1 What is Javascript.

Ans-"JavaScript is a powerful, high-level, interpreted programming language that allows us to create dynamic and interactive web applications. It runs in the browser's engine like V8 and follows the ECMAScript standard. It is single-threaded but supports asynchronous behavior through its event loop. JavaScript treats functions
as first-class citizens and uses prototypal inheritance, allowing both functional and OOP styles. It's also used on the server-side via Node.js, making it a full-stack language."
it executes code line by line it is single threaded

2 How Borwser Execute Javascript.

"When a browser loads a webpage, the HTML parser encounters JavaScript code inside <script> tags. This code is sent to the browser's JavaScript engine like V8 in Chrome. The engine processes the code in two phases: the compilation phase, where memory is allocated and function declarations are hoisted, and the execution phase, where the code is run line-by-line using a call stack. For asynchronous operations, the browser provides Web APIs and a task queue. Once the call stack is empty, the event loop pushes the callback functions into the stack to continue execution. This entire system ensures both synchronous and asynchronous JavaScript can run smoothly in the browser environment.

3 What are features of Javascript.

"JavaScript is a high-level, interpreted, dynamically typed language that su
pports object-oriented and functional programming. It is single-threaded but handles asynchronous operations efficiently using its **event loop** and **callback queue**. JavaScript treats functions as first-class citizens, supports closures, and enables prototypal inheritance. It's also cross-platform and works seamlessly in both browser and server environments, like with **Node.js**."

- "Can you name some core features of JavaScript?"
  "Yes! JavaScript is high-level, dynamically typed, single-threaded but asynchronous using event loop, and supports first-class functions, closures, and prototypal inheritance. It runs in both browser and server environments like Node.js."

4 What is DOM ?

"DOM stands for **Document Object Model**. It is a programming interface provided by the browser that represents the structure of a web page as a **tree of objects**. Using JavaScript, we can dynamically access, change, or remove any element of the page through the DOM. Essentially, DOM connects HTML with JavaScript, making the page interactive."

Deep Explanation in Simple Words:

- Jab browser koi HTML page load karta hai, usse **parse** karke ek **in-memory tree structure** banata hai.
- Ye tree HTML ke har element (like , <p>, <h1>) ko ek **JavaScript object** ke form me convert karta hai.
- Is tree ko hi hum **DOM** kehte hain.
- JavaScript DOM ke through kisi bhi element ko:
    - **Access** kar sakta hai
    - **Modify** kar sakta hai
    - **Delete** ya **Add** kar sakta hai

let heading = document.getElementById("title");
heading.innerText = "Welcome!";

- `document` → pura DOM represent karta hai
- `getElementById` → DOM se element ko access karta hai
- `innerText` → us element ka content change karta hai

Document.getElementById("btn").addEventListener("click")

5 How to perform click event on Dom Using Javascript

A **click event** in JavaScript is triggered when a user clicks on a DOM element, like a button or link. We can handle this using **event listeners**, typically with the addEventListener() method or the onclick property.

Using addLisnter()

```
<button id="myBtn">Click Me</button>

<script>
  const button = document.getElementById("myBtn");
  button.addEventListener("click", function () {
    alert("Button was clicked!");
  });
</script>
```

- Allows multiple event listeners
- Keeps HTML and JS separate (clean code)

2. **Using onclick Property**

```
<button id="myBtn">Click Me</button>
<script>
  document.getElementById("myBtn").onclick = function () {
    alert("Button Clicked!");
  };
</script>
```

🎯 **Why avoid this in large apps?**

- Only one handler allowed per event

- Less flexible

Bonus: How Click Event Works Behind the Scenes

1. Browser loads HTML → DOM is built
2. JavaScript selects the element (getElementById)
3. A function is registered for the "click" event
4. When the user clicks, browser triggers the callback function

6 What is alert() and confirm() ?

"In JavaScript, alert() is used to display a simple message to the user with an OK button, and it pauses code execution until the user clicks OK. On the other hand, confirm() is used to get confirmation from the user with OK and Cancel buttons, and it returns a boolean: true for OK, and false for Cancel. Both are blocking modal popups provided by the browser."

7 What is difference between undefined and null

`undefined` means a variable has been declared but not assigned any value yet.
`null` is an intentional absence of any value — set manually by the developer.

8 What is difference between == and ===

In JavaScript, == is the **loose equality operator** that compares values after performing **type coercion**, whereas === is the **strict equality operator** that checks both **value and data type** without type conversion.

9 What is function . Can we overload function in javascript.

A **function** in JavaScript is a **block of reusable code** that performs a specific task. Functions allow us to write logic once and reuse it multiple times with different inputs.

Example-

```
function greet(name) {
  return "Hello, " + name;
}
console.log(greet("Arjun")); // Hello, Arjun
```

- **Types of Functions in JavaScript:**
  1. **Function Declaration**
  2. **Function Expression**
  3. **Arrow Function**
  4. **Anonymous Function**

5. **Immediately Invoked Function Expression (IIFE)**

- **Can we overload function in javascript.**

JavaScript does not support traditional function overloading like Java or C++.

10 What is callback ? Explain with suitable example.

Ans- "A callback is a function that is passed as an argument and is executed after another function completes its operation. In JavaScript, callbacks are heavily used in asynchronous programming to avoid blocking the main thread. For example, setTimeout, event listeners, and file/network requests all use callbacks to execute code once an operation finishes."

Example-

```
console.log("Start");
setTimeout(function () {
  console.log("This runs after 2 seconds");
}, 2000);
console.log("End");
```

11 What is promise? What are advantage of promise over callback.

Ans-"A Promise is a cleaner alternative to callbacks for handling asynchronous operations. It helps avoid callback hell by allowing method chaining using .then() and error handling with .catch(). Promises make code more readable, maintainable, and structured compared to deeply nested callbacks."

```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const data = "Data from server";
      resolve(data); // or reject("Error while fetching")
    }, 2000);
  });
}

fetchData()
  .then(result => {
    console.log("Success:", result);
  })
  .catch(error => {
    console.error("Failed:", error);
```

```
});
```

12 What is callback hell.

Ans-Callback hell is a situation in JavaScript where you have **multiple nested callbacks**, making your code **difficult to read, understand, debug, and maintain**.

