

What is JavaScript?

JS is a programming language. We use it to give instructions to the computer.

Input (code) → Computer → Output

Setting up VS Code </>

It is a free & popular code editor by Microsoft

Apna College

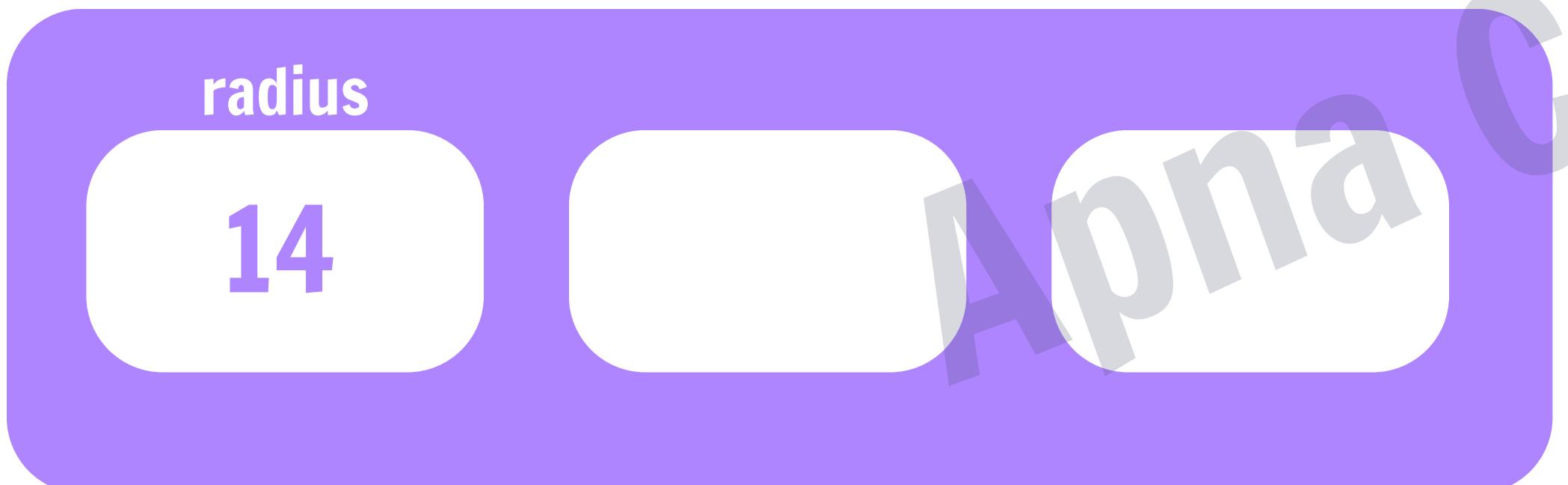
Our 1st JS Code

Console.log is used to log (print) a message to the console

```
console.log("Apna College");
```

Variables in JS

Variables are containers for data



Variable Rules

- Variable names are case sensitive; “a” & “A” is different.
- Only letters, digits, underscore(_) and \$ is allowed. (not even space)
- Only a letter, underscore(_) or \$ should be 1st character.
- Reserved words cannot be variable names.

let, const & var

var : Variable can be re-declared & updated. A global scope variable.

let : Variable cannot be re-declared but can be updated. A block scope variable.

const : Variable cannot be re-declared or updated. A block scope variable.

Data Types in JS

Primitive Types : Number, String, Boolean, Undefined, Null, BigInt, Symbol

Apna College

To **educate** someone is the highest privilege



4.3 Million



510K



...

String

Shradha Khapra



Apna College

Number

Co-founder, Apna College | Ex-Microsoft | Tedx Speaker |
Google SPS'20

441K followers · 500+ connections

Follow



Boolean

LinkedIn

Apna College

Let's Practice

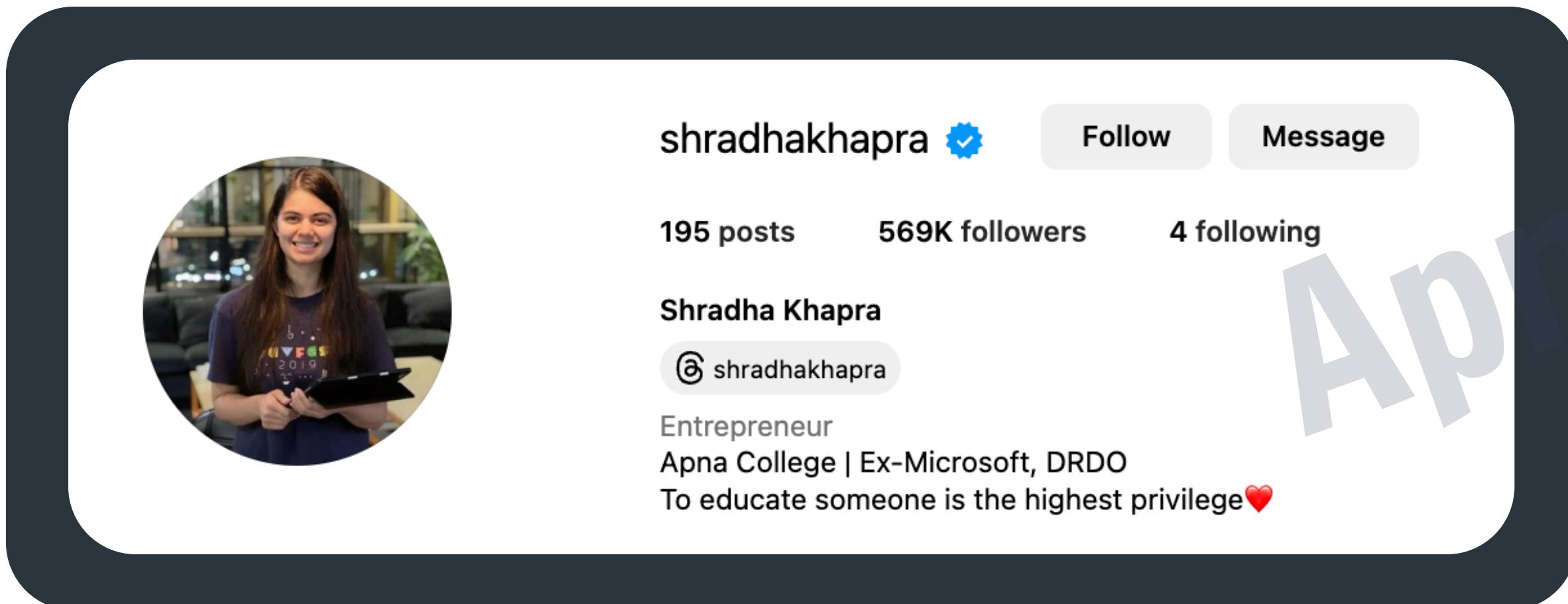
Qs1. Create a const object called “product” to store information shown in the picture.



