NODE AND EXPRESS QUESTIONS

1. What is NodeJS?

- Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser.
- Node JS was created by Ryan Dahl.

2. What Are the Key Features of NodeJs?

- **Asynchronous Event-Driven IO:** Node.js processes requests asynchronously, allowing it to handle concurrent requests efficiently without waiting for responses.

- **Fast Code Execution:** Leveraging the V8 JavaScript Runtime engine, Node.js executes code swiftly, aided by a wrapper over the JavaScript engine.

- **Single-Threaded yet Highly Scalable:** Despite its single-threaded event loop model, Node.js remains highly scalable, as it doesn't block other operations while processing events.

- **Large Community:** With a vast community of JavaScript developers, Node.js benefits from familiarity and ongoing updates, ensuring it stays aligned with the latest web development trends.

- **No Buffering:** Node.js applications avoid buffering data, instead outputting it in chunks, which can enhance performance and efficiency.

3. Explain REPL In NodeJs?

The REPL stands for "**Read Eval Print Loop**". It is a simple program that accepts the commands,

- **READ** - It Reads the input from the user, parses it into JavaScript data structure and then stores it in the memory.
- **EVAL** - It Executes the data structure.
- **PRINT** - It Prints the result obtained after evaluating the command.
- **LOOP** - It Loops the above command until the user presses Ctrl+C two times.

4. Is Node a single threaded application?

- Yes! Node uses a single threaded model with event looping.

5. What is package json?

- The package.json file in Node.js is the heart of the entire application.
- It is basically the manifest file that contains the metadata of the project where we define the properties of a package.

6. What is a module in Node.js?

- In Node.js Application, a Module can be considered as a block of code that provide a simple or complex functionality that can communicate with external application.
- Modules can be organized in a single file or a collection of multiple files/folders. Modules are useful because of their reusability and ability to reduce the complexity of code into smaller pieces.
- Some examples of modules are. http, fs, os, path, etc.

7. Explain the purpose of module exports?

- A module in Node.js is used to encapsulate all the related codes into a single unit of code which can be interpreted by shifting all related functions into a single file

8. What is middleware?

- Middleware comes in between your request and business logic.
- It is mainly used to capture logs and enable rate limit, routing, authentication, basically whatever that is not a part of business logic.

9. Name the types of API functions in Node?

There are two types of functions in Node.js.

- **Blocking functions -** In a blocking operation, all other code is blocked from executing until an I/O event that is being waited on occurs. Blocking functions execute synchronously.
- **Non-blocking functions -** In a non-blocking operation, multiple I/O calls can be performed without the execution of the program being halted. Non-blocking functions execute asynchronously.

10. Which are the areas where it is suitable to use NodeJS?

- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real-time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications

11. Which are the areas where it is not suitable to use NodeJS?

- it’s not suitable for heavy applications involving more of CPU usage.

12. What Is EventEmitter In NodeJs?

- Events module in Node.js allows us to create and handle custom events.
- The Event module contains “EventEmitter” class which can be used to raise and handle custom events.
- EventEmitter provides multiple properties like “on” and “emit”.
- The “on” property is used to bind a function to the event and “emit” is used to fire an event.

13. What do you mean by Asynchronous API?

- All APIs of Node.js library are aynchronous that is non-blocking.
- It essentially means a Node.js based server never waits for a API to return data.
- Server moves to next API after calling it and a notification mechanism of Events of Node.js helps server to get response from the previous API call.

14. What is global installation of dependencies?

- Globally installed packages/dependencies are stored in <user-directory>/npm directory.
- Such dependencies can be used in CLI (Command Line Interface) function of any node.js but cannot be imported using require() in Node application directly.
- To install a Node project globally use -g flag.
- **npm install nodemon -g**

15. What is local installation of dependencies?

- By default, npm installs any dependency in the local mode.
- Here local mode refers to the package installation in node_modules directory lying in the folder where Node application is present.
- Locally deployed packages are accessible via require().
- Syntax ➔ npm install express

16. How to check the already installed dependencies which are globally installed using npm?

- ***npm ls -g***

17. What is Libuv ?

- Libuv is a multi-platform C library that provides support for asynchronous I/O based on event loops.
- It is used by Node.js to achieve its non-blocking, event-driven architecture.
- It was initially developed for Node.js to handle the event-driven architecture, but it has since been adopted by other projects due to its performance and efficiency.

