Q1. What is Node.js and what makes it unique?

Ans: Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that executes JavaScript code server-side. It allows developers to build fast, scalable, and efficient applications using JavaScript.

What makes it unique is its ability to handle multiple requests simultaneously with its event-driven, non-blocking I/O model, making it a popular choice for real-time applications such as chat applications and online games.

Q2. Explain the event loop in Node.js.

Ans: The event loop in Node.js is a single-threaded mechanism responsible for executing JavaScript code and managing async operations. When a request comes in, it's added to a task queue and the event loop picks it up and processes it. While processing, if an async operation is encountered, instead of blocking the execution, a callback is registered in the task queue and the event loop continues to execute other tasks. When the async operation completes, its callback is added to the task queue and executed by the event loop. This process repeats, allowing Node.js to handle multiple requests efficiently without blocking the main thread.

Q3. What is the difference between synchronous and asynchronous code in Node.js?

Ans: Synchronous code in Node.js is executed in a blocking manner, meaning the next line of code won't be executed until the current line has been completed. This can lead to performance issues, as it blocks the event loop from processing other tasks.

Asynchronous code, on the other hand, is executed non-blocking and doesn't halt the execution of subsequent code. Instead, it registers a callback function which will be executed once the async operation has been completed. This allows Node.js to handle multiple tasks efficiently, as the event loop can continue to process other tasks while waiting for the completion of async operations.

Q4. What is the purpose of the Node Package Manager (NPM)?

Ans: Node Package Manager (NPM) is the default package manager for the Node.js platform. It provides a centralized repository of open-source packages that can be easily installed and used in a Node.js project.

The purpose of NPM is to simplify the process of managing dependencies and sharing code between projects and developers. With NPM, developers can:

1. Install packages from the NPM registry with a single command.
2. Automatically manage dependencies and resolve conflicts.
3. Publish their own packages to the NPM registry for others to use.
4. Easily update and manage packages within their project.

Overall, NPM makes it easy for Node.js developers to reuse code, manage dependencies, and collaborate on projects.

Q5. Explain Buffers in Node.js.

Ans: A Buffer in Node.js is a data structure for storing binary data, such as an image or a video file. It is similar to an array of integers but represents fixed-length binary data instead of a dynamic array of characters.

Buffers are useful in Node.js because they allow efficient handling of binary data without having to perform slow encoding/decoding operations between binary and text data formats. For example, when reading a binary file, it can be read directly into a buffer, and then used without converting to a text format.
Buffers can be created from strings, arrays, or from a specified size in bytes. They have various methods for manipulating binary data, such as concatenation, slicing, and comparing.

Q6. What is the difference between the process.nextTick and setImmediate functions in Node.js?

Ans: process.nextTick and setImmediate are two methods in Node.js for scheduling a function to be executed after the current turn of the event loop. However, there is a difference in the priority and timing of the execution of these functions.
process.nextTick is a low-level method that schedules a function to be executed after the current turn of the event loop but before any other I/O events or timers. It has the highest priority and is executed as soon as possible, before any other events in the queue.
setImmediate is a higher-level method that schedules a function to be executed after all I/O events and timers have been processed. It has a lower priority compared to process.nextTick, but it is still executed before any new I/O events or timers.

Q7. What is the global object in Node.js?

Ans: The global object in Node.js is a JavaScript object that is available in all modules and represents the global context of the application. It provides a way to access properties and functions that are globally available, without having to specify the global object explicitly.
In Node.js, the global object is represented by the global keyword. It is similar to the window object in a web browser, and contains properties and methods that are available globally, such as console, setTimeout, and process.

Q8.What is the difference between a Node.js module and a JSON file?

Ans: A Node.js module is a self-contained unit of code that defines a specific functionality, and can be exported and imported into other parts of a Node.js application. Modules are typically implemented as a single JavaScript file and can include functions, objects, and variables that are specific to that module.
In contrast, a JSON (JavaScript Object Notation) file is a data format that is used to store and exchange data between applications. It is a lightweight, text-based, and language-independent format that can be used to represent simple data structures, such as arrays and objects.

Q9. What is the role of the Express.js framework in Node.js?

Ans: Express.js is a web application framework for Node.js that provides a robust set of features for web and mobile applications. It simplifies the process of building and deploying web applications by providing a minimal set of capabilities needed for the majority of web applications, including routing, middleware, and template engines.

Q10. How do you handle uncaught exceptions in Node.js?

Ans: In Node.js, uncaught exceptions can be handled by adding a global exception handler using the 'process. on' method. This will catch any uncaught exceptions and log the error stack trace. It is important to exit the process after handling the exception to ensure that the process does not continue to run in a potentially unstable state.

Alternatively, you can use a process manager like PM2, which automatically restarts the node process in case of an uncaught exception and can log the errors.

Q11.How does clustering work in Node.js?

Ans: Clustering in Node.js is a way to scale an application by creating multiple worker processes that share a single port and work together to handle incoming requests. The Node.js cluster module allows you to create child processes (workers) that can share the same port with the master process. The master process listens on the port and delegates incoming connections to the workers.

Each worker process runs in its own memory space and has its own event loop. This allows the application to take advantage of multiple cores and improve performance. The cluster module also provides load balancing and failover capabilities, as it can automatically redirect incoming connections to another worker in case one worker crashes.

Q12. How does Node.js handle multiple requests simultaneously?

Ans: Node.js uses an event-driven, non-blocking I/O model to handle multiple requests simultaneously. This means that it does not block the execution of the program while waiting for a response from a database or a network request but instead registers a callback function to be executed once the response is received.

The Node.js event loop listens for incoming events and places them in a queue. The event loop then processes events from the queue one by one, executing the associated callback functions. While the event loop is executing a callback function, it can continue to accept and queue new events, allowing the program to handle multiple requests simultaneously.

This allows Node.js applications to handle many concurrent connections with a small number of threads, improving performance and scalability compared to traditional blocking I/O models.

Q13. What are the most common use cases for Node.js?

Ans: Node.js is commonly used for the following use cases:

1. Backend Web Development: Node.js is often used as a backend platform for web applications, providing a server-side JavaScript environment for handling HTTP requests, database access, and other server-side tasks.
2. Real-time Applications: Node.js is well suited for building real-time applications such as chat applications, online games, and collaborative tools, due to its fast and scalable event-driven architecture.
3. Command Line Tools: Node.js can be used to create command line tools, providing a convenient way to automate tasks and perform system administration tasks.
4. Microservices: Node.js is often used to build microservices, allowing developers to build scalable, modular systems with a small footprint and low overhead.

5. Internet of Things (IoT) Applications: Node.js can be used to develop IoT applications, providing a lightweight and efficient runtime for devices with limited resources.

Q14. What is the difference between the ES6 and CommonJS module systems?

Ans: ES6 and CommonJS are two different module systems used in JavaScript. They have different syntaxes, loading patterns, and scoping rules.

The ES6 module system is a standard for organizing and sharing JavaScript code that was introduced with ECMAScript 6 (ES6). ES6 modules are designed to work natively in modern browsers and in Node.js. They support static analysis and tree shaking, which can improve the efficiency of code bundlers.

Q15. What is the difference between a callback and a Promise in Node.js?

Ans: In Node.js, a callback, and a Promise are both used to handle asynchronous operations, but they differ in their implementation and usage.

A callback is a function that is passed as an argument to another function and is executed when the operation that the callback function was passed to has been completed. Callbacks are commonly used in Node.js to handle the completion of asynchronous operations, such as reading from a file or making an HTTP request.