Q 1. What is JavaScript?

JavaScript is the most popular language nowadays it is a client-side scripting language as well as a server-side programming language.

JavaScript is a synchronous single-thread programming language?

- Synchronous:- One command at a time.
- Single-threaded:- In a specific synchronous order.

With the help of JavaScript, we can calculate, manipulate
and validate your data

by using JavaScript we can change the dynamics of content on the page

Q 2. Where can we used JavaScript ? or why to learn JavaScript ?

1: Client side scripting language
 JavaScript , jQuery , Angular , react ,

2: Server side programming language

 Node JS , Express JS

it is used in mobile applications gaming applications, and desktop applications. browser plugin, extension, IoT , Machine learning

Q.3 How JavaScript Work ?

Everything in JavaScript happens inside the execution context

the whole thing execution context inside 2 component

1: Memory component :

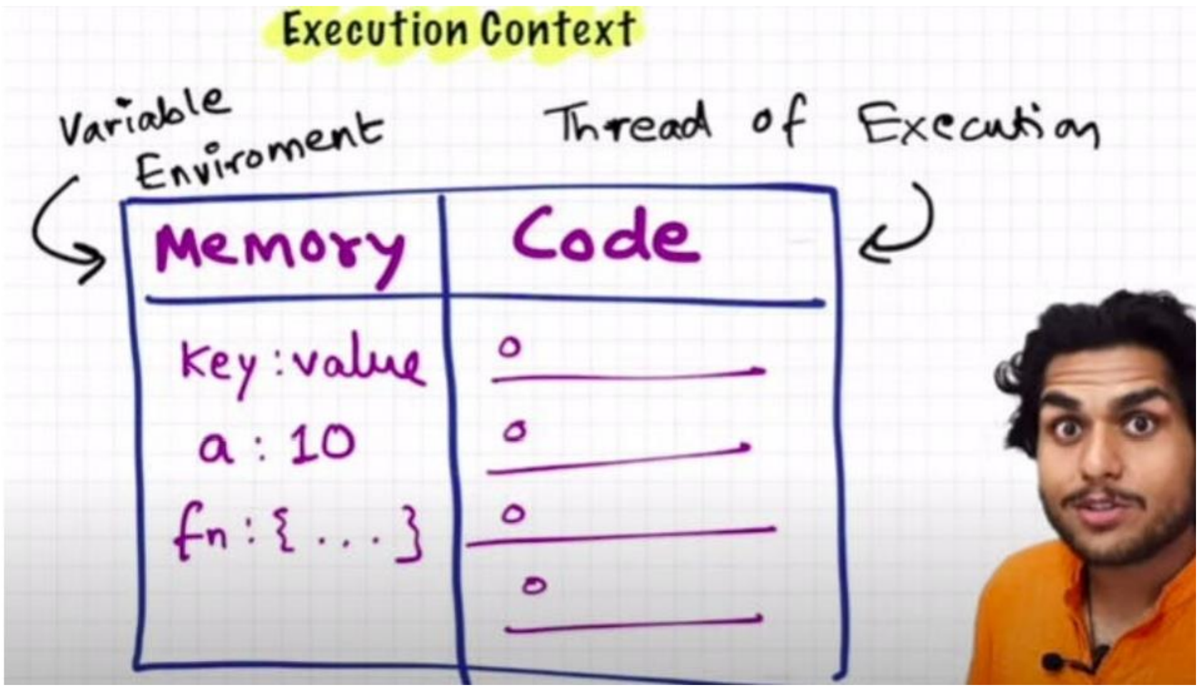
- In the memory component, variable and function values can be stored in a key value pair
- Memory components are also known as variable environment

2: Code components :

- The code component is the place where the whole JavaScript code is executed
- the code component is also known as a thread of execution

JavaScript is a synchronous single-thread programming language?

- Synchronous:- One command at a time.
- Single-threaded:- In a specific synchronous order.



Q.4 What is == and === operators ?
"==" is used to compare values whereas, " === " is used to compare both values and types.

Q.5 What is datatypes in JavaScript ?

- Primitive types: string, Boolean, number, undefined, null, BigInt.
- Non-primitive types: Object.