It usually happens when you perform **asynchronous operations** that depend on each other in a sequence.

13 What is closure ?

Ans- A **closure** is a function that **remembers and has access** to variables from its **outer (lexical) scope**, even after the **outer function has finished executing**.

Example-

```
function outer() {
  let count = 0;  // outer function variable

  return function inner() {
    count++;              // inner function uses outer variable
    console.log(count);
  };
}
const counter = outer();   // outer() returns inner function
counter();  // Output: 1
counter();  // Output: 2
```

14 What is IIFEs(Immediately Invoked Function expression).

Ans- "An IIFE is a JavaScript function that runs immediately after it's defined. It creates a private scope and helps avoid polluting the global environment. It is useful for encapsulation, especially in modular or DOM-related code."

Example-

```
const counter = (function () {
  let count = 0; // private variable

  return {
    increment: function () {
      count++;
```

```
    console.log("Count:", count);
  },
  decrement: function () {
    count--;
    console.log("Count:", count);
  },
  reset: function () {
    count = 0;
    console.log("Reset to 0");
  }
 };
})();

// Usage
counter.increment(); // Count: 1
counter.increment(); // Count: 2
counter.decrement(); // Count: 1
counter.reset();     // Reset to 0
```

15 What is annonymous function ?

Ans-"Anonymous functions are often used for **one-time execution** when you don't need to reuse the function. For example, event handlers or setTimeout() calls are common scenarios where anonymous functions are executed once, without needing a separate named function."

Example-

```
const sum = function (a, b) {
  return a + b;
};
console.log(sum(3, 4));  // Output: 7
```

16 What is hoisting in javascript.

"Hoisting is a JavaScript behavior where function and variable declarations are moved to the top of their scope before the code is executed. With function declarations, the entire function is hoisted, while with variables (declared with var), only the declaration is hoisted, not the initialization. This is why accessing variables before they are assigned can lead to undefined or errors."

Some points-

1. **Function Declarations** are hoisted with their implementation.
2. **Variables declared with var** are hoisted, but initialized with undefined.
3. **let and const** are hoisted, but **cannot be accessed** before initialization.
4. **Function Expressions** are not hoisted like function declarations.

```
1)  greet();  // Output: Hello, World!
    function greet() {
      console.log("Hello, World!");
    }
```

➡️ **Yahaan function declaration hoist ho jata hai, isliye greet() ko usse pehle call kiya jaa sakta hai.**

```
2) console.log(x); // Output: undefined
    var x = 5;
```

➡️ **Yahaan pe, var x ko hoisted kar diya jata hai, lekin uska value (5) us time set nahi hota. Isliye undefined show hota hai, na ki 5.**

```
3) foo();  // Error: foo is not a function
var foo = function () {
  console.log("Function Expression");
};
```

➡️ **Function Expressions ko hoisted nahi kiya jata. Yahaan pe foo ek variable hai jisme ek function expression assign kiya gaya hai, toh foo() ko call karte waqt foo undefined hai**.

17 What is difference between var, let and const keyword.

Ans- "var is function-scoped and can be redeclared and reassigned. let and const are block-scoped. let is reassignable but not redeclarable, while const is neither reassignable nor redeclarable after its initial assignment. Also, let and const are hoisted but not initialized, unlike var."

🔹 1. var:

- **Scope:** Function-level scope (function ke andar hi accessible hota hai).
- **Hoisting:** Hoisted, but initialized with undefined.
- **Reassignable:** Yes, variables declared with var can be reassigned.
- **Redeclarable:** Yes, var can be redeclared in the same scope without error.

Example-

```
var x = 10;
x = 20;  // Reassignable
console.log(x);  // Output: 20

var x = 30;  // Redeclared
console.log(x);  // Output: 30
```

🔹 2. let:

- **Scope:** Block-level scope (only accessible inside the block it is defined in).
- **Hoisting:** Hoisted, but not initialized (temporal dead zone).
- **Reassignable:** Yes, variables declared with let can be reassigned.
- **Redeclarable:** No, let cannot be redeclared in the same scope.

Example-

```
let y = 10;
y = 15;  // Reassignable
console.log(y);  // Output: 15

// let y = 20;  // Error: Cannot redeclare block-scoped variable 'y'
```

🔹 3. const:

- **Scope:** Block-level scope (similar to let).
- **Hoisting:** Hoisted, but not initialized (temporal dead zone).
- **Reassignable:** No, variables declared with const cannot be reassigned after initialization.
- **Redeclarable:** No, const cannot be redeclared in the same scope.

Example

```
const z = 10;
// z = 20;  // Error: Assignment to constant variable.
console.log(z);  // Output: 10

// const z = 30;  // Error: Cannot redeclare block-scoped variable 'z'
```

18 Explain use strict

Ans-"use strict" is a special mode in JavaScript that makes your code more secure. It helps catch common mistakes like using undeclared variables or duplicate declarations.
You can apply it globally or inside a function to enforce stricter rules.

19 What is event bubbling and event capturing

20 What are the primitive datatype in javascript

Ans- "JavaScript has 7 primitive data types: String, Number, Boolean, Undefined, Null, Symbol, and BigInt. These are immutable and stored by value."

