**1.**[**What is Node.js?**](https://www.geeksforgeeks.org/introduction-to-node-js/)

[Node.js](https://www.geeksforgeeks.org/node-js-introduction/) is an [open-source](https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs), cross-platform [JavaScript](https://www.simplilearn.com/tutorials/javascript-tutorial/introduction-to-javascript) runtime environment to executing JavaScript code outside the browser.

It is normally used to build the backend/server-side web application and is highly scalable.

It is built on **Chrome’s V8 JavaScript engine** for executing JavaScript code outside of a browser.

It provides an event-driven, non-blocking (asynchronous) I/O, and cross-platform runtime environment for building highly scalable server-side applications using JavaScript.

**What is Run Time Environment**

It is an environment where your program will be executed.

It act as small operating systems and provide all the functionality necessary for a program to run.

Some functionality are I/O operations, memory management etc

**2.**[**What is the difference between Node.js and JavaScript?**](https://www.geeksforgeeks.org/difference-between-node-js-and-javascript/)

JavaScript is a scripting language whereas Node.js is an engine that provides the runtime environment to run JavaScript code.

Here we have [difference table between Node.js and JavaScript](https://www.geeksforgeeks.org/difference-between-node-js-and-javascript/)

| [Node.js](https://www.geeksforgeeks.org/node-js-introduction/) | [JavaScript](https://www.geeksforgeeks.org/introduction-to-javascript/) |
| --- | --- |
| Server-side runtime environment | Client-side scripting language |
| Allows running JavaScript code on server | Primarily used for web development |
| Built on Chrome’s V8 JavaScript engine | Runs in a web browser’s JavaScript engine |
| Enables building scalable network applications | Executes code within a browser environment |
| Provides access to file system and network resources | Limited to browser APIs and capabilities |
| Supports event-driven, non-blocking I/O operations | Executes in a single-threaded event loop |
| Used for building backend APIs, servers, and applications | Utilized for creating interactive web pages and client-side logic |

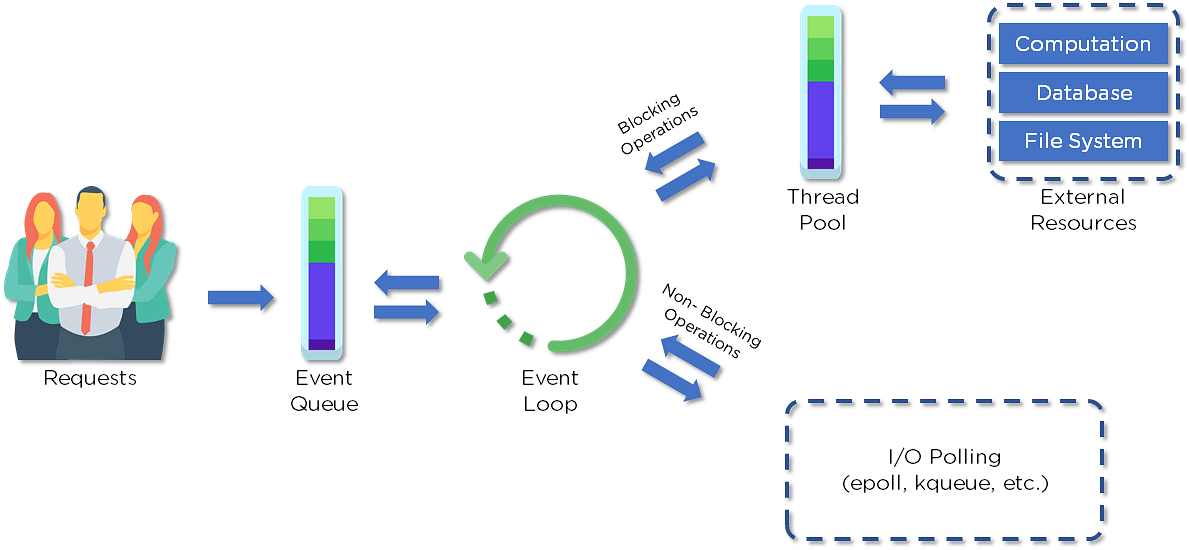
**Q3: What are the benefits of using Node.js? ☆☆**

**Answer:** Following are main benefits of using Node.js

* **Aynchronous and Event Driven** - 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.
* **Very Fast** - Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
* **Single Threaded but highly Scalable** - Node.js uses a single threaded model with event looping. Event mechanism helps server to respond in a non-bloking ways and makes server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and same program can services much larger number of requests than traditional server like Apache HTTP Server.
* **No Buffering** - Node.js applications never buffer any data. These applications simply output the data in chunks.