Q.6 How to declare variable in JS ?

- Using let
- Using const
- Using nothing
- Using var

Example :

Let number = 10;

Var number = 10;

Const number = 10;

Q.7 What is hoisting In JS with example ? or what is function hoisting and variable hoisting ?

Hoisting is a JavaScript mechanism where variables, function declarations, and classes are moved to the top of their scope before code execution.

Remember that JavaScript only hoists declarations, not initialization.

Or

Hoisting is a phenomenon in JavaScript by which can access the function and variable even before initializing it

we can put some value on it, we can access it before any error.

```
getName();
console.log(x);
var x = 10;

function getName() {
  console.log("JavaScript");
}
```

Output :

JavaScript

Undefined

Q.8 What is a function of JS?

It is a block of the statement which takes the set of parameters and returns its result is called a function

Why used : code reusability

```
function printNumber(){
let result = "";
for(let i=0; i<=10; i++){
    result += `${i}`;
}
console.log(result);
}
printNumber();
```

output :

 0123456789

0123456789

Q.9 What is undefined, not defined, and null?

undefined:

- in the first phase (memory allocation) is assigned to each variable a placeholder called undefined
- when the memory is allocated for memory is allocated variable but not assigned any value is called undefined

Not defined :

an object and variable are not even declared not found in the memory phase and tries to access then it called not defined

Null :

Null is value that can be assigned to the variable or and object.

Q.10 What is a closure?

closure is a combination of function and the lexical environment within which that function was declared.

the inner function that accesses the outer enclosing function variable.

the closure has 3 scope changes:

- 1: own scope where variable defines its curly bracket
- 2: outer function variable
- 3: global variable

Advantages of closure?

module design pattern ,currying ,memorization, data hiding ,encapsulation,setTimeout. etc

disadvantages of closure :

over consumption memory, memory leak

```
function outer(){
   var a = 10;
   function inner(){
      console.log(a);
   }
   return inner();
}
outer();
```

output : 10

Q.11 What is a call back function?

the call back function is a function passed into another function as an argument this function is inside the outer function to complete an action

```
function callbackFunction(name) {
  console.log("hello + name");
}
function outerFunction(callback) {
  let name = prompt("enter your name");
  callback(name);
}
outerFunction(callbackFunction);
```

Q.12 What is block ?

Block is scope means all the function and variable access inside the {} is called block scope

Q.13 What is a promise?

a promise is an object that may be produced single value

sometimes in the future with either the promise is resolved or not resolved

i.e. Network error

there are three possible state

- 1: Fulfilled
- 2: Resolved
- 3: Pending

Syntax

```
let promise = new Promise(function(resolve, reject) {
  // executor (the producing code, "singer")

});
```

Example :

```
let cleanCamera = new promise ((resolve, reject) =>{
    let isDone = true;
    if(isDone){
        resolve("cleaning is done");
    }
    else{
        reject("cleaning is not done");
    }
});
```

```
cleanCamera.then((message)=> {
   console.log(message);
}.(error) => {
  console.error(error);
});
```

Q. 14 defines const, let, and var in JS

let and const are block scope, there is a separate memory space

let and const are hoisted undefined

there are separate memory spaces which resolved for this block

let and const we can not access outside the scope {} that is also known as block scope

const: in const when we put some value we can change it later in const

let: let is temporal dead zone along with the unexpected error, undefine and all the thing's

var: var keyword can access outside the block scope because is it global scope

var are introduce in ES5

let and const are introduce in ES6

Q.15 What is the arrow function?

in arrow function It provides a more concise syntax for writing function expressions by removing the "function" and "return" keywords.

Arrow functions are defined using the fat arrow (=>) notation.

It is evident that there is no "return" or "function" keyword in the arrow function declaration.

We can also skip the parenthesis in the case when there is exactly one parameter, but will always need to use it when you have zero or more than one parameter.