Apna College

Let's Practice

Qs2. Create a const object called “profile” to store information shown in the picture.



Apna College

Comments in JS

Part of Code which is **not executed**

```
1 //This is a single line comment  
2  
3 /* This is a multi-line  
4 | comment. */
```

Apria college

Operators in JS

Used to perform some **operation** on data

Arithmetic Operators

`+, -, *, /`

- Modulus
- Exponentiation
- Increment
- Decrement

Apna College

Operators in JS

Assignment Operators

= += -= *= %= **=

Apna College

Operators in JS

Comparison Operators

Equal to `==`

Not equal to `!=`

`>, >=, <, <=`

Equal to & type `===`

Not equal to & type `!==`

Apna College

Operators in JS

Logical Operators

Logical AND **&&**

Logical OR **||**

Logical NOT **!**

Apna College

Conditional Statements

To implement some condition in the code

if Statement

```
let color;  
if(mode === "dark-mode") {  
    color = "black";  
}
```

Adena College

Conditional Statements

if-else Statement

```
let color;  
if(mode === "dark-mode") {  
    color = "black";  
} else {  
    color = "white";  
}
```

Aruna College

Conditional Statements

else-if Statement

```
if(age < 18) {  
    console.log("junior");  
} else if (age > 60) {  
    console.log("senior");  
} else {  
    console.log("middle");  
}
```

Apna College

Operators in JS

Ternary Operators

condition ? true output : false output

```
age > 18 ? "adult" : "not adult";
```

MDN Docs

Apna College

Let's Practice

Qs1. Get user to input a number using prompt("Enter a number:"). Check if the number is a multiple of 5 or not.

Apna College

Let's Practice

Qs2. Write a code which can give grades to students according to their scores:

- **80-100, A**
- **70-89, B**
- **60-69, C**
- **50-59, D**
- **0-49, F**

Apna College

Loops in JS

Loops are used to execute a piece of code again & again

for Loop

```
for (let i = 1; i <= 5; i++) {  
    console.log("apna college");  
}
```

Loops in JS

Infinite Loop : A Loop that never ends

Apna College

Loops in JS

while Loop

```
while (condition) {
```

```
    // do some work
```

```
}
```

Apna College

Loops in JS

do-while Loop

```
do {  
    // do some work  
} while (condition);
```

Loops in JS

for-of Loop

```
for (let val of strVar) {
```

```
    //do some work
```

```
}
```

Loops in JS

for-in Loop

```
for (let key in objVar) {
```

```
    //do some work
```

```
}
```

Let's Practice

Qs1. Print all even numbers from 0 to 100.

Apna College

Let's Practice

Qs2.

Create a game where you start with any random game number. Ask the user to keep guessing the game number until the user enters correct value.

Strings in JS

String is a sequence of characters used to represent text

Create String

```
let str = "Apna College";
```

String Length

```
str.length
```

String Indices

```
str[0], str[1], str[2]
```

Template Literals in JS

A way to have embedded expressions in strings

`this is a template literal`

String Interpolation

To create strings by doing substitution of placeholders

`string text \${expression} string text`

String Methods in JS

These are built-in functions to manipulate a string

- `str.toUpperCase()`
- `str.toLowerCase()`
- `str.trim()` *// removes whitespaces*

Apna College

String Methods in JS

- **str.slice(start, end?)** *// returns part of string*
- **str1.concat(str2)** *// joins str2 with str1*
- **str.replace(searchVal, newVal)**
- **str.charAt(idx)**

Let's Practice

**Qs1. Prompt the user to enter their full name. Generate a username for them based on the input.
Start username with @, followed by their full name and ending with the fullname length.**

eg: user name = “shradhakhapra” , username should be “@shradhakhapra13”