3. **How does Node.js work?**

A web server using Node.js typically has a workflow that is quite similar to the diagram illustrated below. Let’s explore this flow of operations in detail.



* Clients send requests to the webserver to interact with the web application. Requests can be non-blocking or blocking:
* Querying for data
* Deleting data
* Updating the data
* Node.js retrieves the incoming requests and adds those to the Event Queue
* The requests are then passed one-by-one through the Event Loop. It checks if the requests are simple enough not to require any external resources
* The Event Loop processes simple requests (non-blocking operations), such as I/O Polling, and returns the responses to the corresponding clients

A single thread from the Thread Pool is assigned to a single complex request. This thread is responsible for completing a particular blocking request by accessing external resources, such as computation, database, file system, etc.

Once the task is carried out completely, the response is sent to the Event Loop that sends that response back to the client.

**Q4. Is Node.js single-threaded? If yes then explain**

Yes,[Node.js is single-threaded](https://www.geeksforgeeks.org/why-node-js-is-a-single-threaded-language) by default.

**Q5. What is single-threaded in node js?**

Understanding Single-Threaded Architecture

In computing, a thread is the smallest unit of processing that can be scheduled by an operating system.

Traditional server-side applications often use a multi-threaded approach, where multiple threads handle multiple tasks concurrently. Each thread operates independently, allowing for parallel execution of code.

[Node.js](https://www.geeksforgeeks.org/nodejs/), on the other hand, employs a single-threaded model. This means it uses a single thread to handle multiple tasks. To achieve this, Node.js leverages an event-driven, non-blocking I/O model.

The Event Loop and Non-Blocking I/O

The core of Node.js’s single-threaded architecture is the event loop. The event loop continuously cycles through a series of phases, executing callbacks and handling events. Here’s a simplified breakdown of how it works:

**Event Queue**: Incoming requests are placed into an event queue.

**Event Loop:** The event loop picks up requests from the queue and processes them.

**Callbacks**: For each request, Node.js executes the associated callback function. If a callback involves a blocking operation (like file I/O or network requests), it delegates this to the thread pool.

**Thread Pool:** Node.js uses a thread pool (managed by the libuv library) to handle these blocking operations. Once completed, the results are placed back into the event queue for the event loop to process.

By default we have 4 threads at a time.

We can also Increase the Thread size but it’s depending upon our CPU.

Ex: If our CPU has 8 cores so the thread increases to 8.

This design allows Node.js to handle many connections simultaneously without creating a new thread for each connection. Instead, it efficiently manages I/O operations using non-blocking callbacks.

A screenshot of a computer

Description automatically generated



**Q6. What kind of API function is supported by Node.js?**

There are two types of [API functions](https://www.geeksforgeeks.org/types-of-api-functions-in-node-js) supported by Node.js:

Synchronous: These API functions are used for blocking code.

Asynchronous: These API functions are used for non-blocking code.

**Q7. 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. |

**Q8. What are Modules 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.

Note: Nodejs uses the *CommonJS Module standard* implementation in its module ecosystem which allows you to create modular files and share functionality between them.

**Types of Modules:**In Nodejs, there is 3 type of modules namely

* Core Modules
* Local Modules
* Third-Party Modules

Core Modules

The built-in-module in the node.js is known as core module.

There are two types of core modules.

1. Global Module: The module which we can not require and use directly that module is known as global modules.

Ex: console.log

1. Non-Global Module : The modules which need to import using require method then use is known as non-global modules.

Ex: fs,HTTP,etx.

// Working code of fs module

const fs = require('fs'); // Import fs module

fs.writeFileSync('notes.txt', 'I love to code');

Here is the list of some Nodejs Core Modules:

|  |  |
| --- | --- |
| **Core Modules Name** | **Description** |
| fs | To handle the file system. |
| http | To make Node.js act as an HTTP server |
| https | To make Node.js act as an HTTPS server. |
| os | It provides information about the operation system. |
| path | To handle file paths. |
| cluster | To split a single Node process into multiple processes. |
| dns | To do DNS lookups and name resolution functions. |
| tls | To implement TLS and SSL protocols. |
| querystring | To handle URL query strings. |
| url | To parse URL strings. |
| events | To handle events |
| timers | To execute a function after a given number of milliseconds. |

**Local Module**

In Node.js, a local module refers to a module that is created and used within the scope of a specific project.