But, if there are multiple expressions in the function body, then we need to wrap it with curly braces ("{}"). We also need to use the "return" statement to return the required value.

```
// ES5
var sum = function () {
  var a = 5;
  var b = 6;
  return a + b;
};
console.log(sum());

// ES6
const sum = (a, b) => a + b;
console.log(sum(25, 30));
```

Q.16 What is higher order function ?

a function that takes another function as an argument or returns a function from it is known as higher order function

```
function x(){
  console.log("JavaScript");
}
function y(x){
  x();
}
```

Q.17 Explain the terms : map ,filter , reduce ?

map: The map() method is used for creating a new array from an existing one, applying a function to each one of the elements of the first array.

Syntax :

```
var new_array = arr.map(function callback(element, index, array) {
    // Return value for new_array
}[, thisArg])
```

Example

```
const arr = [5, 4, 5, 3, 5, 2, 6];

function double(x) {
  return x * 2;
}
const output = arr.map(double);
console.log(output);
```

filter: filter function is used to filter the values from array.

The filter() method takes each element in an array and it applies a conditional statement against it. If this conditional returns true, the element gets pushed to the output array. If the condition returns false, the element does not get pushed to the output array.

Syntax :

```
var new_array = arr.filter(function  callback(element, index, array) {
      // Return true or false
}[, thisArg])
```

```
const arr = [5, 4, 5, 3, 5, 2, 6];

function isOdd(x) {
  return x % 2;
}
const output = arr.filter(isOdd);
console.log(output);
```

or

```
const arr = [5, 4, 5, 3, 5, 2, 6];

function isOdd(x) {
  return x % 2 === 0;
}
const output = arr.filter(isOdd);
console.log(output);
```

reduce : The reduce() method reduces an array of values down to just one value. To get the output value, it runs a reducer function on each element of the array.
syntax :
arr.reduce(callback[, initial Value])

The call back argument is a function that will be called once for every item in the array. This function takes four arguments, but often only the first two are used.

- *accumulator* - the returned value of the previous iteration
- *current Value* - the current item in the array
- *index* - the index of the current item
- *array* - the original array on which reduce was called
- The initial Value argument is optional. If provided, it will be used as the initial accumulator value in the first call to the call back function.

WAP to find the sum or maximum number In array

```
const arr = [5,2,5,3,2,2];
```

```
function findSum(arr){
  let sum = 0;
  for (let i=0; i<arr.length; i++){
    sum = sum +arr[i];
  }
  return sum;

}
const output = arr.reduce(function(acc,curr){
acc = acc+curr;
return acc;
},0);
console.log(output);
```

OUTPUT : 19

Call: call: call is a predefined method in JS
call () method allows an object to use a method(function) of another object

```
let userDetails = {
  name: "Suraj",
  age: 27,
  designation : "Software engineer",
  PrintDetails:function(){
    console.log(this);
  }
}
userDetails.PrintDetails();

let userDetails2={
name: "Suraj kanwale",
age: 23,
designation : "Software engineer",
}
userDetails.PrintDetails.call(userDetails2);
```

or example :

```
let userDetails = {
  name: "Suraj",
  age:27,
  designation :"software engineer",
}
let printDetails = function(){
  console.log(this);
}
printDetails.call(userDetails);

let userDetails2 = {
  name: "Suraj Kanwale",
  age:28,
  designation :"software engineer",
}
// function borrowing
printDetails.call(userDetails2);
```

