpack for websites to store information, for example you want to save score of the game in browser, if you refresh the browser, score remains there.

you're playing this game on browser, and local storage contains data, if you refresh browser, you don't lose your progress

LocalStorage

Score	30
lives	2

Imagine 'sessionStorage' as a temporary notepad for websites during your visit. They write down info on it while you're on their site, but it's tossed out and forgotten when you leave. It's like a short memory for the site, just while you're there.

Save on Mac or iPad for college.
Plus get a gift card up to $150.
Shop now
MacBook Air 15"
Impressively big. Impossibly thin.

you're using this website

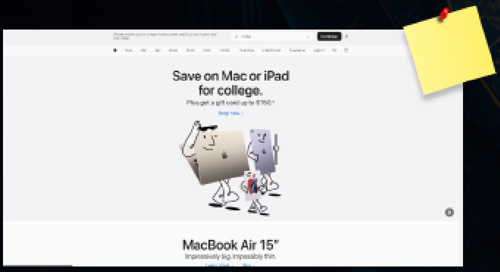
website is saving data and details, related to usage, so that they can remember login info and everything inside session storage.

INTERVIEW QUESTIONS

UNDERSTANDING COOKIE & SESSION

Cookies: Cookies are like small tags that websites attach to your browser. They help websites remember you even when you come back later, they save your location and few more details like what things you checked out on website and other type of data inside the browser and these details are called cookies.

1st time visiting, website save details



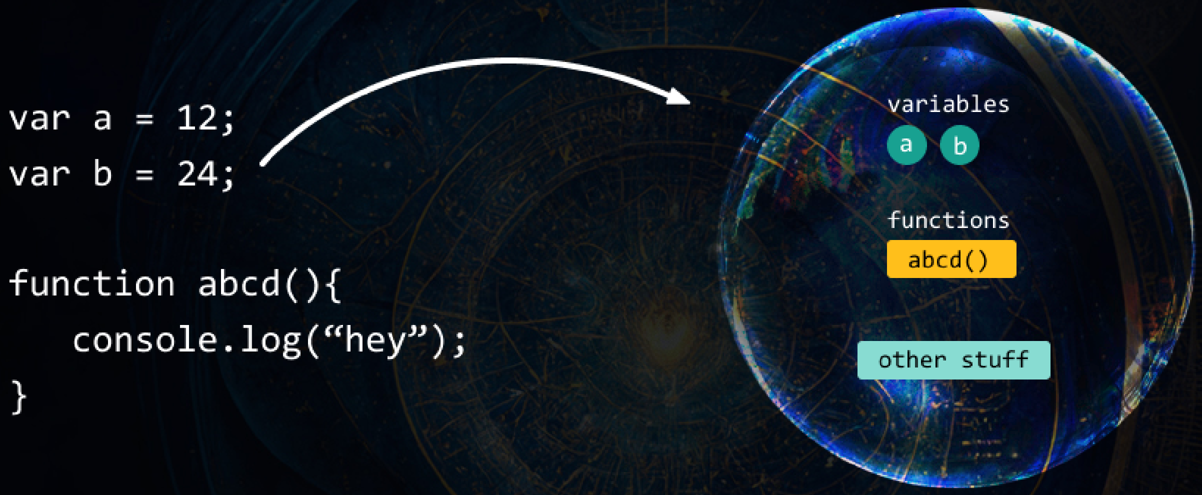
15 days later visiting again, website remembers you because of the previous details

Session: A session is like a special memory a website has only while you're visiting. It forgets everything once you leave, if you don’t specify the expiration time.

INTERVIEW QUESTIONS

UNDERSTANDING LEXICAL ENVIRONMENT

Think of a lexical environment like a little bubble where your code lives. It holds all the things your code needs, like variables and functions, and keeps them organized. When your code runs, it looks inside its own bubble to find what it needs.



```
var a = 12;
var b = 24;

function abcd(){
   console.log(“hey”);
}
```