18. How to uninstall a dependency using npm?

- npm uninstall dependency-name

19. How to update a dependency using npm?

- ***npm update***

20. What are buffer objects in nodejs?

- In Node.js, Buffer objects are used to represent binary data in the form of a sequence of bytes.
- Many Node.js APIs, for example streams and file system operations, support Buffers, as interactions with the operating system or other processes generally always happen in terms of binary data

21. What is a blocking code?

- If application has to wait for some I/O operation in order to complete its execution any further then the code responsible for waiting is known as blocking code.

22. How Node prevents blocking code?

- By providing callback function. Callback function gets called whenever corresponding event triggered.

23. What is Event Loop?

- The event loop in Node.js is a mechanism that allows it to handle multiple asynchronous tasks concurrently within a single thread. It continuously listens for events and executes associated callback functions.
- Node js is a single threaded application but it support concurrency via concept of event and callbacks.
- Node thread keeps an event loop and whenever any task get completed, it fires the corresponding event which signals the event listener function to get executed.

24. What is Piping in Node?

Piping is a mechanism to connect output of one stream to another stream. It is normally used to get data from one stream and to pass output of that stream to another stream. There is no limit on piping operations.

25. What are streams?

Streams are objects that let you read data from a source or write data to a destination in continuous fashion.

26. How many types of streams are present in Node.

In Node.js, there are four types of streams.

- **Readable** − Stream which is used for read operation.
- **Writable** − Stream which is used for write operation.
- **Duplex** − Stream which can be used for both read and write operation.
- **Transform** − A type of duplex stream where the output is computed based on input.

27. Name some of the events fired by streams.

Each type of Stream is an Event Emitter instance and throws several events at different instance of times. For example, some of the commonly used events are:

- **data** − This event is fired when there is data is available to read.
- **end** − This event is fired when there is no more data to read.
- **error** − This event is fired when there is any error receiving or writing data.
- **finish** − This event is fired when all data has been flushed to underlying system.

28. Name some of the attributes of package.json?

Following are the attributes of Package.json

name , version ,description ,author ,dependencies ,repository ,main, keywords

29. What is callback hell?

- Callback hell is an issue caused due to a nested callback.
- This causes the code to look like a pyramid and makes it unable to read To overcome this situation we use promises.

30. How can you avoid callback hells?

There are lots of ways to solve the issue of callback hells:

- modularization: break callbacks into independent functions,
- use a control flow library, like async.
- use generators with Promises,
- use async/await (note that it is only available in the latest v7 release and not in the LTS version

31. Explain the usage of a buffer class in Nodejs?

- Buffer class in Node.js is used for storing the raw data in a similar manner of an array of integers.
- But it corresponds to a raw memory allocation that is located outside the V8 heap.
- It is a global class that is easily accessible can be accessed in an application without importing a buffer module.
- Buffer class is used because pure JavaScript is not compatible with binary data. So, when dealing with TCP streams or the file system, it's necessary to handle octet streams.

32. Explain chaining in Nodejs?

- Chaining is a mechanism whereby the output of one stream is connected to another stream creating a chain of multiple stream operations.

33. List down the major security implementations within Nodejs?

- Major security implementations in Node.js are: Authentications, Error Handling

34. Explain the concept of URL module?

- The URL module splits up a web address into readable parts

35. Describe the exit codes of Nodejs?

In Node.js, exit codes are a set of specific codes which are used for finishing a specific process.

These processes can include the global object as well.