21 What are different types of popup boxes available in javascript

Ans - **1. Alert Box**

- **Purpose**: Displays a message to the user
- **Appearance**: Contains a single "OK" button.
- **Usage**: Commonly used to convey information or warnings.
- **Behaviour**: Pauses code execution until the user clicks "OK".
- **Example**:

```
alert("This is an alert message!");
```

2. Confirm Box

- **Purpose**: Asks the user to confirm an action.
- **Appearance**: Contains "OK" and "Cancel" buttons.
- **Returns**: true if "OK" is clicked, false if "Cancel" is clicked.
- **Example**:

```
let result = confirm("Do you want to proceed?");
if (result) {
  // User clicked OK
} else {
  // User clicked Cancel
}
```

3. Prompt Box

- **Purpose**: Collects user input.
- **Usage**: Used for gathering data like names or preferences.
- **Syntax**:

```
let userInput = prompt("Enter your username:");
```

- **Returns**: User input as a string, or null if "Cancel" is clicked.
- **Example**:

```
let name = prompt("What's your name?");
if (name) {
  alert("Hello, " + name + "!");
```

```
} else {
  alert("No name entered.");
}
```

Key Differences

Feature	Alert Box	Confirm Box	Prompt Box
Purpose	Display information	Get user confirmation	Collect user input
Buttons	OK only	OK and Cancel	OK and Cancel
Return Value	undefined	true or false	User input string or null
Use Case	Notifications, warnings	Confirming actions (e.g., delete)	Gathering data from user

22 What will happen if an infinite while loop is run in javascript

Ans-"Since JavaScript is single-threaded, running an infinite while loop will block the main thread. This means the browser cannot respond to any user interaction or execute other scripts, leading to UI freeze or crash. No events or asynchronous operations can be processed because the Event Loop gets stuck waiting for the infinite loop to end, which it never does."

Example- "To avoid blocking the thread, I'd consider using setInterval or recursive setTimeout, which allows the Event Loop to remain unblocked and responsive."

23 List HTML DOM mouse events

onclick,ondblclick, onmouseover, onmouseout, onmouseup,onmousemove

Ans- "HTML DOM mouse events are triggered based on user interaction with the mouse. For example, onclick for clicking, ondblclick for double-clicking, onmouseover and onmouseout for hover effects, onmouseup when the mouse button is released, and onmousemove when the mouse moves. These events allow us to make web pages interactive and dynamic."

==

- onclick: when the mouse clicks an element
- ondblclick: when the mouse double-clicks
- onmouseover: when the mouse hovers over
- onmouseout: when the mouse leaves the element
- onmouseup: when the mouse button is released
- onmousemove: when the mouse moves over the element

How we use daynamically in javascript

```
const btn = document.getElementById("myBtn");
btn.addEventListener("click", () => {
```

```
  alert("Clicked via JS!");
});
```

24 How to get the last index of a string in javascript lastIndexOf

Ans- "In JavaScript, lastIndexOf() helps to find the last position of a character or substring in a string. It returns the index of the last match or -1 if not found. You can also specify a starting point for the search with the optional fromIndex parameter."

Example –

```
const str = "banana";
console.log(str.indexOf("a")); // Output: 1 (pehla 'a' ka index)
console.log(str.lastIndexOf("a")); // Output: 5 (a ka last occurrence index)
```

Example-LastIndexOf

```
const str = "JavaScript is awesome!";
const lastIndex = str.lastIndexOf("a");
console.log(lastIndex); // Output: 15 (last 'a' ka index)
```

lastIndexOf whis specific charecter

```
const str = "JavaScript is awesome!";
const lastIndex = str.lastIndexOf("a", 10);
console.log(lastIndex); // Output: 7 (searching from index 10, last 'a' milta hai index 7 pe)
```

If value is not found

```
const str = "Hello, world!";
const lastIndex = str.lastIndexOf("z");
console.log(lastIndex); // Output
```

 indexOf() sirf pehla occurrence dikhata hai, jabki lastIndexOf() last occurrence dikhata hai.

25 Describe negative infinity in javascript

Ans- "-Infinity in JavaScript means a value smaller than any other number. It usually appears when a negative number is divided by zero, like -10 / 0. It's treated like a number and can be compared or used in calculations."

26 Explain await and aync ? How to use await and async?

Ans-"async and await are used to handle asynchronous code in a cleaner and more readable way. "async and await are used in JavaScript to handle asynchronous operations in a clean and readable way.

When I write async before a function, it means that the function will always return a **Promise**. Even if it returns a normal value, JavaScript will automatically convert it into a Promise.

Inside that async function, I can use await to pause the execution **until a Promise is resolved or rejected**. This makes the code behave like synchronous code, so it's easier to read and manage.
For example, instead of using .then() and .catch() to handle API calls, I can write:
Example-

```
async function getData() {
  const response = await fetch("https://api.com/data");
  const data = await response.json();
  console.log(data);
}
```

This way, the code is cleaner, there's less nesting, and I can handle errors easily with a try-catch block."

"await can only be used inside an async function. If I use it outside, JavaScript will throw a syntax error. That's why both always go together."

27 How to handle the excption in javascript.

Ans- "In JavaScript, I handle exceptions using try, catch, and optionally finally.
The try block contains the code that might throw an error. If any error occurs, it jumps to the catch block, where I can handle the error gracefully. The finally block is optional, and it always runs — whether there's an error or not. It's useful for cleanup tasks like closing connections or hiding loaders.
For example:

```
try {
  let result = riskyFunction(); // might throw an error
  console.log(result);
} catch (error) {
  console.log("Error occurred:", error.message);
} finally {
  console.log("This will always run");
}
```

This way, the app doesn't crash and I can control how errors are handled."

Tips-

"This is important in real-world apps — especially when working with APIs, file systems, or user inputs — to keep the user experience smooth and bug-free."

28 What is Node.js? Explaing the advantage of Node.js Over java and php

Simple Answer: Node.js is a JavaScript runtime built on Chrome's V8 engine that allows you to run JavaScript outside the browser, like on a server.
Why it's better than Java/PHP:

- Uses JavaScript on both frontend and backend.
- Non-blocking and event-driven (handles many requests at once).
- Lightweight and fast due to the V8 engine.
- Great for real-time apps like chat or live games.