Arrays in JS

Collections of items

Create Array

```
let heroes = [ "ironman", "hulk", "thor", "batman" ];
```

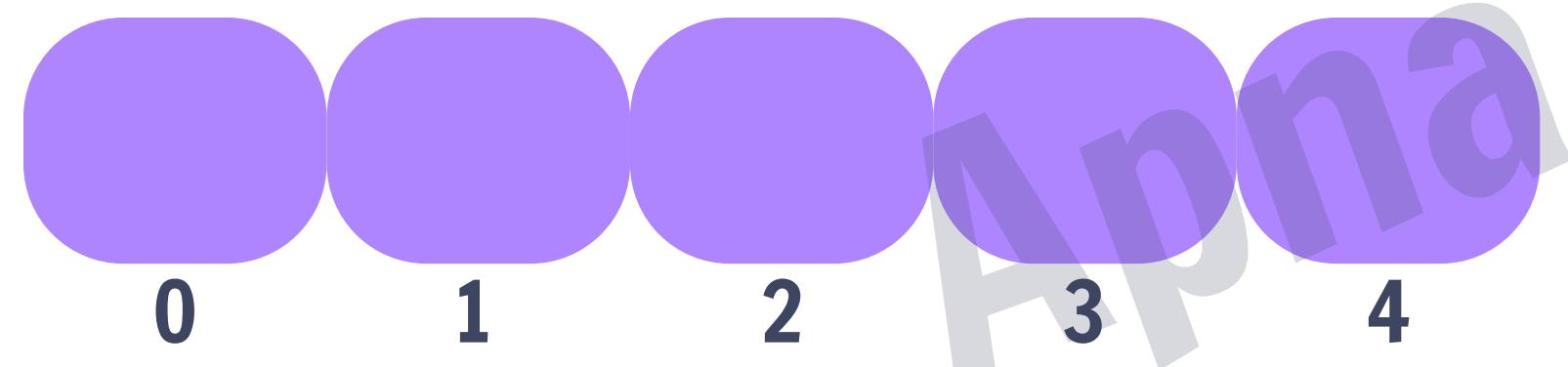
```
let marks = [ 96, 75, 48, 83, 66 ];
```

```
let info = [ "rahul", 86, "Delhi" ];
```

Arrays in JS

Array Indices

arr[0], arr[1], arr[2]



Looping over an Array

Print all elements of an array

Apna College

Let's Practice

Qs. For a given array with marks of students -> [85, 97, 44, 37, 76, 60]

Find the average marks of the entire class.

Apna College

Let's Practice

Qs. For a given array with prices of 5 items -> [250, 645, 300, 900, 50]

All items have an offer of 10% OFF on them. Change the array to store final price after applying offer.

Apna College

Arrays in JS

Array Methods

Push() : add to end

Pop() : delete from end & return

toString() : converts array to string

Apna College

Arrays in JS

Array Methods

Concat() : joins multiple arrays & returns result

Unshift() : add to start

shift() : delete from start & return

Apna College

Arrays in JS

Array Methods

Slice() : returns a piece of the array

`slice(startIdx, endIdx)`

Splice() : change original array (add, remove, replace)

`splice(startIdx, delCount, newEl...)`

Let's Practice

Qs. Create an array to store companies -> “Bloomberg”, “Microsoft”, “Uber”, “Google”, “IBM”, “Netflix”

- a. Remove the first company from the array
- b. Remove Uber & Add Ola in its place
- c. Add Amazon at the end

Functions in JS

Block of code that performs a specific task, can be invoked whenever needed

Apna College

Functions in JS

Function Definition

```
function functionName() {  
    //do some work  
}
```

Function Call

```
functionName();
```

```
function functionName( param1, param2 ...) {  
    //do some work  
}
```

Apna College

Arrow Functions

Compact way of writing a function

```
const functionName = ( param1, param2 ... ) => {  
    //do some work  
}
```

```
const sum = ( a, b ) => {  
    return a + b;  
}
```

Let's Practice

Qs. Create a function using the “function” keyword that takes a String as an argument & returns the number of vowels in the string.

Qs. Create an arrow function to perform the same task.

Apna college

forEach Loop in Arrays

arr.**forEach**(callBackFunction)

CallbackFunction : Here, it is a function to execute for each element in the array

*A callback is a function passed as an argument to another function.

```
arr.forEach( ( val ) => {  
    console.log(val);  
})
```

Let's Practice

Qs. For a given array of numbers, print the square of each value using the `forEach` loop.

Apna College

Some More Array Methods

Map

Creates a new array with the results of some operation. The value its callback returns are used to form new array

```
arr.map( callbackFnx( value, index, array ) )
```

```
let newArr = arr.map( ( val ) => {  
    return val * 2;  
})
```

Some More Array Methods

Filter

Creates a new array of elements that give true for a condition/filter.

Eg: all even elements

```
let newArr = arr.filter( ( val ) => {  
    return val % 2 === 0;  
})
```

Some More Array Methods

Reduce

Performs some operations & reduces the array to a single value. It returns that single value.

JavaScript Demo: Array.reduce()

```
1 const array1 = [1, 2, 3, 4];
2
3 // 0 + 1 + 2 + 3 + 4
4 const initialValue = 0;
5 const sumWithInitial = array1.reduce(
6   (accumulator, currentValue) => accumulator + currentValue,
7   initialValue,
8 );
9
10 console.log(sumWithInitial);
11 // Expected output: 10
```

Let's Practice

Qs. We are given array of marks of students. Filter our of the marks of students that scored 90+.

Qs. Take a number n as input from user. Create an array of numbers from 1 to n.

Use the reduce method to calculate sum of all numbers in the array.

Use the reduce method to calculate product of all numbers in the array.

The 3 Musketeers of Web Dev

HTML
(structure)



CSS
(style)



JS
(logic)



Apna college

Starter Code

<style> tag connects HTML with CSS

<script> tag connects HTML with JS

Apna College

```
<html>  
  <head>  
    <title> Website Name </title>  
  </head>  
  <body>  
    <!-- Content Tags -->  
  </body>  
</html>
```