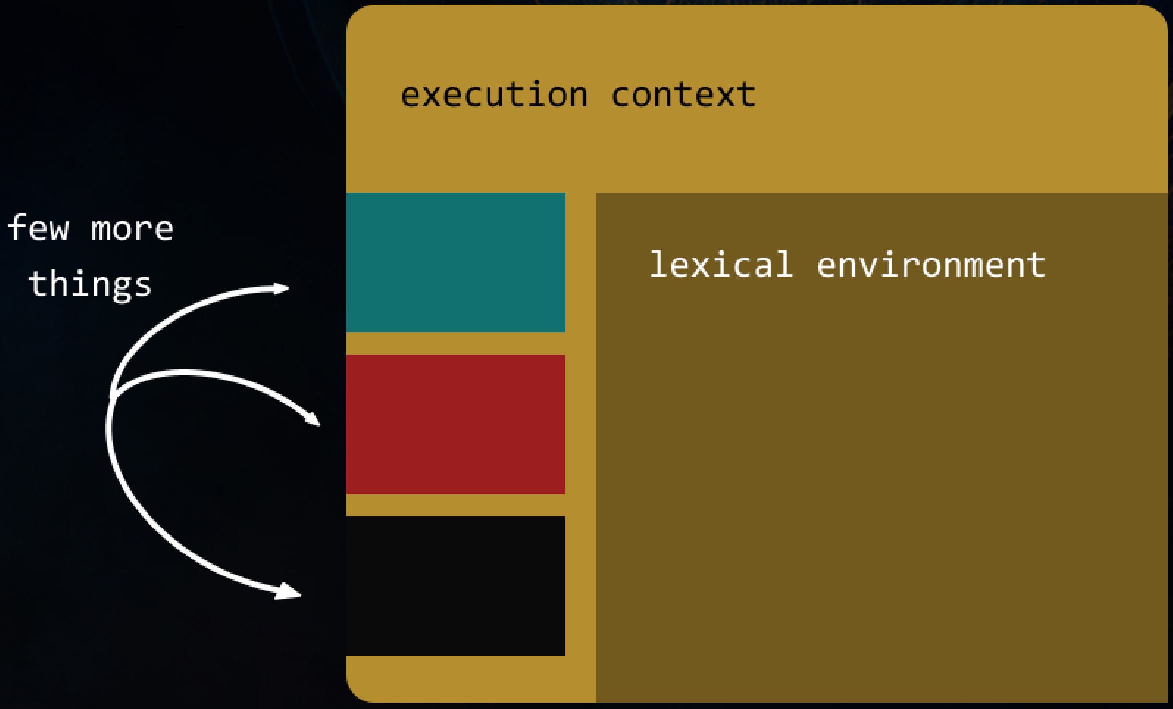
INTERVIEW QUESTIONS

UNDERSTANDING EXECUTION CONTEXT

Each time a function is called, a new execution context is created. It helps manage the scope of variables, keeps track of the call stack, and ensures proper execution of the code. It's the environment in which your code operates and performs its tasks.

it’s like a stage for code to run.

you call a function -> execution context gets created -> it contains lexical environment of the code which means variables, functions and other things related to it to execute that function.



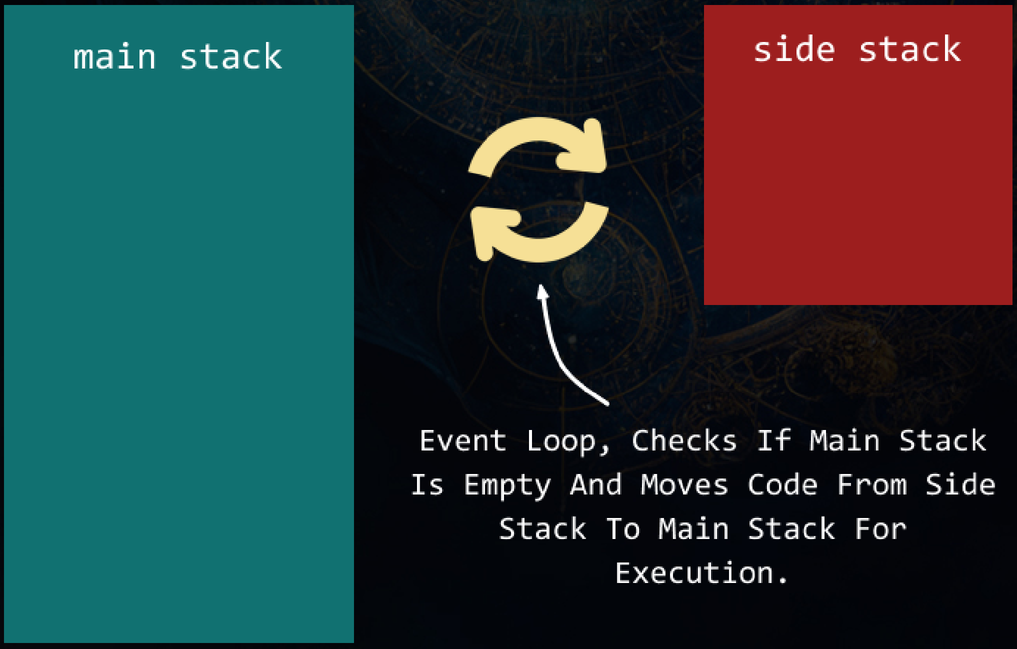
INTERVIEW QUESTIONS

UNDERSTANDING EVENT LOOP

event loop is the on which checks whether the main stack is empty or not and if it empty it takes task from side stack to main stack for execution.

if you don't know, main stack is executed first and contains synchronous code.

other than that, there is one more stack called side stack and it contains async code and waits until the main stack gets empty.


main stack
side stack
Event Loop, Checks If Main Stack Is Empty And Moves Code From Side Stack To Main Stack For Execution.

INTERVIEW QUESTIONS

UNDERSTANDING PROMISES

sometimes code takes time to execute and we never know when it will resolve, like how many seconds, minutes or maybe hours, but we want whenever it finishes, we want to print “done”, but it’s mandatory “done” should only be printed when the code executes, but again the problem is we don’t how much time it will take to execute or finish, in such cases we can use callbacks or promises.

Think of a promise like a special agreement between your code and something that takes time, like this :

“Hey, I promise to let you know when I'm done, whether it's good news or bad news.”