We don’t installed globally or shared across multiple projects.

Local modules are typically written by the developer and stored within the project's directory structure.

They are specific to the project and are not available outside of it unless explicitly shared.

**Key Characteristics of Local Modules:**

1. **Defined Within the Project**: Local modules are created as files or directories within the project. They are not installed from a central repository but are custom-defined by the developer.
2. **Scope**: They are only available to the project in which they are defined. This means they are included and used through relative paths or require statements specific to the project directory.
3. **No Global Impact**: Local modules do not affect or interact with other Node.js projects unless explicitly shared or included.
4. **Managed via module.exports and require**:
   * **Exporting**: In the module file, you use module.exports to specify what functionality (functions, objects, classes, etc.) should be made available to other files.
   * **Importing**: In other files within the same project, you use require with a relative path to include the local module and use its functionality.

**Example of a Local Module:**

1. **Creating the Module**:
   * Create a file named mathUtils.js in your project directory
2. // mathUtils.js
3. function add(a, b) {
4. return a + b;
5. }
6. function subtract(a, b) {
7. return a - b;
8. }
9. module.exports = { add, subtract };

**Using the Local Module**:

* Create another file, such as app.js, in the same project directory.
* // app.js
* const mathUtils = require('./mathUtils'); // Import the local module
* console.log(mathUtils.add(5, 3));       // Output: 8
* console.log(mathUtils.subtract(5, 3));  // Output: 2

**Third-party Modules:**

Third-party modules can be installed from the NPM (Node Package Manager) available online.

Firstly we need to initialize the npm using the *npm init*command before npm can be used. It creates a package.json file in the root directory and it stores all the information about the third-party module that we have installed as a dependency.

To install the third part module we write

npm install “module\_name”

ex: npm install express, npm install bcrypt etc.

**Q9. What is npm?**

Npm stands for Node Package Manager.

It is a command-line tool and a package registry that allows developers to manage and share code packages or modules.

It makes it easy to add and use third-party libraries and tools created by other people in your Node.js applications.

Its advantages include dependency management, version control, centralized repository, and seamless integration with Node.js projects.

**Q10. What is middleware?**

[Middleware](https://www.geeksforgeeks.org/middleware-in-express-js) is the function that works between the request and the response cycle. Middleware gets executed after the server receives the request and before the controller sends the response.

**Types of Middleware**

Express JS offers different types of middleware and you should choose the middleware on the basis of functionality required.

* **Application-level middleware:**Bound to the entire application using **[app.use()](https://www.geeksforgeeks.org/express-js-app-use-function/)** or **[app.METHOD()](https://www.geeksforgeeks.org/express-js-app-method-function/)** and executes for all routes.
* **Router-level middleware**: Associated with specific routes using[**router.use()**](https://www.geeksforgeeks.org/express-js-router-use-function/) or **[router.METHOD()](https://www.geeksforgeeks.org/express-js-router-method-function/)** and executes for routes defined within that router.
* **Error-handling middleware:**Handles errors during the request-response cycle. Defined with four parameters (err, req, res, next).
* **Built-in middleware:**Provided by Express (e.g., express.static, express.json, etc.).
* **Third-party middleware**: Developed by external packages (e.g., body-parser, morgan, etc.).

**Q11. What is Control Flow?**

Control flow refers to the order in which individual statements, instructions, or function calls are executed or evaluated in a programming language.

Control flow mechanisms allow a program to make decisions, repeat operations, and jump to different sections of code based on certain conditions.

**Key Control Flow Mechanisms**

* **Sequential Execution:** Code is executed line by line, from top to bottom.
* **Conditional Statements:** if, else if, and else statements allow code to be executed based on specific conditions.
* **Loops:** for, while, and do...while loops enable repetitive execution of code blocks.
* **Function Calls:** Functions are invoked based on the program’s flow and the logic defined.
* **Exception Handling:** try, catch, and finally blocks manage errors and exceptions in the program.