Apna College

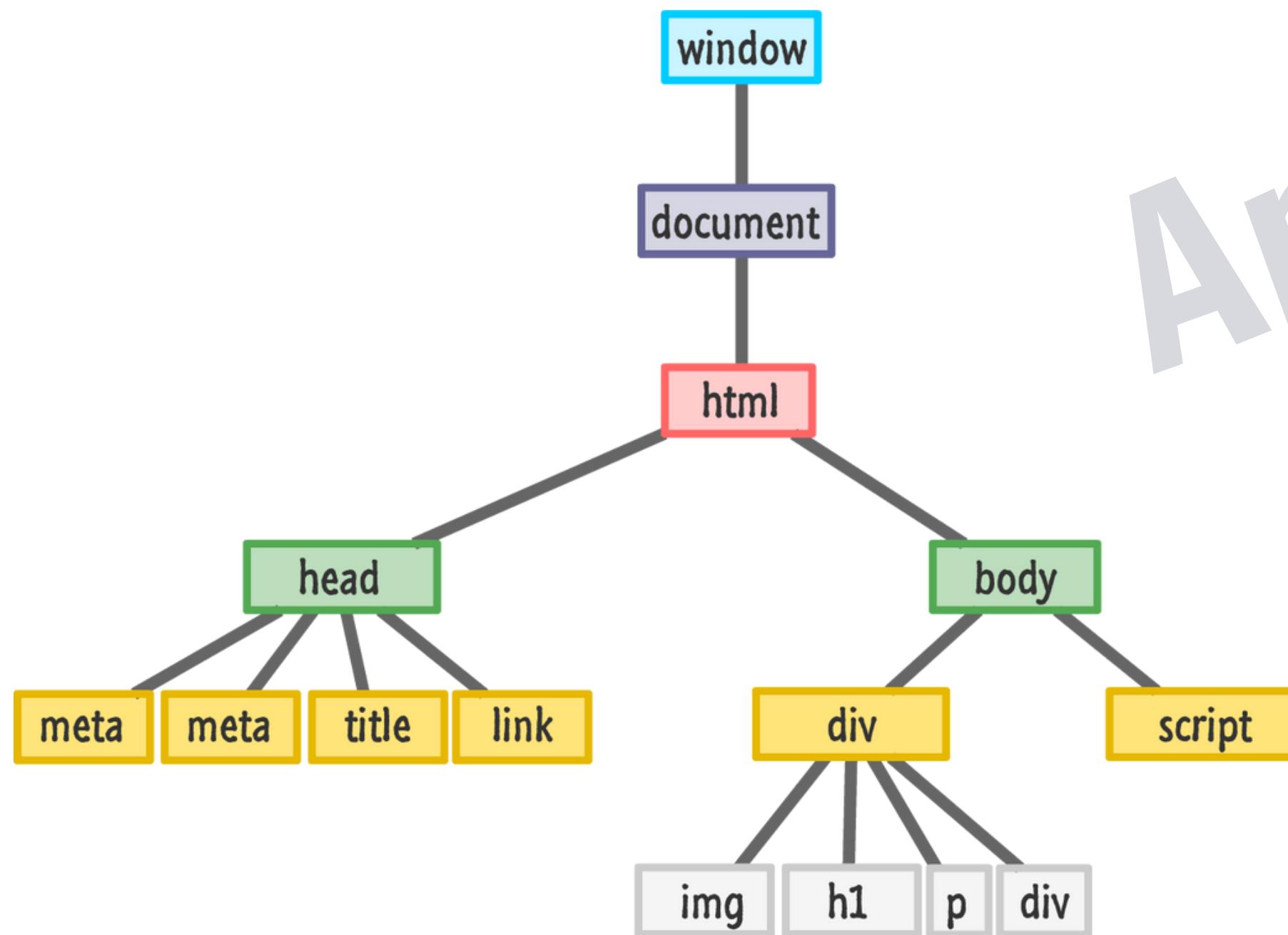
Window Object

The **window** object represents an open window in a browser. It is browser's object (not JavaScript's) & is automatically created by browser.

It is a **global** object with lots of properties & methods.

What is DOM?

When a web page is loaded, the browser creates a **Document Object Model (DOM)** of the page



DOM Manipulation

Selecting with id

document.getElementById("myId")

Selecting with class

document.getElementsByClassName("myClass")

Selecting with tag

document.getElementsByTagName("p")

Apna College

DOM Manipulation

Query Selector

```
document.querySelector("#myId / .myClass / tag")
```

//returns first element

```
document.querySelectorAll("#myId / .myClass / tag")
```