Below are some of the exit codes used in Node.js:

- Uncaught fatal exception
- Unused
- Fatal Error
- Internal Exception handler Run-time failure
- Internal JavaScript Evaluation Failure

36. Is cryptography supported in Nodejs?

- Yes, Node.js does support cryptography through a module called Crypto. This module provides various cryptographic functionalities like cipher, decipher, sign and verify functions etc.
  - ⇨ ***const crypto = require('crypto');***
- The crypto module is used for encrypting, decrypting, or hashing any type of data.
- This encryption and decryption basically help to secure and add a layer of authentication to the data.
- The main use case of the crypto module is to convert the plain readable text to an encrypted format and decrypt it when required.

37. What are the different ways of implementing Addons in NodeJS?

There are three options for implementing Addons:

- N-API
- nan direct use of internal V8
- libuv
- Node.js libraries

38. Which are the different console methods available?

There are around 21 inbuilt console methods, we can also build our own prototypes using new Console constructor function here are a few popular one's

- ***console.clear()*** will clear only the output in the current terminal viewport for the Node.js
- ***console.error([data][, ...args])*** Prints to stderr with newline. Multiple arguments can be passed, with the first used as the primary message and all additional used as substitution
- ***console.table(tabularData[, properties])*** a table with the columns of the properties of tabularData (or use properties) and rows of tabularData and log it.

39. What is the datatype of console?

- The datatype of console is an object

40. What are the utilities of OS module in NodeJS?

The os module provides operating system-related utility methods and properties. It can be accessed using:

⇨ ***const os = require('os');***

41. Which are the global objects in Node JS?

- __dirname , __filename, console, exports, global, module
- clearImmediate(immediateObject), clearInterval(intervalObject)
- setTimeout(callback, delay[, ...args]), clearTimeout(timeoutObject)
- process, queueMicrotask(callback), require()
- setImmediate(callback[, ...args]), setInterval(callback, delay[, ...args])
- TextDecoder, TextEncoder
- URL, URLSearchParams, WebAssembly

42. What is Callback?

- A JavaScript callback is a function which is to be executed after another function has finished execution.
- A more formal definition would be - Any function that is passed as an argument to another function so that it can be executed in that other function is called as a callback function.

43. How do you manage packages in your node.js project?

- It can be managed by a number of package installers and their configuration file mostly use npm or yarn.
- Both provide almost all libraries of javascript with extended features of controlling environment-specific configurations.
- To maintain versions of libs being installed in a project we use package.json and package-lock.json so that there is no issue in porting that app to a different environment.

44. How is Node.js better than other frameworks most popularly used?

- Node.js provides simplicity in development because of its non-blocking I/O and even-based model results in short response time and concurrent processing, unlike other frameworks where developers have to use thread management.
- It runs on a chrome v8 engine which is written in c++ and is highly performant with constant improvement.
- Also since we will use Javascript in both the frontend and backend the development will be much faster.
- And at last, there are ample libraries so that we don’t need to reinvent the wheel.

45. What is control flow in Node.js?

- Control flow in Node.js refers to the sequence in which statements and functions are executed. It manages the order of execution, handling asynchronous operations, callbacks, and error handling to ensure smooth program flow.

46. Explain the purpose of ExpressJS package?

- Express.js is a framework built on top of Node.js that facilitates the management of the flow of data between server and routes in the server-side applications.
- It is a lightweight and flexible framework that provides a wide range of features required for the web as well as mobile application development.
- Express.js is developed on the middleware module of Node.js called connect.
- The connect module further makes use of http module to communicate with Node.js. Thus, if you are working with any of the connect based middleware modules, then you can easily integrate with Express.js.

47. What is the order in which control flow statements get executed?

The order in which the statements are executed is as follows:

I. Execution and queue handling
II. Collection of data and storing it
III. Handling concurrency
IV. Executing the next lines of code

48. Difference between setImmediate() and process.nextTick() methods

- The ***process.nextTick()*** method is used to add a new callback function at the start of the next event queue. it is called before the event is processed.
- The ***setImmediate*** is called at the check phase of the next event queue. It is created in the poll phase and is invoked during the check phase.