**Q12. What is the Event Loop?**

The event loop allows [Node](https://www.geeksforgeeks.org/nodejs/) to perform non-blocking I/O operations despite the fact that JavaScript is single-threaded.

It is done by assigning operations to the operating system whenever and wherever possible.

**Why Event Loop is important?**

Most operating systems are multi-threaded and hence can handle multiple operations executing in the background. When one of these operations is completed, the kernel tells Node.js, and the respective callback assigned to that operation is added to the event queue which will eventually be executed. This will be explained further in detail later in this topic.

Features of Event Loop:

* An event loop is an endless loop, which waits for tasks, executes them, and then sleeps until it receives more tasks.
* The event loop executes tasks from the event queue only when the call stack is empty i.e. there is no ongoing task.
* The event loop allows us to use callbacks and promises.
* The event loop executes the tasks starting from the oldest first.(FIFO)

**13. What are the main disadvantages of Node.js?**

Here are some main [disadvantages of Node.js](https://www.geeksforgeeks.org/the-pros-and-cons-of-node-js-in-web-development) listed below:

* **Single-threaded nature:** May not fully utilize multi-core CPUs, limiting performance.
* **NoSQL preference:** Relational databases like MySQL aren’t commonly used.
* **Rapid API changes:**Frequent updates can introduce instability and compatibility issues.

**Q14. What is REPL?**

Full form is **(READ, EVAL, PRINT, LOOP).**

It is Similar to the Shell (Unix/Linux) and command prompt, REPL is command prompt for node.js.

We can run our node.js code in REPL.

The work of REPL can be understood from its full form:

* **Read**: It reads the inputs from users and parses it into JavaScript data structure. It is then stored to memory.
* **Eval**: The parsed JavaScript data structure is evaluated for the results.
* **Print**: The result is printed after the evaluation.
* **Loop**: Loops the input command. To come out of NODE REPL, press **ctrl+c** twice

**15. 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 used at client-side that runs on the browser.

**Q16. What is package.json in Node.js?**

The **package.json** file is the heart of Node.js system. It is a metadata file that contains project-specific information such as dependencies, scripts, version, author details, and other configuration settings required for managing and building the project.

**Q17. What is Event-Driven in Node.js**

Event-driven programming is a programming pattern that focuses on the flow of events or actions that occur within a program. In this pattern, the program responds to events as they occur, rather than executing code in a linear manner.

This approach is particularly useful for building interactive applications, such as web applications.

In Node.js, the event-driven model allows developers to write non-blocking, asynchronous code that responds to events as they occur, without waiting for blocking operations to complete.

In Node.js, events are handled using the EventEmitter class.

This class provides a way to raise and handle events within a program.

To use the EventEmitter class, you need to create an instance of it, and then use the on() method to attach event listeners to the instance. When an event occurs, the EventEmitter instance will call all registered listeners in order.

Here is an example code that demonstrates the use of EventEmitter in Node.js:

const EventEmitter = require('events');  
  
// create a new instance of EventEmitter

const myEmitter = new EventEmitter();

// attach a listener for the 'greet' event

myEmitter.on('greet', () => {

  console.log('Hello, world!');

});

// emit the 'greet' event

myEmitter.emit('greet');

A diagram of a diagram

Description automatically generated

**Q18. What is Stream In Node.js?**

Streams in NodeJS are a way to move data from a source to a destination in a bit-by-bit (or let’s say, in chunks), to avoid any Out-of-Memory Errors.

Streams are collections of data — just like arrays or strings.

The difference is that streams might not be available all at once, and they don’t have to fit in memory.

This makes streams really powerful when working with large amounts of data, or data that’s coming from an external source one *chunk* at a time.

**Example 1**

To understand what exactly a stream does, let us consider a scenario.  
We have two buckets named source and destination. The source bucket is full of water, while the destination bucket is empty. We cannot move these two buckets from their spots, however, we have a third movable bucket called buffer. We can use this buffer to transfer water from source to destination.

Now, in the simplest of approach, let’s say, we transfer all the water from source to the buffer bucket. We carry this buffer to the destination, and then transfer everything from buffer to destination. However, there is an alternate approach as well, if we have a hose pipe called stream. We can connect source and destination with this stream, and then transfer water from source to destination in small amounts. This is what streams enable us with in NodeJS.



