**BACKEND WITH NODE JS**

**COMMANDS FOR TERMINAL OPARATIONS**

* **FILES:** 
  + **CREATE FILES USING COMMANDS: touch file name**
  + **UPDATE THE NAME OF THE FILE: mv old file name new file name**
  + **DELETE THE FILE: rm file name**
* **FOLDERS:**
  + **CREATE FILES USING COMMANDS: mkdir folder name**
  + **UPDATE THE NAME OF THE FILE: mv old folder name new folder name**
  + **DELETE THE FILE: rmdir folder name**
* **DIRECTORY:**
  + **GET IN: cd/path/path [TAB KEY]**
  + **GET OUT: cd “-“**

**Node JS Questions**

**Q0. Node JS vs JavaScript:**

**JavaScript**

1. **Js used in Frontends**
2. **Js cannot connect to DataBase.**

**And access top Browser APIs and DOM.**

1. **Js is a Programming language and its work directory is on the Clint Side.**
2. **Js Deployed on Web Browsers.**

**Node JS**

1. **Node js used in Backend Programming.**
2. **Node can work with APIs and DataBase.**
3. **Node js is a server site Environment & and uses v8 Engine to run.**
4. **Node JS Deployed on Servers.**

**Q1.** 🔗 **Modules & Types:** In Node.js, modules are a way to organize and structure code by encapsulating related functionality into separate files. Each file in Node.js is treated as a separate module and can export specific parts of the code from one module to be used in another. This helps in creating maintainable and scalable applications.

**There are two types of modules in Node.js:**

1. **Core/ Global Modules**: These are built-in modules that come with Node.js. we can use them by simply requiring them in your code. Examples include **fs** (File System), **HTTP** (HTTP server), and **path** (File and directory path utilities).
2. 🔗 **User-Defined/Local Modules**: These are modules created by the developer to organize their code. we create a module by writing our code in a separate file and then using the **module.exports** or **exports** object to expose functions, variables, or objects for use in other files.

**example of user-defined-Modules:**

🔗 **// index.js**

**// Requiring the custom module**

**const exampleModule = require('./exampleModule');**

**// Using functions from the custom module**

**const sum = exampleModule.add(5, 3);**

**const product = exampleModule.multiply(4, 6);**

**console.log('Sum:', sum); // Output: Sum: 8**

**console.log('Product:', product); // Output: Product: 24**

🔗 **//exampleModule.js**

**// Function to add two numbers**

**const add = (a, b) => {**

**return a + b;**

**};**

**// Function to multiply two numbers**

**const multiply = (a, b) => {**

**return a \* b;**

**};**

**// Exposing functions for external use**

**module.exports = {**

**add,**

**multiply**

**};**

**Q2.** 🔗 **What HTTP does in node js:** In Node.js, the **HTTP** module is one of the core modules that allows you to **create an HTTP server and handle HTTP requests and responses**. It provides a foundation for building web applications and APIs.

**Q3- Package.json  
  
1. What is package.json:** is a **metadata file** used in Node.js projects. It serves as a configuration file and plays a central role in managing a Node.js application.

* **Metadata file for Node.js projects.**
* **Manages dependencies, configuration, and scripts.**

**2.** 🔗 **How to create package.json:**

* **npm init [manually] or npm init -y [automatic] [ in the project directory]**

**3. How to add packages (dependencies):**

* **Use npm install [ package-name ]**

**4. How to uninstall packages:**

* **npm uninstall [ package-name]**

**5.** 🔗 **Example of package.json contents:**

* **Name:** Name of the project.
* **Version:** Project version.
* **Description:** Brief project description.
* **Main:** Entry point (usually index.js/main.js/server.js).
* **Scripts:** Custom commands (npm run).
* **Dependencies:** Production dependencies.
* **DevDependencies:** Development dependencies.
* **Keywords:** Project-related keywords.
* **Author:** Project author.
* **License:** License type.

**Q.6 How not to push modules in GitHub:** we need to have a file called .**gitignore** and mention the name such as **“node\_module & .env”**

**Q5. What to do if all node modules were deleted mistakenly:** it might show 2 types of errors one module not found in pakaeges.json or reference error functions are not defined  
we have to go to the right directory in the terminal and write the command “**npm I or npm install**” It will download all modules mentioned in the packages.

**Q4. Node js single-threaded or multi-threaded:** Node.js is generally considered **single-threaded**. It uses an **event-driven,** **non-blocking I/O(input/output)** model that allows it to handle many simultaneous connections without the need for multi-threading. The single thread, often referred to **as the event loop, efficiently manages asynchronous operations.**

In summary, while Node.js itself is primarily single-threaded, it leverages asynchronous I/O and an event-driven model to achieve high concurrency and responsiveness.