Impressive One-liner: "Node.js is lightweight, fast, and non-blocking — perfect for real-time, scalable applications."

29 What are the limitation of Node.Js

30 How Node.Js Works

Ans- "Node.js works on a single-threaded event loop architecture. Instead of waiting for tasks like reading a file or accessing a database to complete, Node.js sends them to the background and continues handling other incoming requests. This is called non-blocking I/O. It allows Node.js to serve thousands of users at the same time without getting stuck or slowed down. That's why it's perfect for real-time apps like chat, gaming, or APIs."

Example-

```
Console.log("start the execution….")
setTimeout(()=>{
console.log(Async task……..);
},2000);
Console.log("and");
```

Output-

Start the execution ……

And

Aysnc task ……

31 How Node.js is single threaded

Ans- "Node.js is single-threaded because it uses a single main thread to run JavaScript code. This thread handles one task at a time using an event loop. But when a task takes time, like reading a file or calling an API, Node.js sends it to the background using internal threads and continues running other code. Once the task is done, the callback is added to the event loop and gets executed. So, even though it's single-threaded, it handles many tasks efficiently using non-blocking I/O."

If interviewer asks **"Why single thread?"**, you can say:
"Because JavaScript was designed to run in the browser with a single thread, and Node.js keeps that model for simplicity and speed, while using async patterns to stay non-blocking."

32 Explain any five built-in package/Dependency name in node.js

Ans- **1. fs (File System)**
Use: To work with files (read/write/delete).
Import:

```
import fs from 'fs';
fs.readFile('file.txt', 'utf-8', (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

Interview Tip:
"I use fs module for file operations like reading and writing files on the server."

---

2. http
Use: To create a simple web server.
Import:

```
import http from 'http';

const server = http.createServer((req, res) => {
  res.end('Hello from Node.js server');
});
server.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

Interview Tip:
"http helps me build basic web servers directly without any frameworks."

---

3. path
Use: To manage file paths easily.
Import:

```
js
CopyEdit
import path from 'path';
import { fileURLToPath } from 'url';
import { dirname } from 'path';

const __filename = fileURLToPath(import.meta.url);
```

```
const __dirname = dirname(__filename);

const fullPath = path.join(__dirname, 'folder', 'file.txt');
console.log(fullPath);
```

Interview Tip:
"I use the path module to safely work with file and folder paths, especially when paths get complex."

---

4. os
Use: To get operating system details.
Import:

```
import os from 'os';

console.log('OS platform:', os.platform());
console.log('Total memory:', os.totalmem());
```

Interview Tip:
"os module is useful to access system-level information like CPU, memory, and OS type."

---

5. events
Use: To handle custom events.
Import:

```
import { EventEmitter } from 'events';

const emitter = new EventEmitter();

emitter.on('welcome', () => {
  console.log('Welcome to Node.js Events!');
});

emitter.emit('welcome');
```

Interview Tip:
"The events module helps me build event-driven features, like custom triggers in real-time apps."

✅ **Bonus Tip to Say at the End:**
"Using built-in modules with import syntax makes my code modern, modular, and clean — and I don't need to install anything extra for these."

33 What is module in node.js

Ans- "A module in Node.js is simply a separate file that contains some code like functions, variables, or classes, and we can export and reuse it in other files. It makes our code more organized and reusable."
"Suppose I have a file that only handles math operations. Instead of writing all code in one big file, I create a separate math module and use it wherever needed."

34 what is Module.exports

Ans-"module.exports in Node.js is used to export functions, objects, or variables from one file so that they can be reused in another file. It helps in code reusability and keeping the project organized."

35 How to create server in node.js

Ans- "In Node.js, we use the built-in 'http' module to create a server. We call 'http.createServer' to handle incoming requests and send responses, and we listen on a port like 3000 to accept connections."

Example-

```
// Step 1: Import http module
import http from 'http';

// Step 2: Create server
const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' }); // Status and headers
  res.end('Hello, World!'); // Response to client
});

// Step 3: Listen on port
server.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

36 How node.js handle multiple request

Ans-

"Node.js handles multiple requests using an event-driven, non-blocking architecture. Even though it has a single thread, it sends time-consuming tasks like database operations to the background, continues handling other requests, and when the task is finished, it picks up the result using the event loop and callback functions. This way, Node.js can handle thousands of requests at once without waiting."

Example-

```
import http from 'http';
const server = http.createServer((req, res) => {
  if (req.url === "/heavy") {
    // Simulate heavy task
    setTimeout(() => {
      res.end("Heavy task done!");
    }, 3000);
  } else {
    res.end("Simple response");
  }
});

server.listen(3000, () => {
  console.log('Server running at http://localhost:3000');
});
```

37 How to use url module in node.js

"In Node.js, the url module is used to parse and manage URLs easily. It helps in breaking down a full URL into parts like protocol, hostname, pathname, and query parameters, which makes it easier to work with incoming requests."

Example-

```
 import { parse } from 'url'; // Modern import style (no require)

const myUrl = 'http://localhost:3000/home?name=John&age=25';

// Parse the URL
const parsedUrl = parse(myUrl, true);  // true to get query as object

console.log(parsedUrl.hostname); // 'localhost'
console.log(parsedUrl.pathname); // '/home'
console.log(parsedUrl.query);    // { name: 'John', age: '25' }
```

38 What is setInterval, setTimeout

Ans- "**In JavaScript, setInterval() and setTimeout() are used to schedule code execution. setInterval() runs a function repeatedly after a given time interval, while setTimeout() executes a function once after a specified delay. These are part of JavaScript's timer functions to handle asynchronous operations.**"