Explanation:

* fs.createReadStream('example.txt', { encoding: 'utf8' }): This creates a readable stream for the file example.txt. The encoding option is set to 'utf8' to read the file as a string.
* readStream.on('data', callback): This event is emitted whenever a chunk of data is available to read. The callback function processes each chunk.
* readStream.on('end', callback): This event is emitted when the stream has been fully read, indicating that there is no more data.
* readStream.on('error', callback): This event is emitted if an error occurs while reading the stream.

**Q19. What is Buffer in Node.js?**

The**Buffer class in Node.js** is used to **perform operations on raw binary data**. Generally, **Buffer refers to the particular memory location in memory**.

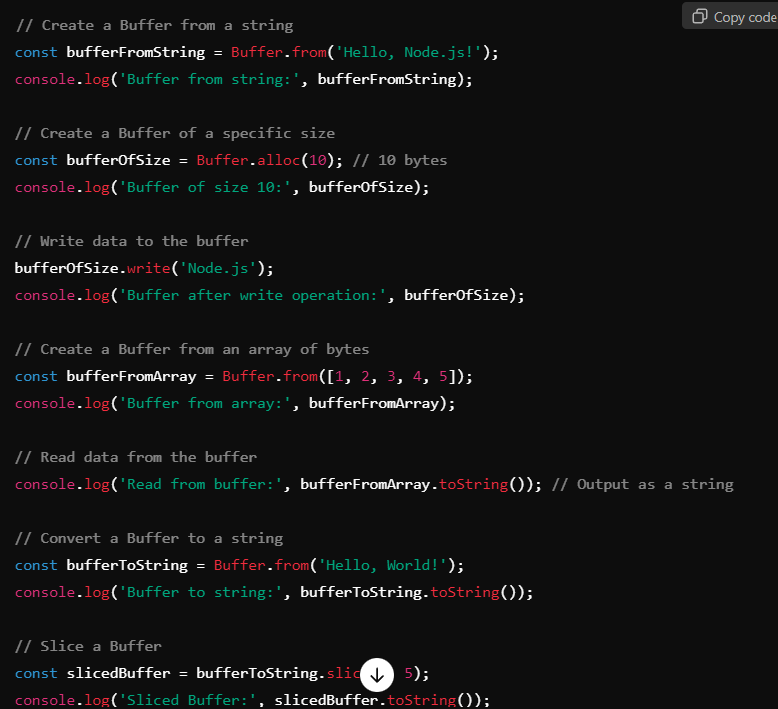
Buffer and array have some similarities, but the difference is array can be any type, and it can be resizable. Buffers only deal with binary data, and it can not be resizable

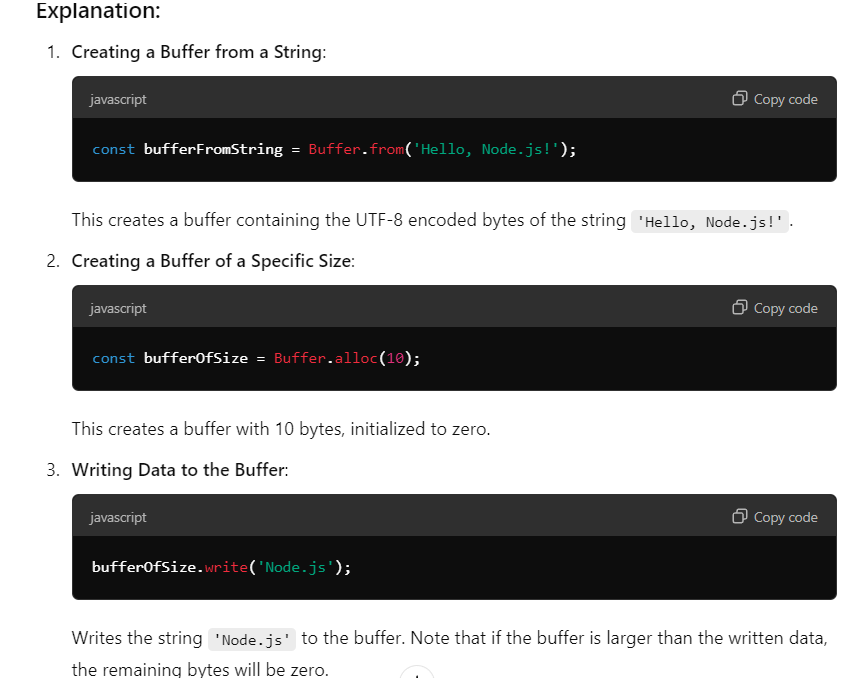
Binary data refers to data that consists of binary values, as opposed to text data, which consists of characters and symbols.

