Q1. What is Node.js? Where can you use it?

Node.js is a runtime environment that allows you to run JavaScript code outside of a web browser. It is built on the V8 JavaScript engine, which is the same engine that powers Google Chrome. Node.js enables developers to write server-side and networking applications in JavaScript

Node.js is a versatile platform that can be used for a wide range of applications, from building simple REST APIs to complex, real-time web applications.

Uses of Node.js

- Server-side scripting
- Building APIs
- Real-time applications
- Command-line tools
- Microservices architecture
- Data streaming applications

Q2. Explain callback in Node.js.

In Node.js, a callback is a function passed as an argument to another function, which will be invoked later when a particular operation is completed or when a certain condition is met.

It is a non-blocking function that executes upon task completion, enabling asynchronous processing. It facilitates scalability by allowing Nodejs to handle multiple requests without waiting for operations to conclude.

Q3. What are the advantages of using promises instead of callbacks?

There are several advantages of promises over the callbacks, which make asynchronous code more readable, maintainable, and easier to work with

- Promises allow you to write asynchronous code in a more sequential and readable manner, resembling synchronous code. This is because promises chain together using .then() and .catch() methods, making the code flow clearer and easier to understand compared to nested callbacks.
- Promises help to mitigate the problem of callback hell, which occurs when you have multiple nested callbacks, leading to code that is difficult to read and maintain. Promises allow you to chain asynchronous operations together without nesting, resulting in cleaner and more manageable code.
- Promises provide built-in error handling mechanisms through the .catch() method, allowing you to handle errors in a centralized manner at the end of the promise chain. This makes error

handling more explicit and easier to manage compared to callbacks, where error handling often involves nested if statements.

- Promises support method chaining, which allows you to perform multiple asynchronous operations sequentially in a fluent and concise manner. This makes it easy to express complex asynchronous workflows without nesting callbacks.
- Promises serve as the foundation for async/await syntax in JavaScript, which provides a more synchronous-looking syntax for writing asynchronous code. Async/await builds on top of promises, making asynchronous code even more readable and easier to reason about, especially for developers familiar with synchronous programming.
- Promises support functional programming concepts like composition, allowing you to compose multiple asynchronous operations together using methods like Promise.all() and Promise.race(). This enables you to execute multiple asynchronous tasks concurrently or sequentially and handle their results collectively.

Q4. What is NPM?

NPM stands for Node Package Manager. It is the default package manager for Node.js, and it is used to install, manage, and share packages of JavaScript code. NPM comes bundled with Node.js installation, making it readily available for developers working with Node.js projects.

Q5. What are the modules in Node.js? Explain

In Node.js, modules are reusable pieces of code that encapsulate related functionality. They help organize code into separate files or directories, making it easier to manage, maintain, and reuse code across different parts of an application.

There are two types of modules in Node.js

1. Core Modules
2. User-Defined Modules

Core Modules:

These are built-in modules that come bundled with Node.js and provide essential functionality for various tasks such as file system operations, networking, and cryptography. Core modules are accessed using their module names without requiring a file path.

Some examples of core modules

- fs (file system)
- http (HTTP server/client)
- path (file path utilities)
- util (utility functions).

User-Defined Modules

These are modules created by developers to encapsulate specific functionality within their applications. User-defined modules are typically organized into separate files or directories and are loaded using relative paths. Each file in Node.js is treated as a separate module, and the functionality within. the file can be exported and imported into other files using the module.exports or exports object.