49. What are some commonly used timing features of Node.js?

- **setTimeout/clearTimeout –** This is used to implement delays in code execution.
- **setInterval/clearInterval –** This is used to run a code block multiple times.
- **setImmediate/clearImmediate** – This is used to set the execution of the code at the end of the event loop cycle.
- **process.nextTick** – This is used to set the execution of code at the beginning of the next event loop cycle.

50. What is the difference between setTimeout() and setImmediate() method?

- The **setImmediate** function is used to execute a particular script immediately whereas the **setTimeout** function is used to hold a function and execute it after a specified period of time.

51. What are the advantages of using promises instead of callbacks?

- The main advantage of using promise is you get an object to decide the action that needs to be taken after the async task completes. This gives more manageable code and avoids callback hell.

52. How does Node.js overcome the problem of blocking of I/O operations?

- Since the node has an event loop that can be used to handle all the I/O operations in an asynchronous manner without blocking the main function.
- even though we have single-threaded JS, I/O ops are handled in a nonblocking way.

53. How does Express.js handle routing?

- Express.js has a simple routing system that allows developers to define and manage application routes.
- Routes can be determined using the app.get() and app.post() methods.
- It can be associated with callback functions executed when a request is made to the route.

54. How does Express.js handle middleware?

- Express.js has a middleware system that allows developers to define and manage middleware functions.
- These functions can perform tasks such as authentication, validation, or modification of request and response objects.
- Middleware functions are executed in the order they are defined. They can be added to the application using the app.use() method.

55. How does Express.js handle request and response objects?

- Express.js has a request and response object system that provides access to information about incoming requests and outgoing responses.
- The request object contains information about the incoming request, such as the URL, method, and headers.
- The response object is used to send a response back to the client.
- Developers can use methods such as ***res.send(), res.json(),*** and ***res.render()*** to send responses to the client.

56. What is the difference between a traditional server and an Express.js server?

- A traditional server is a standalone server that is built and managed independently.
- While an Express.js server is built using the Express.js framework.
- It runs on top of Node.js. Express.js provides a simple and efficient way to create and manage web applications.
- It offers a wide range of features and tools for handling routing, middleware, and request or response objects.

57. How does Express.js handle error handling?

- Express.js provides a built-in error-handling mechanism using the next() function.
- When an error occurs, you can pass it to the next middleware or route handler using the next() function.
- You can also add an error-handling middleware to your application that will be executed whenever an error occurs.

58. How does Express.js handle file uploads?

- Express.js provides support for file uploads through middleware functions and the request object.
- Developers can use middleware functions like multer or busboy to handle file uploads.
- It can access the uploaded files through the request object.

59. Mention some of the databases with which Express JS is integrated.

- MySQL, MongoDB, PostgreSQL, SQLite, Oracle

60. How does Express.js differ from other Node.js frameworks?

- Express.js is a flexible framework that provides only the essential features required for web application development.
- On the other hand, other Node.js frameworks, such as ***Meteor, Sails.js,*** and ***Koa.js***, offer more features but may not be required for smaller projects.
- Express.js is a good choice for simple, fast, and scalable web applications.

61. What is the difference between res.send() and res.json() in Express.js?

- res.send() is used to send a response with any type of data (string, object, buffer, etc.).
- While res.json() is used to send a JSON response. res.json() also sets the Content-Type header to application or JSON.

62. What is the use of app.use() in Express.js?

- app.use() is used to add middleware functions to an Express application.
- It can be used to add global middleware functions or to add middleware functions to specific routes.

63. What is the purpose of the next() function in Express.js?

- The next() function is used to pass control from one middleware function to the next function.
- It is used to execute the next middleware function in the chain.

64. What is the purpose of the req.params object in Express.js?

- The req.params object is used to access route parameters in Express.js.
- Route parameters capture values from the URL and pass them to the request handler.