```
// This will log "Hello" every 2 seconds
```

```
setInterval(() => {
  console.log("Hello");
}, 2000); // 2000ms = 2 seconds
// This will log "Hello after 3 seconds" after 3 seconds
setTimeout(() => {
  console.log("Hello after 3 seconds");
}, 3000); // 3000ms = 3 seconds
```

Key Differences:

- **setInterval()**: Executes the code repeatedly at intervals.
- **setTimeout()**: Executes the code only once after a delay.

ressive One-liner for Interview:

"`setInterval()` is used for repeating tasks at regular intervals, while `setTimeout()` is for one-time execution after a delay."

39 What is __dirname and __filename

Ans- "In Node.js, __dirname and __filename are two special variables that provide information about the file system. __dirname gives the directory path of the currently executing file, while __filename gives the absolute file path, including the file name itself. These variables are useful for working with relative file paths, loading modules, or performing file operations

1. **__dirname**: Provides the directory path.
2. **__filename**: Provides the full path of the file.

40 What is synchronous/Blocking and Asynchronous/Non-blocking code in node.js

Ans- synchronous-

"Synchronous or blocking code in Node.js means the program runs line by line, and each line waits for the previous one to finish before moving ahead. If one line takes time, the whole program waits.

```
const fs = require('fs');

const data = fs.readFileSync('file.txt', 'utf8');
```

```
console.log(data);  // This line waits until file is completely read
console.log('This prints after reading the file');
```

Asynchronous-

Asynchronous or non-blocking code means the program doesn't wait. It can move to the next task while waiting for the previous one (like file read or API call) to complete in the background. This is very useful in Node.js because it helps handle many requests without getting stuck or blocked."

```
const fs = require('fs');

fs.readFile('file.txt', 'utf8', (err, data) => {
  console.log(data); // This runs later, after file is read
});

console.log('This prints first, without waiting');
```

"Synchronous code blocks the flow and waits for each task, while asynchronous code continues execution and handles time-consuming tasks in the background using callbacks, promises, or async/await."

41 What is file system in node.js

Ans- The **File System (fs)** module in Node.js allows you to **interact with files and folders** on your system — like reading, writing, updating, deleting files, and creating directories. This module comes **built-in** with Node.js, so you don't need to install it.

Mostly file system operations-

- fs.readFile() – to read files.
- fs.writeFile() – to write or overwrite files.
- fs.appendFile() – to add to an existing file.
- fs.unlink() – to delete a file.
- fs.mkdir() – to create folders.

1. Read File – fs.readFile()

1) import fs from 'fs';

```
fs.readFile('./demo.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log('File Content:', data);
});
```

✅ **2. Write File – fs.writeFile()**

```
import fs from 'fs';

fs.writeFile('./demo.txt', 'Hello from Node.js!', (err) => {
  if (err) throw err;
  console.log('File written successfully!');
});
```

✅ 3. Append File – fs.appendFile()

```
js
CopyEdit
import fs from 'fs';

fs.appendFile('./demo.txt', '\nNew line added!', (err) => {
  if (err) throw err;
  console.log('Data appended successfully!');
});
```

✅ 4. Delete File – fs.unlink()

```
js
CopyEdit
import fs from 'fs';

fs.unlink('./demo.txt', (err) => {
  if (err) throw err;
  console.log('File deleted successfully!');
});
```

✅ 5. Create Directory – fs.mkdir()

```
js
CopyEdit
import fs from 'fs';

fs.mkdir('./newFolder', (err) => {
  if (err) throw err;
  console.log('Folder created!');
});
```

✅ 7. Rename File – fs.rename()

```
import fs from 'fs';

fs.rename('./demo.txt', './renamed.txt', (err) => {
  if (err) throw err;
  console.log('File renamed successfully!');
});
```

42 What are the differenct type of flag using in node.js

Ans- Node.js supports various flags for execution, debugging, performance, experimental features, and configuration.
These flags help developers customize behavior and improve development and deployment workflows.

(In my **Skill Exchange Platform**, I used the --env-file flag to load sensitive environment variables during development.
I also used --inspect while debugging real-time messaging issues in the backend.