apply: apply method same like call method

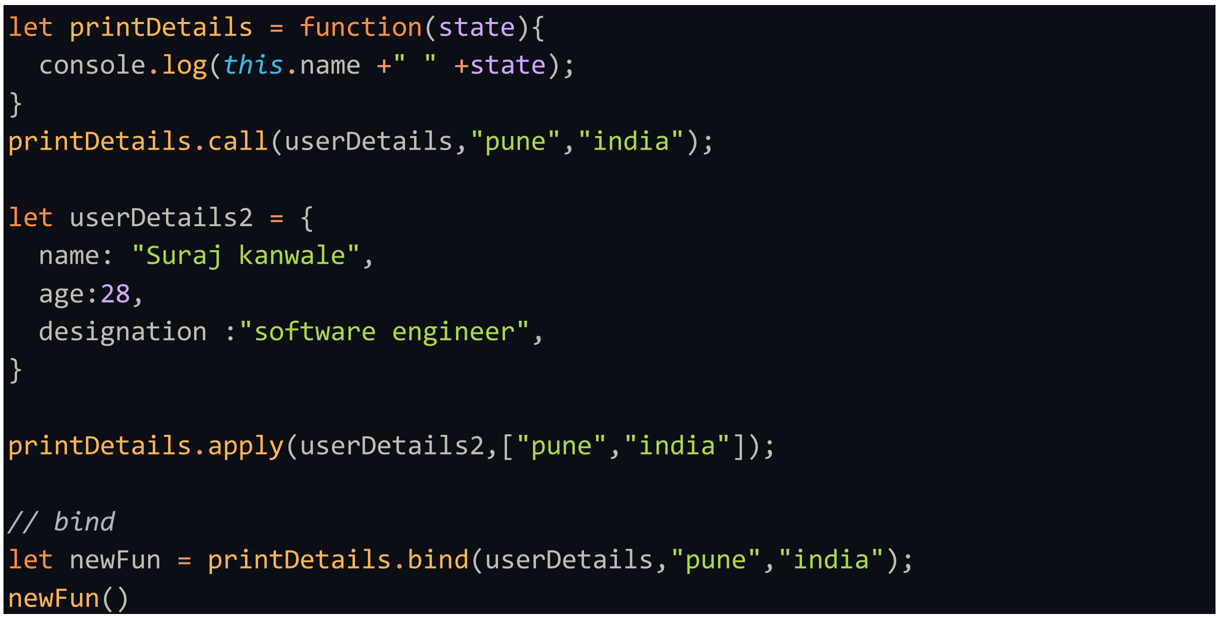
it allows you pass in an argument to the list of array format

```
let userDetails = {
  name: "Suraj",
  age:27,
  designation :"software engineer",
}
let printDetails = function(state){
  console.log(this.name +" " +state);
}
printDetails.call(userDetails,"pune","india");

let userDetails2 = {
  name: "Suraj Kanwale",
  age:28,
  designation :"software engineer",
}

printDetails.apply(userDetails2,["pune”, india"]);
```

bind : bind function to same as a call function it will take copy of the function

and then invoke latter]

```
let userDetails = {
  name: "Suraj",
  age:27,
  designation :"software engineer",
}
```



```
let printDetails = function(state){
  console.log(this.name +" " +state);
}
printDetails.call(userDetails,"pune","india");

let userDetails2 = {
  name: "Suraj kanwale",
  age:28,
  designation :"software engineer",
}

printDetails.apply(userDetails2,["pune","india"]);

// bind
let newFun = printDetails.bind(userDetails,"pune","india");
newFun()
```

Q.18 What is event bubbling and event capturing in JS ?

1 : Event Bubbling:

event bubbling means to get an event handled by the innermost element and then propagated to the outer elements

by default, event will event bubbling

child to parent

example:

```
var div=document. query Selector("div");

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

   console.log("button");

});
```

Output : div

2: Event capture :

the event capture procedure is rarely used

the event is captured first by the outermost element and then propagated to the innermost element var div=document.querySelector("div");

div.addEventListener('click’, () =>{

    console.log("div");},true);

3: Stop propagation :

it will stop the propagation of child to parent or parent to child

var div=document.querySelector("div");

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

    event.stopPropagation("div");

    console.log("div");

},true);

4. StopImmeditatePropogation

suppose we have multiple events then 1,2,3 after 3 event call, if we want stop the execution then we used for StopImmeditatePropogation

5: prevent default

if we want to stop by default behaviour for a particular event then we can use prevent default

example

anchor tag : it will navigate one page to another page, then we used stop page navigation

Q.19 What is shallow copy & deep copy ? How to copy objects on JS ?