**Q7. How to run server continuously in local server:** we have to install an npm called “nodemon” by using the command **“npm I nodemon -g”** -g to have the npm in local device globally and to run it we have to use **“nodemon .index.js”**

**Q8. Which function should we use synchronous or asynchronous**: in node js using the **asynchronous function gives us an upper hand**, the **asynchronous function** won’t wait to complete a function and execute the next command after executing a command so the flow and the speed always maintain, by using allowing **the event loop** to continue processing other tasks,  
 on the other hand **Synchronous Function** Executes tasks sequentially, **blocking the event loop** until the function completes.  
**example =>**

**Asynchronous function Example**

**function asynchronousFunction() {  
console.log("Asynchronous Function: Start");**// Simulate an asynchronous task using setTimeout

**setTimeout(()** **=>** **{**

**console.log("Synchronous Function:Task Complete");**

**}**, 2000**)**;  
 **console.log("Synchronous Function:** **Continue Execution");**

**}**  
**asynchronousFunction();**

**Synchronous Function Example**

**function synchronousFunction() {**

**console.log("Synchronous Function:Start");**

// Simulate a time-consuming task

for (let i = 0; i < 5e7; i++) **{**

// Do nothing, just consume time  
**}**  
**console.log("Synchronous Function: End")**;

**}  
synchronousFunction();**

**Q9.** 🔗 **How to take Data from the terminal and, CURD Operations:** In node js we have an object called PROCESS in the process we have argv(arguments variable) to access that we have to write process.argv and initialize the variable, in arge array index, 0&1 are already reserved so we have to write process.argv[2]

* **Create a File:**
  + Import the **'fs**' module using const **fs** = **require('fs')**;.
  + Access command line arguments using **process.argv.**
  + Use **fs.writeFileSync(fileName, 'data')**; to create file.
* **Add Data to the File:**
  + Use **fs.appendFileSync(fileName, “dataToAdd”+ '\n')**; to add data to the file.
  + **\n** is used to add data in a new line.
* **Delete File:**
  + Utilize **fs.unlinkSync(fileName);** to delete the file synchronously.
* **Read File:**
  + Read file content synchronously using **fs.readFileSync(fileName, 'utf-8').**
  + For asynchronous reading, use **fs.readFile(fileName, 'utf-8', callback).**

**Note: Always handle errors appropriately, especially in asynchronous operations**

**Q10. UTF-8 in Node.js:** **Character Encoding: UTF-8** is a variable-width character encoding that can represent every character in the Unicode character set. It is the default character encoding for JavaScript strings in Node.js.

**Default Encoding:** When **reading or writing text data** in Node.js, if no specific encoding is provided, UTF-8 is usually assumed. For example, when using readFileSync or writeFileSync without specifying an encoding, UTF-8 is used by default.

**Q12.** 🔗 **What is Promise in JS Example:** Promises are a fundamental concept in JavaScript that provides a way to work with **asynchronous operations more effectively**. A Promise **represents a value that may be available now, or in the future, or never**. It allows you to handle asynchronous operations in a more structured and readable manner compared to traditional callback patterns.   
**1. A Promise can be in one of three states:**

* **Pending:** The initial state; the promise is neither fulfilled nor rejected.
* **Fulfilled:** The operation was completed successfully, and the promise has a resulting value.
* **Rejected:** The operation failed, and the promise has a reason for the failure.

**2. How to Handel a Promise:**

* **The then method**: is used to handle the fulfillment of the Promise. If the operation is successful, the result parameter in the then block contains the resolved value.
* **The catch method**: is used to handle the rejection of the Promise. If the operation fails, the error parameter in the catch block contains the reason for the rejection.

🔗 EXAMPLE CODE LINK:

**Q11.** 🔗 **Buffer in Node JS:** In Node.js, **a buffer is a temporary storage space for binary data**. It is essentially a **raw memory allocation** that can be used **to store binary data such as images, files, and network responses.**

**Key characteristics of buffers in Node.js:**

* **Binary Data Storage:** Buffers are designed to handle binary data directly. They can store data in a way that doesn't require any encoding or interpretation, making them suitable for working with files, network protocols, and other raw data sources.
* **Fixed Size:** Buffers have a fixed size, set when they are created. Once a buffer is allocated, its size cannot be changed.
* **Efficient Manipulation:** Buffers provide methods to efficiently manipulate binary data, including methods for copying, slicing, and modifying the data.