Examples of binary data include images, audio and video files, and raw data from a network.

**Buffer Methods:**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| **1** | Buffer.alloc(size) | It creates a buffer and allocates size to it. |
| **2** | Buffer.from(initialization) | It initializes the buffer with given data. |
| **3** | Buffer.write(data) | It writes the data on the buffer. |
| **4** | toString() | It read data from the buffer and returned it. |
| **5** | Buffer.isBuffer(object) | It checks whether the object is a buffer or not. |
| **6** | Buffer.length | It returns the length of the buffer. |
| **7** | Buffer.copy(buffer,subsection size) | It copies data from one buffer to another. |
| **8** | Buffer.slice(start, end=buffer.length) | It returns the subsection of data stored in a buffer. |
| **9** | Buffer.concat([buffer,buffer]) | It concatenates two buffers. |



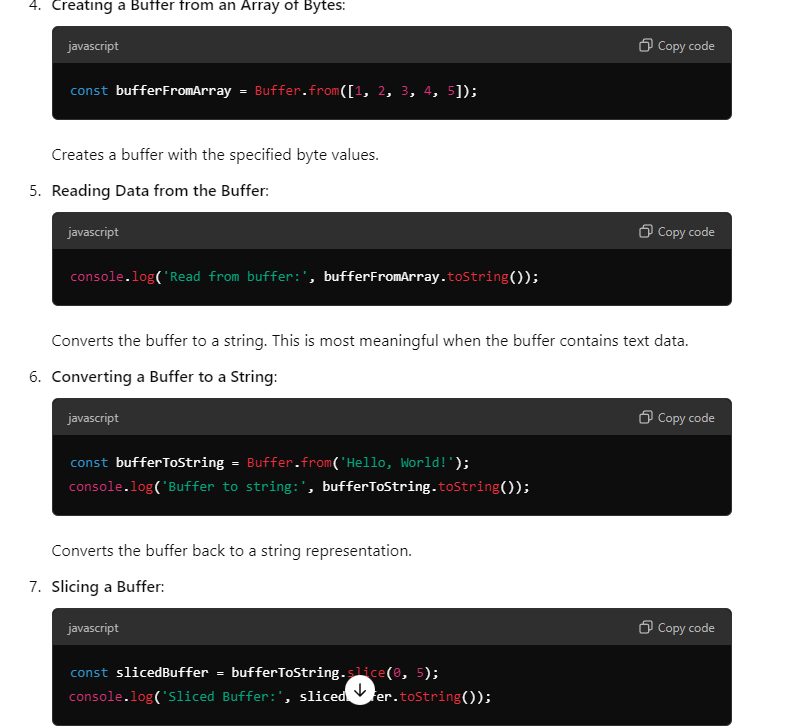


**Q20. What is the Crypto Module?**

The crypto module in Node.js is part of the core libraries, meaning it is built into Node.js and does not require any external dependencies. It provides a way to perform cryptographic operations such as:

* Hashing data to produce fixed-size digests.
* Encrypting and decrypting data using symmetric and asymmetric algorithms.
* Generating cryptographic signatures and verifying them.
* Creating secure random numbers and keys.

These capabilities make the crypto module an essential tool for developing secure applications that require data integrity, confidentiality, and authentication.



**Q21 . Explain Some important modules of Node.js?**

There are many modules in which some are very important

1. Fs module
2. Path module
3. Os module

**File System Module**

The Node.js file system module allows developers to work with the file system on your computer.

To include the File System module, use the require() method:

**Extra Knowledge**

NodeMailer

 **Port 25**: Typically not used for client-to-server communication due to spam issues.

 **Port 465**: Used for SMTP over SSL/TLS but is somewhat deprecated.

 **Port 587**: Recommended for SMTP with STARTTLS for secure email submission.

Important FO Migration

Sometimes the type of the data-time column get change so check it in new database.