1. Use the spread (...) syntax
2. Use the Object.assign() method
3. Use the JSON.stringify() and JSON.parse() methods

```
// normal copy
```

```
var obj1 = {
    name : "Suraj",
    age : 27
}
 var obj2 = obj1;
 console.log(obj1);
 obj1.name = "Suraj Kanwale"
 console.log(obj1);
```

2 : Shallow copy : Shallow copy is a bit-wise copy of an object. A new object is created that has an exact copy of the values in the original object. If any of the fields of the object are references to other objects, just the reference addresses are copied i.e., only the memory address is copied.

```
// shallow copy
var obj1 = {
    name : "Suraj",
    age : 27
}
    var obj2 = Object.assign({},obj1);
 var obj2 = {...obj1}

 console.log(obj1);
 obj1.name = "Suraj Kanwale"
 console.log(obj1);
```

3 : deep copy : A deep copy copies all fields, and makes copies of dynamically allocated memory pointed to by the fields. A deep copy occurs when an object is copied along with the objects to which it refers.

```
// deep copy
var obj1 = {
    name : "Suraj",
    age : 27
}
    var obj2 = JSON.parse(JSON.stringify(obj1))

 console.log(obj1);
 obj1.name = "Suraj Kanwale"
 console.log(obj1);
```

Q.20 what is Currying ?

Currying is the process of taking a function with multiple arguments and turning it into a sequence of functions each with only a single argument.

Currying is named after a mathematician **Haskell Curry**. By applying currying, a n-ary function turns it into a unary function.

```
// currying
function Addition(a) {
    return function (b) {
      return function (c) {
        return a + b + c;
      };
    };
  }
  let res = Addition(2)(3)(4);
  console.log(res);


  // Real time example
  userObj = {
    name: "Suraj",
    age: 23,
  };
  function userinfo(obj) {
    return function (userinfo) {
      return obj[userinfo];
    };
  }
  let result = userinfo(userObj);
  console.log(result("age"));
```

Q.21 What is Template Literals ?

ES6 introduces very simple string templates along with placeholders for the variables. The syntax for using the string template is ${PARAMETER} and is used inside of the back-ticked string.

```
// ES5
 var firstname = "Suraj";
 var lastname = "Kanwale";

 console.log("my name is " + firstname + " & my last name is " + lastname +
".");
```

```
//ES6

let firstname = "Suraj";
let lastname = "Kanwale"

console.log(`my name is ${firstname}  & my last name is ${lastname} . `  );
```

Q.22 What is rest operator ?

rest parameter is an improved way to handle function parameters, allowing us to more easily handle various input parameter in a function.

the rest parameter syntax allows us to represent an indefinite numbers

of arguments as an array

it is denoted as …

```
// ES5
function sum(a, b) {
  console.log(a + b);
}
sum(2 + 4 + 4 + 5 + 6);

// ES6
function sum(...inputs) {
  console.log(...inputs);
  let total = 0;
  for (let i of inputs) {
    total += i;
  }
  console.log(total);
}
sum(2, 3, 4, 5, 6, 7, 8, 4, 4, 4, 24, 45, 5);
```

Q.23 What is spread operator?

Spread operator allows iterables(arrays / objects / strings) to be expanded into single arguments/elements.

```

// ES6
function sum(a,b,c){
    console.log(a+b+c);
}
```

```
var arrVal = [3,4,5];
sum (...arrVal);
```

Q.24 What is default parameter ?

In E5, we need to depend on logical OR operators to handle default values of function parameters. Whereas in ES6,

the Default function parameters feature allows parameters to be initialized with default values if no value or undefined is passed.

```
//ES5

var calculateArea = function (height, width) {

  height = height || 50;

  width = width || 60;

  return width * height;

};

console.log(calculateArea()); //300
```

The default parameters makes the initialization more simpler,

```
//ES6

var calculateArea = function (height = 50, width = 60) {

  return width * height

};

console.log(calculateArea()); //300
```

Q.25 What is async & await ?

An async function is a function declared with the async keyword, and the await keyword is permitted within it.

The async and await keywords enable asynchronous, promise-based behaviour to be written in a cleaner style, avoiding the need to explicitly configure promise chains.

```
async function getdata(){
    let handlePromice = new Promise ((resolve,reject) => {
        setTimeout(()=> {
           resolve("all done");
        }, 1000);
    })

    let x= await handlePromice;
    // handlePromice.then((x)=>{
        // console.warn(x);
    // })
    console.warn(x);
}
getdata();

// real time fatch  data
async function getdata(){
    let data = fetch ("file path ");
    let = await data.json()
    console.warn(data);
}
```

Q.26 What is prototype in JS ?