🔗 **Example:**

* **// Creating a buffer with a size of 10 bytes**

**const buffer = Buffer.alloc(10);**

* **// Writing data into the buffer**

**buffer.write("Hello");**

* **// Reading from the buffer**

**console.log(buffer.toString()); // Outputs: Hello**

**Q13.** 🔗 **How to handle async function in node js:**

**Asynchronous Function Example**

*// Example 1: Using a simple asynchronous function with setTimeout*

*// Global variables*

let a = 10;

let b = 10;

*// Asynchronous function*

const **result** = () => {

  setTimeout(() => {

    b = 20; *// Modifying the global variable* ***'b' after a 2-second delay***

  }, 2000);

};

result(); *// Calling the asynchronous function*

console.log("a+b", a + b); *// This will log the initial value of a+b, not considering the asynchronous modification* ***and print 20.***

**NOTE: we don’t have to write promise every time to fetch because in node js promises already there in build**

**Handling Asynchronous operations using Promises**

*// Global variables*

let x = 10;

let y = 10;

*// Asynchronous operation wrapped in a Promise*

const **asyncResult** = new Promise((resolve, reject) => {

  setTimeout(() => {

    y = 20; *// Modifying the global variable* ***'y' after a 2-second delay***

    resolve(y); *// Resolving the Promise with the updated value of 'y'*

  }, 2000);

});

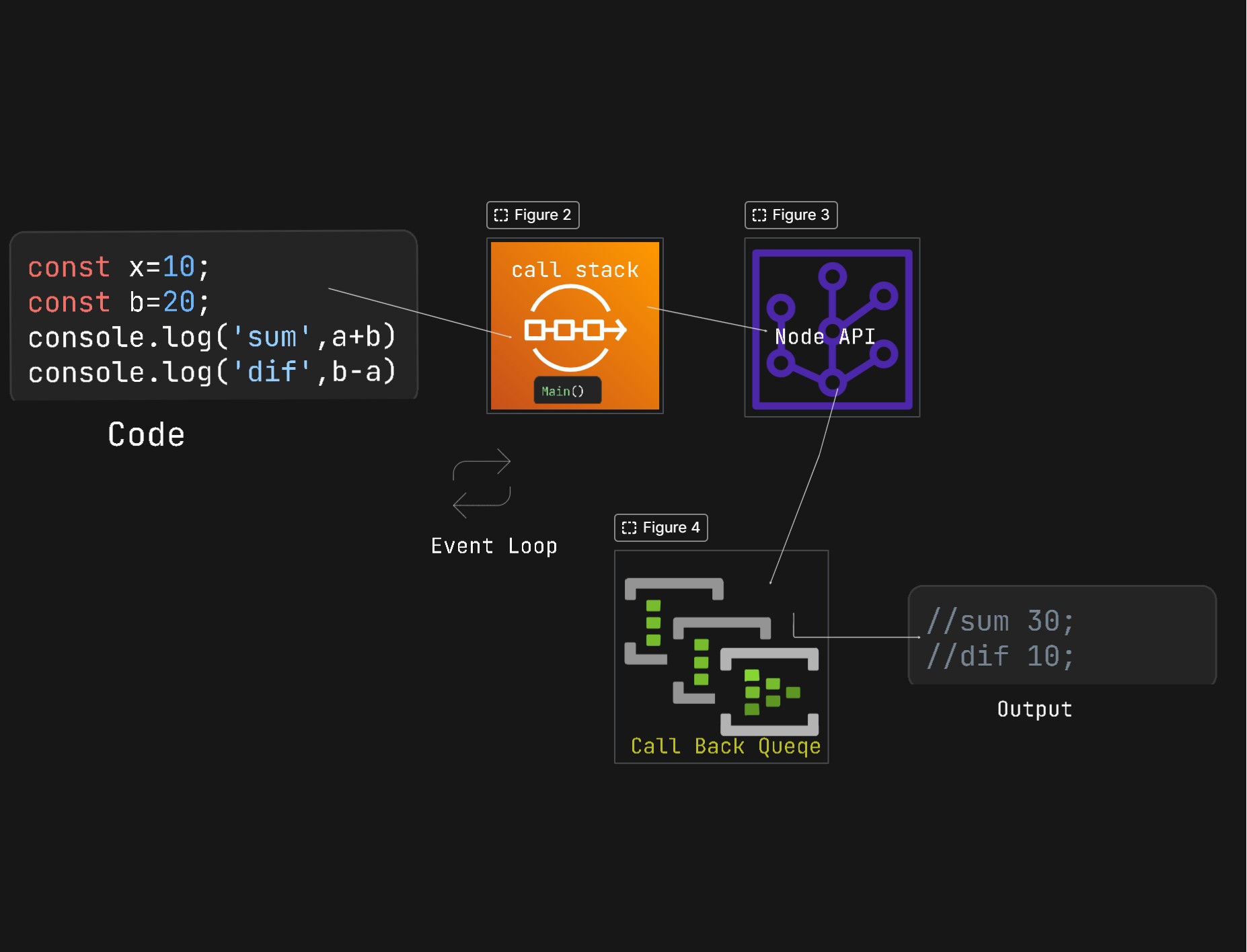
*// Using Promise's 'then' method to handle the resolved value*