//returns a NodeList

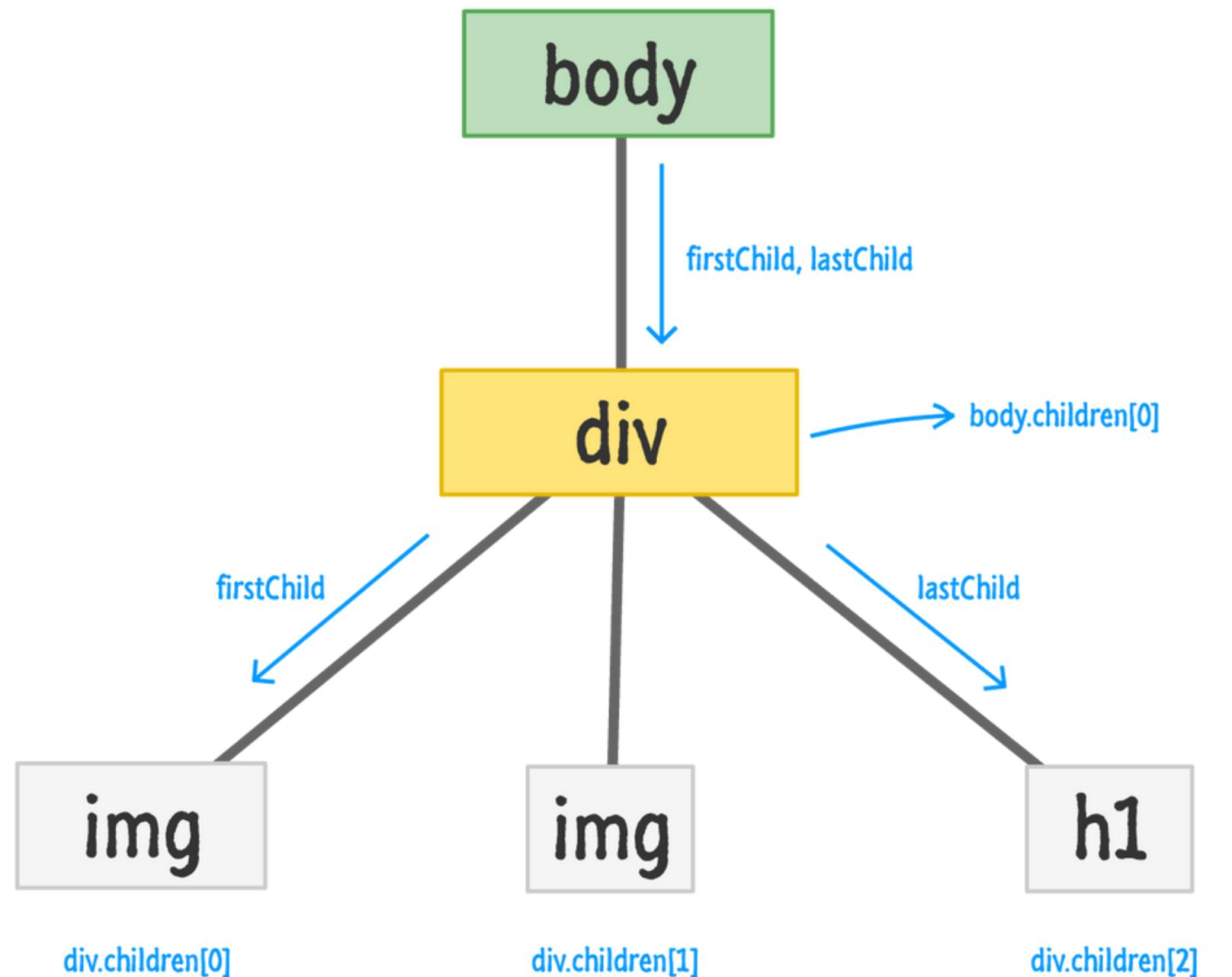
Apna College

DOM Manipulation

Properties

- **tagName** : returns tag for element nodes
- **innerText** : returns the text content of the element and all its children
- **innerHTML** : returns the plain text or HTML contents in the element
- **textContent** : returns textual content even for hidden elements

Homework



Let's Practice

Qs. Create a H2 heading element with text - “Hello JavaScript”. Append “from Apna College students” to this text using JS.

Qs. Create 3 divs with common class name - “box”. Access them & add some unique text to each of them.

DOM Manipulation

Attributes

- `getAttribute(attr)` //to get the attribute value
- `setAttribute(attr, value)` //to set the attribute value

Style

- `node.style`

Apna College

DOM Manipulation

Insert Elements

```
let el = document.createElement("div")
```

- `node.append(el)` //adds at the end of node (inside)
- `node.prepend(el)` //adds at the start of node (inside)
- `node.before(el)` //adds before the node (outside)
- `node.after(el)` //adds after the node (outside)

Delete Element

- `node.remove()` //removes the node

Let's Practice

Qs. Create a new button element. Give it a text “click me”, background color of red & text color of white.

Insert the button as the first element inside the body tag.

Qs. Create a `<p>` tag in html, give it a class & some styling.

Now create a new class in CSS and try to append this class to the `<p>` element.

Did you notice, how you overwrite the class name when you add a new one?
Solve this problem using `classList`.

Events in JS

The change in the state of an object is known as an Event

Events are fired to notify code of "interesting changes" that may affect code execution.

- Mouse events (click, double click etc.)
- Keyboard events (keypress, keyup, keydown)
- Form events (submit etc.)
- Print event & many more

Event Handling in JS

```
node.event = () => {  
    //handle here  
}
```

example

```
btn.onclick = () => {  
    console.log("btn was clicked");  
}
```

Event Object

It is a special object that has details about the event.

All event handlers have access to the Event Object's properties and methods.

```
node.event = (e) => {  
    //handle here  
}
```

e.target, e.type, e.clientX, e.clientY

Event Listeners

`node.addEventListener(event, callback)`

`node.removeEventListener(event, callback)`

***Note : the callback reference should be same to remove**

Let's Practice

Qs. Create a toggle button that changes the screen to dark-mode when clicked & light-mode when clicked again.

Apna College

Classes & Objects

Apna College

Prototypes in JS

A JavaScript object is an entity having state and behavior (properties and method).

JS objects have a special property called prototype.

We can set prototype using `__proto__`

*If object & prototype have same method,
object's method will be used.

Classes in JS

Class is a program-code template for creating objects.

Those objects will have some state (variables) & some behaviour (functions) inside it.

```
class MyClass {  
    constructor( ) { ... }  
  
    myMethod( ) { ... }  
}
```

```
let myObj = new MyClass( );
```

Classes in JS

Constructor() method is :

- automatically invoked by new
- initializes object

```
class MyClass {  
    constructor() { ... }  
    myMethod() { ... }  
}
```

Inheritance in JS

inheritance is passing down properties & methods from parent class to child class.

```
class Parent {  
}
```

```
class Child extends Parent {  
}
```

*If Child & Parent have same method, child's
method will be used. [Method Overriding]

super Keyword

The super keyword is used to call the constructor of its parent class to access the parent's properties and methods.

super(args) // calls Parent's constructor

super.parentMethod(args)

Let's Practice

Qs. You are creating a website for your college. Create a class User with 2 properties, name & email. It also has a method called `viewData()` that allows user to view website data.

Qs. Create a new class called Admin which inherits from User. Add a new method called `editData` to Admin that allows it to edit website data.

Error Handling

try-catch

```
try {  
    ... normal code  
}  
  catch ( err ) { //err is error object  
    ... handling error  
}
```

What this chapter is about?

async await >> promise chains >> callback hell

Apna College

Sync in JS

Synchronous

Synchronous means the code runs in a particular sequence of instructions given in the program. Each instruction waits for the previous instruction to complete its execution.