Whenever you create any object. JavaScript engine automatically puts this hidden properties into and object and attached to your object. And you have access this

So if we want to access this hidden property

arr.__proto__

And this is the object where JavaScript putting this function and methods

In not case in array we can access object also

What is prototype chain ?

Each object has a private property which holds a link to another object called its **prototype**.

That prototype object has a prototype of its own, and so on until an object is reached with null as its prototype. By definition, null has no prototype, and acts as the final link in this **prototype chain**.

Object prototype is actually null is called prototype chain

```
let arr = ["suraj", "kanwale"];

let object = {
    name : "suraj",
    age : 20
}
```

Q.27 What is destructing in JS ?

Destructuring is a JavaScript expression that allows us to extract data from arrays, objects, and maps and set them into new, distinct variables.

Destructuring allows us to extract multiple properties, or items, from an array at a time

```
let student = {
  name: "Suraj",
  age: 27,
  course: "CSE",
  Address: {
    street: "Maharunge",
    city: "Pune",
    state: "Maharastra",
  },
};

// destructring
let { street, city, state } = student.Address;
console.log(`STREET: ${street} & CITY: ${city} & STATE : ${state}`);

// generally accessing the object
console.log(student.name);
```

Q.28 What are the new feature of ES6?

- let and const Keywords
- Arrow Functions
- Multi-line Strings
- Default Parameters
- Template Literals
- Destructuring Assignment
- Enhanced Object Literals
- Promises
- Classes
- Modules

Q.29 what is event loop and how it work ?

JavaScript is a single-threaded programming language, which means it executes all of the instructions line by line in a synchronous manner.

Thus since everything works on the main thread, there seems to be no possibility of executing parallel processes in JavaScript.

Memory Organization of JavaScript:

The JavaScript Engine consists of two main components:

- **Memory Heap** — this is where the memory allocation happens, all of our object variables are assigned here in a random manner.
- **Call Stack** — this is where your function calls are stored.

Q.30 what is DOM ? How to access the tag

DOM means changing the dynamic content of the web page.

we can access the HTML Tag and modify then and return back to the HTML element,

// nav

CSS => nav{}

JavaScript => document.querySelector('nav');

jQuery => $('nav');

// access id

Nav id = 'test'

CSS : #id{ }

JavaScript : document.querySelector('#test');

jQuery : $('test');

Nav class = container

CSS => .container {}

JavaScript => document,querySelector('.container');

jQuery => $('container');

```
// access document
console.log(document);

// access head section
console.log(document.head);

// accees title section
console.log(document.title);

// accees body section
console.log(document.body);


// access nav
let navTag = document.querySelector('nav');
console.log(navTag);

// anchor tag

let anchorTag = document.querySelector('nav a')
console.log(anchorTag);

// how to static content into dynamics

anchorTag.innerText = 'Suraj';

// how to static content into dynamics  h1 Tag
let h1Tag = document.querySelector('#msg');
h1Tag.innerText = 'Good Evening';


// style property

h1Tag.style.backgroundColor = 'red';
h1Tag.style.color = 'white';
h1Tag.style.textAlign = 'center';
h1Tag.style.padding = '15px';
h1Tag.style.fontSize = '25px';
h1Tag.style.boxShadow = ' 0 0 10px black';
```

```html
</head>

<body>
    <nav>
        <a href="#">Suraj Kanwale</a>
    </nav>

    <div>
        <h1 id="msg">Good morning</h1>
    </div>

    <script src="index.js"></script>
</body>
</html>
```

Q.31 What is "use strict"; Mode ?

Q.32 what is local storage and session storage ?

local Storage: local storage means it is permanent storage Until unless you delete it, it will be displayed,

if you closed the browser, and again open, or restart the server data is still there

session Storage: is only per session, per session means if you closed the browser then it will be clear, until unless we open the browser data will still be there, and close your browser, session end, and data close

Q.33 what are the array method in js