ancResult.then((newValuey) => {

  console.log("x+y", x + newValuey); *// Logging the result of x + newValuey* ***after the asynchronous operation is complete and print 30.***

});

**Q14. What is Event Loop:**



**EVENT LOOP WORKING IMAGE**

**Main()**

**Q.15 What is JSON.parse() & JSON.stringify and its use**: In JavaScript, JSON.parse and JSON.stringify are functions used **for converting between JSON (JavaScript Object Notation) strings and JavaScript objects**. They serve the following purposes:

* **JSON.stringify(Adding Data): Purpose: Converts a JavaScript object to a JSON string.**
  + **Use Case:** When we want to send data from a client (written in JavaScript) to a server or when we need to store data in a file in a JSON format.
* **JSON.parse(Reading Data): Purpose:** Parses a JSON string and converts it into a JavaScript object.
  + **Use Case:** When we receive JSON data from a server or read a JSON file and want to work with the data in your JavaScript application.

**Example 2**

**JSON.parse(): While Reading**

**const** jsonString ='{"name":"John","age":25};

**const** obj = *JSON*.parse(jsonString);

**console.log**(obj);

*// { name: "John", age: 25 }*

**Example 1**

**JSON.stringify():While Writing**

**const** obj = **{ name: "John", age: 25 };**

**const** jsonString = *JSON*.stringify(obj);

**console.log**(jsonString);

*// '{"name": "John", "age":25}'*

// Common HTTP Status Codes

200 OK - The request was successful.

201 Created - The request has been fulfilled, resulting in the creation of a new resource.

204 No Content - The server successfully processed the request, but there is no additional content to send.

400 Bad Request - The server could not understand the request due to invalid syntax.

401 Unauthorized - Authentication is required and has failed or has not been provided.

403 Forbidden - The server understood the request but refuses to authorize it.

404 Not Found - The requested resource could not be found on the server.

405 Method Not Allowed - The method specified in the request is not allowed for the resource.

500 Internal Server Error - A generic error message returned when an unexpected condition was encountered.

// Informational Codes (1xx)

100 Continue

101 Switching Protocols

102 Processing

103 Early Hints

// Successful Codes (2xx)

200 OK

201 Created

202 Accepted

203 Non-Authoritative Information

204 No Content

205 Reset Content

206 Partial Content

207 Multi-Status

208 Already Reported

226 IM Used

// Redirection Codes (3xx)

300 Multiple Choices

301 Moved Permanently

302 Found

303 See Other

304 Not Modified

305 Use Proxy

307 Temporary Redirect

308 Permanent Redirect

// Client Error Codes (4xx)

400 Bad Request

401 Unauthorized

402 Payment Required

403 Forbidden

404 Not Found

405 Method Not Allowed

406 Not Acceptable

407 Proxy Authentication Required

408 Request Timeout

409 Conflict

410 Gone

411 Length Required

412 Precondition Failed

413 Payload Too Large

414 URI Too Long

415 Unsupported Media Type

416 Range Not Satisfiable

417 Expectation Failed

418 I'm a teapot

421 Misdirected Request

422 Unprocessable Entity

423 Locked

424 Failed Dependency

425 Too Early

426 Upgrade Required

428 Precondition Required

429 Too Many Requests

431 Request Header Fields Too Large

451 Unavailable For Legal Reasons

// Server Error Codes (5xx)

500 Internal Server Error

501 Not Implemented

502 Bad Gateway

503 Service Unavailable

504 Gateway Timeout

505 HTTP Version Not Supported

506 Variant Also Negotiates

507 Insufficient Storage

508 Loop Detected

510 Not Extended

511 Network Authentication Required