Asynchronous

Due to synchronous programming, sometimes imp instructions get blocked due to some previous instructions, which causes a delay in the UI. Asynchronous code execution allows to execute next instructions immediately and doesn't block the flow.

Callbacks

A callback is a function passed as an argument to another function.

Apna College

Callback Hell

Callback Hell : Nested callbacks stacked below one another forming a pyramid structure.
(Pyramid of Doom)

This style of programming becomes difficult to understand & manage.

Promises

Promise is for “eventual” completion of task. It is an object in JS.

It is a solution to callback hell.

```
let promise = new Promise( (resolve, reject) => { .... } )
```

Function with 2 handlers

***resolve & reject are callbacks provided by JS**

Promises

A JavaScript Promise object can be:

- Pending : the result is undefined
- Resolved : the result is a value (fulfilled)
- Rejected : the result is an error object

`resolve(result)`

`reject(error)`

*Promise has state (pending, fulfilled) & some result (result for resolve & error for reject).

Promises

.then() & .catch()

promise.then((res) => { })

promise.catch((err))=>{ })

Apna College

Async-Await

async function always returns a promise.

async function myFunc() { }

await pauses the execution of its surrounding async function until the promise is settled.

IIFE : Immediately Invoked Function Expression

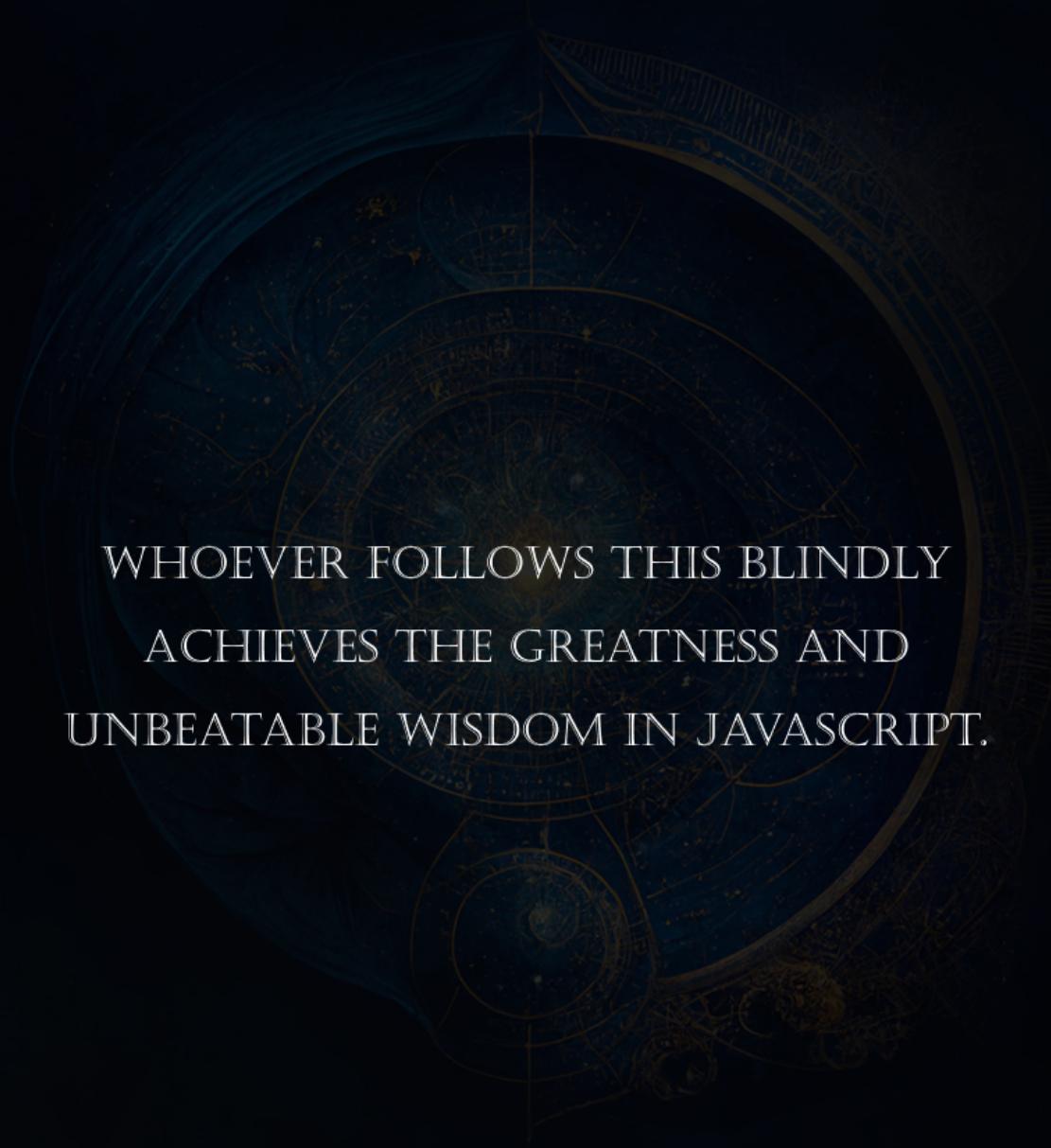
IIFE is a function that is called immediately as soon as it is defined.

```
(function () {  
  // ...  
})();  
  
(() => {  
  // ...  
})();  
  
(async () => {  
  // ...  
})();
```

S H E R Y I A N S C O D I N G S C H O O L
P R E S E N T S



Bible of
JavaScript



WHOEVER FOLLOWS THIS BLINDLY
ACHIEVES THE GREATNESS AND
UNBEATABLE WISDOM IN JAVASCRIPT.