```
node --env-file=.env.development --inspect server.js)
```

43 What is stream in node.js? Explain the types of stream

Ans- Streams in Node.js allow you to read/write data piece by piece instead of all at once. There are four types: **Readable, Writable, Duplex, and Transform**.
They are commonly used for **file I/O, data compression, and real-time applications.**

If asked:
"How are streams different from buffers?"
Say:
Buffers load all data in memory at once. Streams handle data in chunks, which makes them **more efficient for large-scale or real-time processing**.

44 How to pipe stream in node.js

Ans- In Node.js, the .pipe() method is used to **connect two or more streams**—like connecting a readable stream to a writable stream. This allows data to **flow automatically** from the source to the destination, without manually handling events like data, end, or write.
It's commonly used for **file reading/writing**, **data transformation**, and **real-time streaming**.

45 What is request and response in node.js
46 What is package.json and package.lock
47 What is npm ? How to install dependency/module at application level and Environment elevel
48 How do you manage packages in your node.js project
49 How Node.js is better then other framework
50 What are the some commonly used timing features of Node.js
51 What is fork in node.js
52 How do you create a simple server in node.js that return hello world
53 How many types of API functions are there in node.js
54 What is REPL and how to use it?
55 What is purpose of module.exports.

56 What is an event-loop in Node.js
57 If Node.js is single threaded then how does it handle multiple request/ concurrency
58 Differenctiate between process.nextTick() and setImmediate()
59 What is node.js stream
60 What is middleware
61 Explain what a reactor pattern in node.js
62 Describes the exit code of Node.js
63 What is an EventEmitter in node.js
64 What is a thread pool and which library handles it in node.js
65 What is purpose of NODE_ENV
66 How would you connection mongodb database to node application
67 What are different type of http request
68 What is difference between get and post
69 What is query string and how to send the data in get request
70 What is the use body parser
71 How to set the path of static file in express
72 What are types of Middleware in express ? Explain with suitable example
73 Does order of middleware matters in express.

Ans- **Yes, the order of middleware in Express.js is extremely important.**
Middleware functions in Express are executed **in the order they are defined** in your code. If a middleware is placed after a route or another middleware, it **won't be executed** for earlier requests.

So yes, **middleware order absolutely matters** in Express.
Defining them in the correct sequence is critical to ensure your app's routes, validations, and protections work as expected.

74 What is express.js
75 What are some distinctive features of Express

Ans- Express.js is a **fast, lightweight, and unopinionated backend framework** built on top of Node.js. It provides many powerful features that make building web applications and RESTful APIs easier and cleaner.
Here are some of its most distinctive features:

1) Routing System
2) Middleware Support
3) **RESTful API Support**
4) Easy Integration with Databases
5) **Templating Support**
6) **Error Handling Mechanism**
7) **Minimal & Unopinionated**

76 is Express.js fron-end or backend framework?

Ans- It is used to build **server-side logic**, create **RESTful APIs**, manage **HTTP requests and responses**, handle **middleware**, and connect with databases like MongoDB or MySQL. It doesn't deal with the **user interface (UI)** or browser-based logic — that's the job of frontend frameworks like **React.js**, **Angular**, or **Vue**.

77 Why do we use express.js

Ans- **Express.js** is a **fast, minimalist web framework for Node.js** that simplifies the process of building **web servers**, **RESTful APIs**, and **backend logic**.
While Node.js provides the core functionality to create a server, Express adds a powerful set of features that make web development much easier and more organized.

78 What is difference between express.js and node.js
79 What do you understand by Scaffolding in Express.Js
80 Which are the argument available to an Express.Js route handler function
81 How can you allow CORS in Express.jS?
82 How can you deal with Error handling in Express.js? Explain with an example
83 Write a code to start serving static file Express.JS
84 How can we render plain HTML in express
85 How can we send the data while rendering page in express
86 How to enable debugging in express app?
87 What is routing and How routing works in express
88 How dynamic routing works in express.js

89 What is Middleware in Express.js?

Ans-"Middleware in Express.js is a function that runs between request and response to handle tasks like validation, authentication, logging, and more."

"Middleware functions act like middle steps between request and response. They can check, modify, or stop a request before sending the final response.

Important Points:

- Middleware functions **always have** (req, res, next) parameters.
- next() is used to move to the next middleware or route.
- Middleware can be **global** (app.use()) or **specific** to a route.
- We can have **multiple middlewares**.

```
import express from 'express';

const app = express();

// Simple Middleware
```

```
app.use((req, res, next) => {
  console.log('This is a middleware');
  next();  // pass to the next middleware or route
});

// Route
app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(3000);
```

90 What is the req and res in express js

"In Express.js, req is used to get client request data like body, params, query, and res is used to send back the response to the client like JSON, HTML, or status codes."

In **Express.js**:

- req = **Request** → jo data client (browser ya app) server ko bhejta hai.
- res = **Response** → jo data server client ko wapas bhejta hai.

Example-

```
app.get('/hello', (req, res) => {
  console.log(req.query); // Request data from client
  res.send('Hello World!'); // Response to client
});
```

91 How to handle different http methods in express

"In Express.js, we handle different HTTP methods by using app.get(), app.post(), app.put(), and app.delete() functions to perform different actions for different types of client requests."

"In Express.js, we use different functions like app.get(), app.post(), app.put(), and app.delete() to handle different types of HTTP requests separately. This helps organize the server code properly based on what the client wants to do — like reading data, creating new data, updating, or deleting it."

Example -

```
app.get('/', (req, res) => {  // for GET
```

```
  res.send('GET Request received');
});

app.post('/', (req, res) => { // for POST
  res.send('POST Request received');
});

app.put('/', (req, res) => {  // for PUT
  res.send('PUT Request received');
});