65. What is the difference between req.query and req.params in Express.js?

- req.query is used to access the query parameters in a URL. While req.params is used to access route parameters in a URL.

66. What is npm and its advantages?

- npm (Node Package Manager) is the default package manager for Node.js.
- It allows developers to discover, share, and reuse code packages easily.
- Its advantages include dependency management, version control, centralized repository, and seamless integration with Node.js projects.

67. How to import a module in Node.js?

- We use the require module to import the External libraries in Node.js.
- The result returned by require() is stored in a variable which is used to invoke the functions using the dot notation.

68. What is the difference between Node.js and AJAX?

- Node.js is a JavaScript runtime environment that runs on the server side
- whereas AJAX is a client-side programming language that runs on the browser.

69. What is body-parser in Node.js?

- Body-parser is the Node.js body-parsing middleware.
- It is responsible for parsing the incoming request bodies in a middleware before you handle it.
- It is an NPM module that processes data sent in HTTP requests.

70. How to write hello world using node.js?

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

// Create a server object
http.createServer(function (req, res) {
    res.write('Hello World!');
    res.end();
}).listen(3000);
```

71. What is fork in Node.js?

- Fork is a method in Node.js that is used to create child processes.
- It helps to handle the increasing workload.
- It creates a new instance of the engine which enables multiple processes to run the code.

72. What is the difference between spawn() and fork() method?

- Both these methods are used to create new child processes the only difference between them is that ➔
    - ***spawn()*** method creates a new function that Node runs from the command line
    - whereas ***fork()*** function creates an instance of the existing fork() method and creates multiple workers to perform on the same task.

73. What is CORS in Node.js?

- The word CORS stands for “Cross-Origin Resource Sharing”.
- Cross-Origin Resource Sharing is an HTTP-header based mechanism implemented by the browser which allows a server or an API to indicate any origins (different in terms of protocol, hostname, or port) other than its origin from which the unknown origin gets permission to access and load resources.
- The cors package available in the npm registry is used to tackle CORS errors in a Node.js application.

74. What is the difference between Node.js and JavaScript?

Feature	Node.js	JavaScript
Type of Language	Server-side runtime environment	Client-side scripting language
Primary Use	Allows running JavaScript code on server	Primarily used for web development
Underlying Engine	Built on Chrome’s V8 JavaScript engine	Runs in a web browser’s JavaScript engine
Application Scope	Enables building scalable network applications	Executes code within a browser environment
Access to Resources	Provides access to file system and network resources	Limited to browser APIs and capabilities
Handling I/O Operations	Supports event-driven, non-blocking I/O operations	Executes in a single-threaded event loop
Common Use Cases	Used for building backend APIs, servers, and applications	Utilized for creating interactive web pages and client-side logic

75. What is the difference between Express.js and Node.js?

Feature	Express.js	Node.js
Type	Framework	Runtime environment
Language	JavaScript	JavaScript
Purpose	To build web applications	To run JavaScript code outside of a browser
Features	Routing, middleware, template engines, etc.	Event-driven, non-blocking I/O, etc.
Dependencies	Node.js	N/A
Popularity	Very popular	Very popular
Examples	GitHub, Uber, Netflix	PayPal, LinkedIn, Walmart

76. What is the difference between Synchronous and Asynchronous functions?

Feature	Synchronous Functions	Asynchronous Functions
Execution Blocking	Blocks the execution until the task completes	Does not block the execution; allows other tasks to proceed concurrently
Waiting for Completion	Executes tasks sequentially; each task must complete before the next one starts	Initiates tasks and proceeds with other operations while waiting for completion
Return Value	Returns the result immediately after completion	Typically returns a promise, callback, or uses event handling to handle the result upon completion
Error Handling	Errors can be easily caught with try-catch blocks	Error handling is more complex and often involves callbacks, promises, or async/await syntax
Usage Scenario	Suitable for simple, sequential tasks with predictable execution flow	Ideal for I/O-bound operations, network requests, and tasks requiring parallel processing