Interview Bible

Var Let And Const

== And ===

First Class Functions

Constructor Functions

New Keyword

iife

Map Filter, Reduce, Sort

Object In Js

Accessing Objects Properties Two Ways

Prototype & Prototypal Inheritance

Strict Mode JS

!! In Js

This Keyword

Call, Apply, Bind

Pure And Impure Functions

Lambda Functions

Currying

Temporal Dead Zone

Interview Bible

Closures

Sync Vs Async JS

LocalStorage Vs SessionStorage

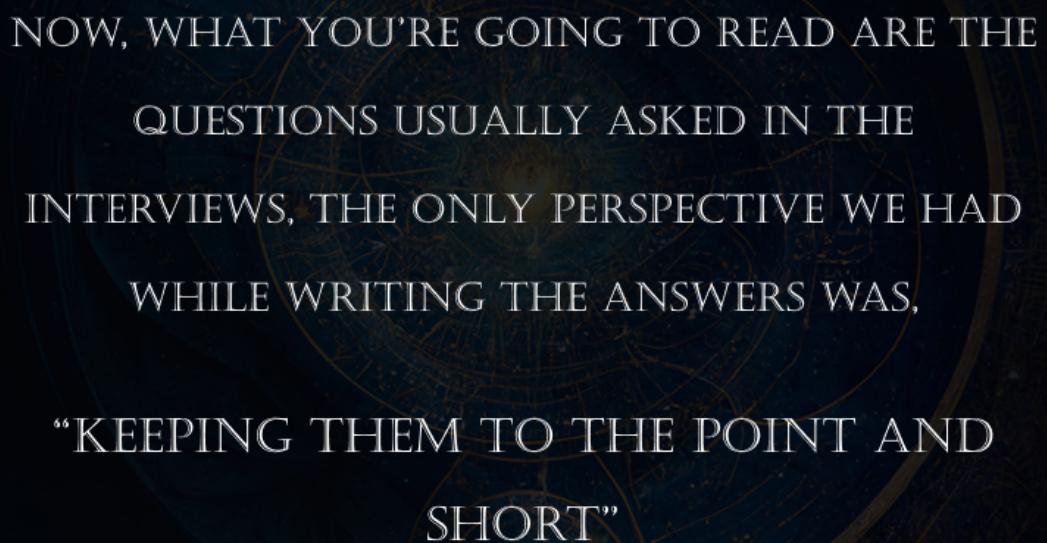
Cookie & Session

Lexical Environment

Execution Context

Event Loop

Promises



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 & const 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 accessed 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";
 }

var person1 = new abcd();

constructor function



**new keyword in front 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( );
```

① new keyword make a blank {}

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

```
{  
    name: "harsh"  
}
```

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

INTERVIEW QUESTIONS

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 Highlights

- 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 Highlights

- 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

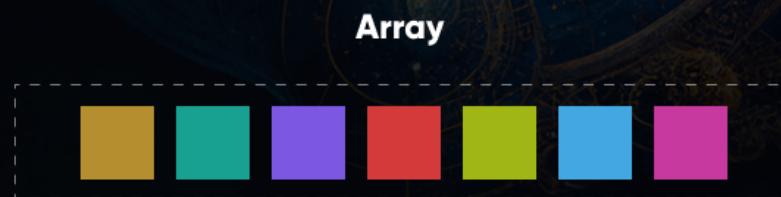
EXPLAIN MAP, FILTER REDUCE.

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



Loop Runs For

1st Time

Acc =

2nd Time

Acc =

3rd Time

Acc =

Last Time

Acc =

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.
```

The screenshot shows a browser console with the following text:

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

Annotations on the right side of the screenshot explain the properties:

- A yellow arrow points from the text "we created name" to the "name" property in the list.
- A red box encloses the properties: __defineGetter__, __defineSetter__, __lookupGetter__, __lookupSetter__, __proto__, constructor, hasOwnProperty, isPrototypeOf, propertyIsEnumerable, toLocaleString, toString, and valueOf. A red arrow points from the text "but we didn't created these" to this box.

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

INTERVIEW QUESTIONS

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'} ⓘ  
  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.

INTERVIEW QUESTIONS

UNDERSTANDING PROTOTYPAL INHERITANCE

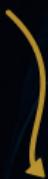


that's shinchan ke papa

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.

INTERVIEW QUESTIONS

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 !! in front 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.

INTERVIEW QUESTIONS

UNDERSTANDING CALL

to change function's this value to some object of our choice we can use 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); ↗ reference error, using  
let a = 12;           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.

INTERVIEW QUESTIONS

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.



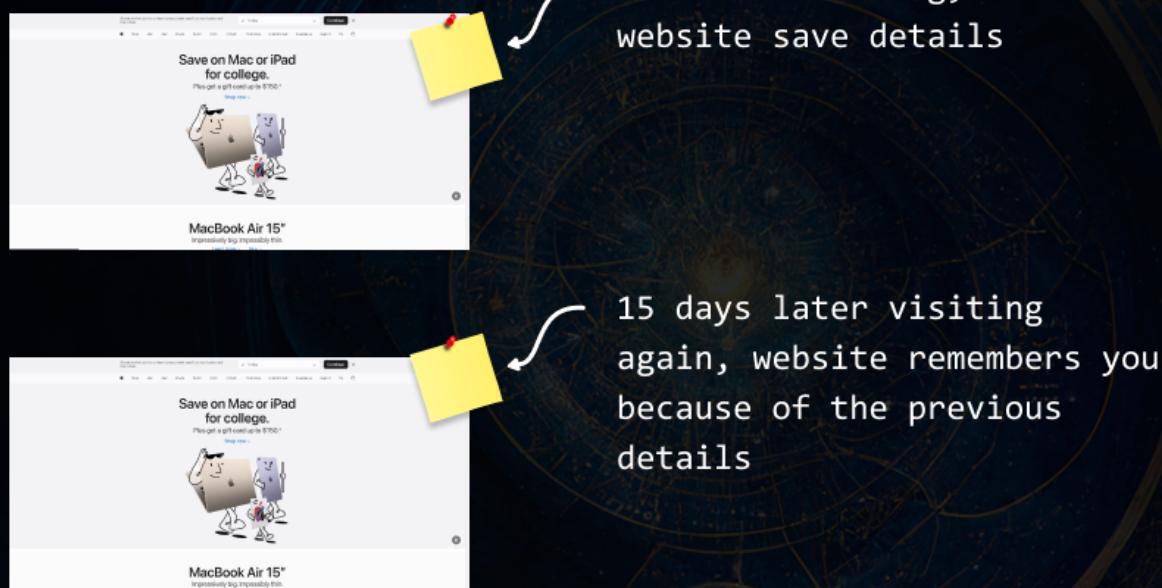
you're using this website

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.