app.delete('/', (req, res) => {  // for DELETE
  res.send('DELETE Request received');
});
```

93. How to create a rest api in node js with express
Explain react-
React is a **JavaScript Library** known for front-end development (or user interface). It is popular due to its **component-based architecture**, **Single Page Applications (SPAs),** and **Virtual DOM** for building web applications that are fast, efficient, and scalable.

How to create rest API-
Ans- "To create a REST API in Node.js, we first install Express.js, then create an app using express(). After that, we define different routes for GET, POST, PUT, and DELETE methods based on what client requests. We send back JSON responses. Finally, we start the server using app.listen()."

Step	Action
1	npm init -y
2	npm install express
3	Create app.js
4	Write Express server code
5	Define routes for GET, POST, PUT, DELETE
6	Start server with app.listen(3000)

94. Difference between put and patch
"PUT = Replace full resource"
"PATCH = Change only what is needed"

"In REST APIs, PUT is used for full updates where the entire resource is replaced with new data, while PATCH is used for partial updates where only a specific part of the resource is modified. PUT expects complete data, PATCH needs only the fields that are changing."

1. **Meaning**:
    - **PUT** updates the **entire resource**.
    - **PATCH** updates **only a part** of the resource.
2. **Data sending**:
    - **PUT** needs **full updated data**.
    - **PATCH** needs **only the changed fields**.
3. **Behavior**:
    - **PUT** replaces the **old data completely**.
    - **PATCH modifies** only the given parts, rest remains unchanged.
4. **Usage**:
    - **PUT** is used when **big changes** or **full update** is needed.
    - **PATCH** is used when **small correction** is needed.
5. **Example**:
    - **PUT**: Updating full user profile (name, email, age).
    - **PATCH**: Updating only user's email.
6. **Idempotency**:
    - Both **PUT** and **PATCH** are generally idempotent (sending the same request multiple times gives the same result).
7. **Real-world Example**:
    - **PUT**: Facebook me pura profile update karna.
    - **PATCH**: Sirf email ya profile photo change karna.

Database :-

1 What do you understand by term database ?
2 Define DBMS
3 Define RDBMS
4 What are various type of relationships in database? define them
5 Explain normalization

6 What are different types of normalization?
7 What is SQL ?
8 How many SQL statement used ? Define them
9 Enlist some commands of DDL , DML and DCL
10 Explain the term record, field and table in database
11 What is view in database
12 What are advantage and disadvantage of view?
13 Define Entity , Entity Type and Entity Set
14 Define a weak entity set
15 What do you undestand by join ? Explain different types of Join
16 What is self join
17 Explain Primary key, Foreign key, unique key, composite key, super key
18 What is projection in database?
19 What is difference between delete, truncate and drop command
20 Based on given table, solve the following query ?
21 **Employee table**

empId	empName	Age	Address
1001	Rohan	26	Delhi
1002	Ankit	30	Gurgaon
1003	Gaurav	27	Mumbai
1004	Raja	32	Nagpur

a) Write the SELECT command to display the details of the employee with empid as 1004.
b) Write the SELECT command to display all the records of table Employees.

c) Write the SELECT command to display all the records of the employee whose name starts with the character 'R'.

d) Write a SELECT command to display id, age and name of the employees with their age in both ascending and descending order.
e) Write the SELECT command to calculate the total amount of salary on each employee from the below Emp table.

22.What is mongodb ? What are advantage of mongodb
23.What is difference between SQL and NoSQL database
24.What is document in mongodb?
25.What is collection in mongodb
26.What is mongo shell
27.What are some features of mongodb
28.How to add data in mongodb ?

29.How do you update data in mongodb
30.How do you delete document
31.How to perform query in mongodb
32.What are datatypes in mongodb
33.What is index and how to create index in mongodb
34.Explain the set modifier in mongodb
35.Does mongodb support primary key and foreign key relationship ?
36.Explain the structure of ObjectId in mongodb

Ans :-ObjectID is a 12-byte BSON type. These are:

4 bytes value representing seconds

- 3 byte machine identifier
- 2 byte process id
- 3 byte counter

Is it true that MongoDB uses BSON to represent document structure?

37.What are Indexes in MongoDB?

In MondoDB, Indexes are used to execute query efficiently. Without indexes, MongoDB must perform a collection scan, i.e. scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.

38.By default, which index is created by MongoDB for every collection?

In-which language mongodb is writtern ?

39.What will happen when you remove a document from database in MongoDB? Does MongoDB remove it from disk?

40.What is capped and uncapped collection in mongodb
41.What is lookup in mongodb
42.How to get record of second max age record from mongodb collection
43.What is mongoose ?
44.What is mongoose Schmea
45.How to set reference in mongoose Schema
46.How populate works in mongoose ?
47.List the name of the mongoose method to perform CRUD operation
48.What is default connction pool size in mongoose
49.How to set connection pool size in mongoose

50.What is connection pool and what are advantage of using connection pool size

React

1.what is memoization and how does it improve performance ?
Ans-Memoization is a technique used to improve performance by **storing the result of a function** so we don't have to **recalculate it every time**.
For example, if I calculate the square of a number and the number hasn't changed, I can save the result and use it again.
In React, we use useMemo() for this. It remembers the result of a calculation between re-renders — which makes our app faster.
Also, for API calls, we can store (cache) the response. If the same request comes again, we give the saved result instead of hitting the server again.
So overall, memoization helps reduce repeated work, which makes our app faster and more efficient.

2.how is async await difference from promise

Ans-**Async/Await** is just a **cleaner and easier way** to write **Promises** in JavaScript.

🧠 **Promises:**

- Promises are used to handle asynchronous tasks like API calls.
- It uses .then() and .catch() for success and error handling.

Example-

```
fetchData()
  .then(response => console.log(response))
  .catch(error => console.log(error));
```

🧠 **Async/Await:**

- Introduced in ES8 (2017).
- It allows us to write asynchronous code like synchronous code.
- You use await inside a function marked with async.

Example-

```
async function getData() {
  try {
    const response = await fetchData();
```

```
    console.log(response);
  } catch (error) {
    console.log(error);
  }
}
```

✅ **Main Differences (Interview Friendly Points):**

Feature	Promise	Async/Await
Syntax	Uses .then() and .catch()	Uses async and await
Code readability	Can be nested and messy	Cleaner and looks like sync code
Error Handling	Handled with .catch()	Handled using try...catch
Learning Curve	Slightly harder for beginners	Easier to understand and maintain

---

🎤 **Example Interview Answer:**
Async/await is a modern way to work with promises in JavaScript.
Instead of chaining .then() methods, I can use await inside an async function to pause the code until a promise is resolved.
It makes my code cleaner, more readable, and easier to manage, especially when I have multiple async operations.

3.what are generator function and when would you see them ?
ans- **What is a Generator Function in JavaScript?**
A **generator function** is a special function in JavaScript that can **pause and resume** its execution.
It is declared using the function* syntax and uses the yield keyword to **pause** the function and return a value.

Why and When to Use Generator Functions?
Use generator functions when:

- You want to **pause the function midway** and resume later.
- You need to **generate a sequence of values** (like numbers or IDs).
- You want to **control execution flow** step by step.
- You're building **custom iterators** or handling **lazy evaluation** (generate values only when needed).

Example:

```
function* numberGenerator() {
  yield 1;
  yield 2;
  yield 3;
}
const numbers = numberGenerator();
console.log(numbers.next().value); // 1
console.log(numbers.next().value); // 2
console.log(numbers.next().value); // 3
```

Each next() call continues execution until it reaches the next yield.

🗣️ **Simple Interview Answer:**

A generator function is a special type of function in JavaScript that allows us to pause execution using yield and resume it later.
It is useful when we want to generate a sequence of values or need more control over the function's execution.
I would use it for things like custom iterators, lazy loading data, or generating infinite sequences.

4.How does javascript handle aoncurrency ?

Ans-JavaScript handles concurrency using its event loop and asynchronous features like callbacks, promises, and async/await.
Even though it runs on a single thread, asynchronous operations are handled by browser or Node APIs in the background, and the results are added to a queue.
The event loop then picks up those results when the main thread is free, allowing JavaScript to manage multiple tasks efficiently without blocking.

✅ **Simple Answer (Interview Style):**
JavaScript runs one thing at a time because it's single-threaded.
But it can still do many tasks without waiting — like reading files, calling APIs, or timers.
These tasks are sent to the browser or Node.js to finish in the background.
When they are done, the results go into a queue.
JavaScript checks the queue and runs the next task when it's ready.
This system is called the **event loop**, and it helps JavaScript do multiple things smoothly.

5. what is the temporal dead zone in js

Ans-In JavaScript, variables declared with let and const are **not available** before their declaration line.
If you try to use them too early, it's called being in the **Temporal Dead Zone**, and it will throw an error.
Example-

8) console.log(a); // ❌ ReferenceError: Cannot access 'a' before initialization
   let a = 10;
9) console.log(b); // ✅ undefined
   var b = 10;

6. what is the difference between shallow freeze and deep freeze

Ans-
Shallow Freeze:

- Object.freeze(obj) only **freezes the top-level properties** of the object.
- **Nested objects inside** are **still mutable (changeable)**.

Example-

```
const person = {
  name: "Amit",
  address: {
    city: "Delhi"
  }
};

Object.freeze(person);

person.name = "Ravi"; // ❌ Not allowed (frozen)
person.address.city = "Mumbai"; // ✅ Allowed (nested object is not frozen)

console.log(person);
// Output: { name: "Amit", address: { city: "Mumbai" } }
```

Deep Freeze:

- It **recursively freezes** the object and all **nested objects** inside it.
- Nothing can be changed at any level.

Example-

```
const user = {
  name: "Asha",
  details: {
    age: 30,
    address: {
      city: "Delhi"
    }
  }
};

deepFreeze(user);

user.details.age = 35; // ❌ Doesn't change
user.details.address.city = "Mumbai"; // ❌ Doesn't change

console.log(user);
// Output stays the same
```

8. **what is the difference between a synchronous function and an asynchronous function**
   ans-The difference between **synchronous** and **asynchronous** functions in JavaScript is **how they handle execution and waiting for tasks like API calls, file reading, etc.**

   Synchronous Function

- Executes **line by line**, one after another.
- **Blocks** the code execution until the current task is finished.
- If a task takes time (like file reading), it will make everything **wait**.

  Example-

```
function syncFunc() {
  console.log("Start");
  // Simulate a time-consuming task
  for(let i = 0; i < 1e9; i++) {}
  console.log("End");
}

syncFunc();
```

  Output:

```
Start
End
```

  Asynchronous Function

- Starts a task but **doesn't wait** for it to finish.
- Moves to the next line and comes back to the task when it's done.
- **Non-blocking**, uses mechanisms like **setTimeout**, **fetch**, **Promises**, or **async/await**.

  Example using setTimeout:

```
function asyncFunc() {
```

```
  console.log("Start");

  setTimeout(() => {
    console.log("Inside Timeout");
  }, 2000);

  console.log("End");
}

asyncFunc();
```

Output:

Start
End
Inside Timeout

9. **Explain the concept of IIFE.**

Ans-**IIFE** stands for **Immediately Invoked Function Expression**.
It's a function in JavaScript that runs **immediately after it is defined**.

Key Points:

- It's a **function expression**, not a declaration.
- It's **wrapped in parentheses** to turn it into an expression.
- It's **called immediately** with () at the end.

Example-1

```
(function () {
  let message = "Hello from IIFE!";
  console.log(message);
})();
```

Example-2

IIFE with parameters:

```
(function(name) {
  console.log("Hello " + name);
})("Rahul");
```

10. **How does setTimeOut work internallt in js.**

Ans-

What is setTimeout()?

setTimeout() is a **browser or Node.js API function** used to run a piece of code **after a certain delay** (in milliseconds), without blocking the rest of the code.

Internally, how does it work?

JavaScript is **single-threaded**, but it supports **asynchronous behavior** using something called the **Event Loop** and **Web APIs** (in the browser) or **libuv** (in Node.js).

Steps: How setTimeout() works

1. You call setTimeout() with a callback and delay.

```
setTimeout(() => {
```

```
  console.log("Executed after 2 seconds");
}, 2000);
```

2. The callback and delay are sent to the **Web API (in browser)** or **libuv (in Node)**. A **timer** starts counting **in the background**, not blocking the main thread.
3. After the timer finishes, the callback function is sent to the **callback queue (task queue)**.
4. The **Event Loop** checks if the main thread (call stack) is empty.
5. Once it's empty, it **pushes** the callback from the queue to the **call stack**, and it gets executed.

Example:

```
console.log("Start");

setTimeout(() => {
  console.log("Inside setTimeout");
}, 2000);

console.log("End");
```

Output:
Start
End
Inside setTimeout
Because the timeout runs **asynchronously**, "Inside setTimeout" runs **after 2 seconds**, even though it comes earlier in the code.

11. What is a Service Worker?
Ans-A **Service Worker** is a special JavaScript file that runs **in the background** of your website. It helps your website work even when there is **no internet**.

How It Works Without Internet?
When you first open a website **with internet**, the **Service Worker** runs in the background and:

1. **Saves (caches) important files** like HTML, CSS, JS, and images.
2. These saved files are stored in your **browser's storage** (called **Cache Storage**).
3. Next time, even **without internet**, the Service Worker gives you the saved (cached) files.
4. So the website opens using the **local saved data**, not from the internet.

12.What is Optional chaing in js
Ans-Optional chaining (?.) is a feature in JavaScript that lets you safely access deeply nested object properties without getting an error if a property doesn't exist.

Example-

```
let user = {}; // No profile property

console.log(user.profile.name); // ❌ ERROR: Cannot read 'name' of undefined
```

Optional chaining says: "If the thing exists, go ahead. If not, just give me undefined — don't crash."

Fix using Optional Chaining:

```
console.log(user?.profile?.name); // ✅ Outputs undefined, no error
```