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");  
}  
}
```



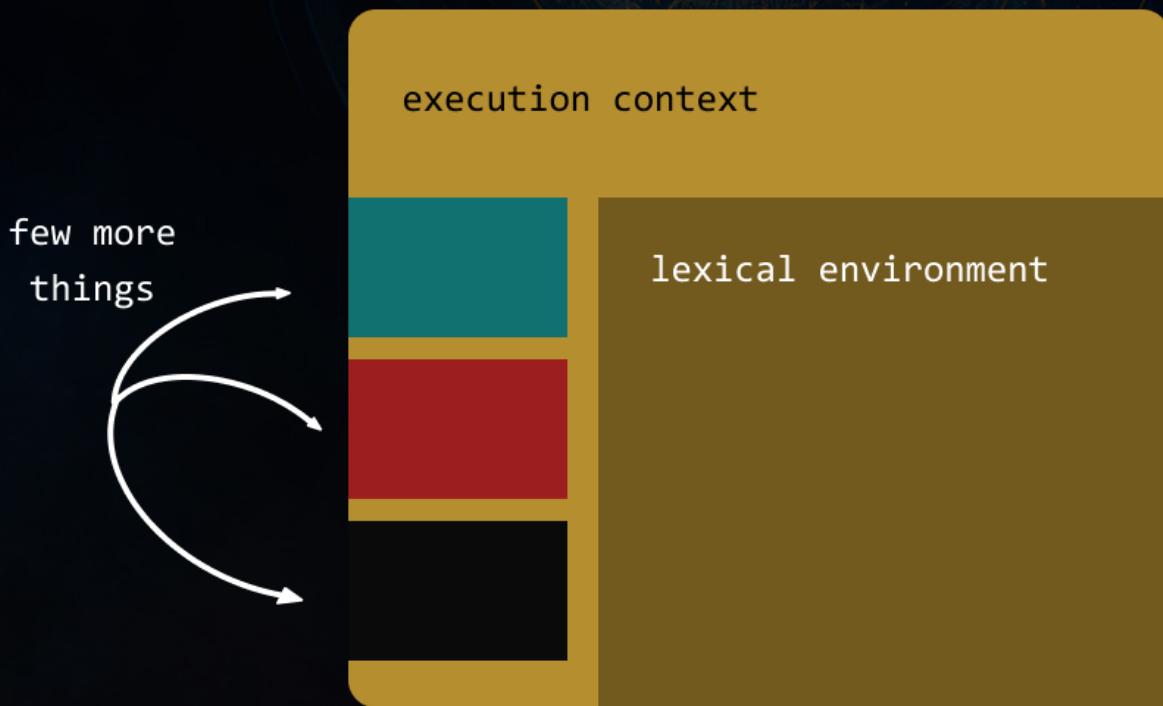
INTERVIEW QUESTIONS

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.



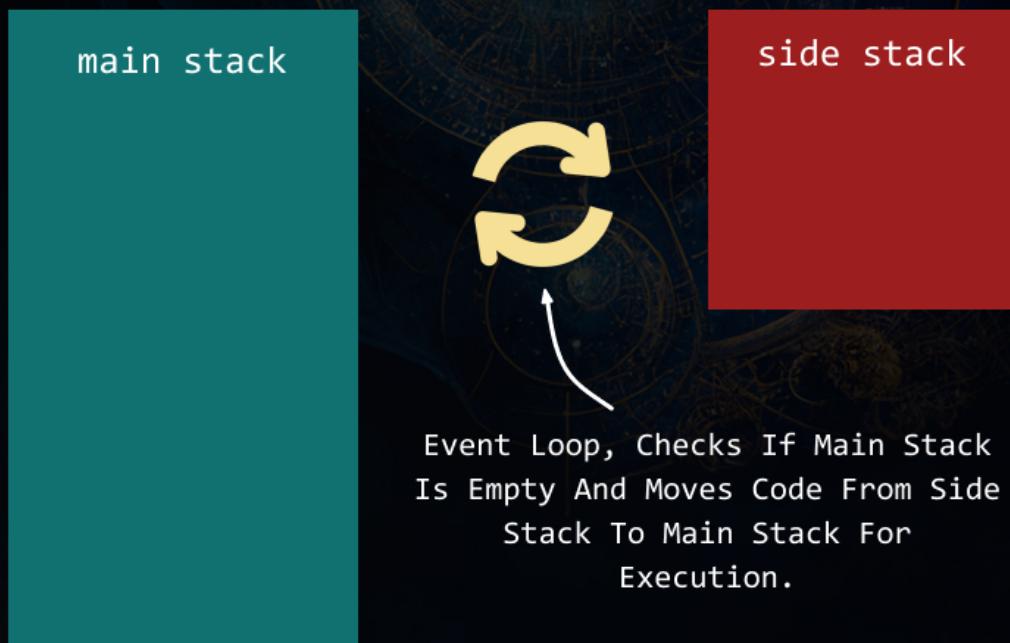
INTERVIEW QUESTIONS

UNDERSTANDING EVENT LOOP

event loop is the one 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.



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 know 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.”