Node.js Basics

1. What is Node.js and what is it used for?

Node.js is a runtime environment that allows JavaScript to be executed on the server side. It is used for building scalable, high-performance network applications and real-time services.

2. Explain the main differences between Node.js and traditional web server environments like Apache or Nginx.

Architecture:

- **Node.js:** Single-threaded, event-driven, non-blocking I/O. Handles many connections simultaneously.
- **Apache/Nginx:** Multi-threaded or multi-process. Each connection may create a new thread/process.

Language:

- **Node.js:** Uses JavaScript for server-side scripting.
- **Apache/Nginx:** Uses various languages (PHP, Python, Ruby) through modules.

Use Case:

- **Node.js:** Best for real-time applications, like chat apps and games.
- **Apache/Nginx:** Suitable for serving static content, handling traditional web server tasks.

Handling Requests:

- **Node.js:** Handles many requests in a single thread using asynchronous callbacks.
- **Apache/Nginx:** Uses multiple threads/processes, which can handle each request separately.

3. What is the V8 engine and how does Node.js utilize it?

V8 engine is Google's open-source JavaScript engine that compiles JavaScript directly into machine code.

How Node.js uses it:

- **Runs JavaScript:** Node.js uses V8 to run JavaScript on the server.
- **Speed:** V8 makes Node.js fast by turning JavaScript into machine code.

4. Describe the event-driven architecture of Node.js.

Node.js employs an event-driven architecture where operations are handled asynchronously, responding to events like I/O requests or user actions, optimizing efficiency and scalability in server-side applications.

5. What are some common use cases for Node.js?

- Real-Time Applications
- API Development
- Streaming Applications
- Server-Side Web Applications

6. How does Node.js handle asynchronous operations?

- Event Loop
- Promises
- Callbacks
- Async/Await

7. What is the purpose of the package.json file in a Node.js project?

The package.json file in a Node.js project specifies the project's metadata, dependencies, and scripts, enabling easy management and sharing of the project's configuration. It serves as the central repository for project settings, facilitating consistent development and deployment.

8. Explain the role of the Node Package Manager (NPM).

npm is a package manager that allows us to install third-party packages.

9. What is the node_modules folder and why is it important?

The node_modules folder is where all the packages (or modules) your project needs are stored. When you install packages using npm, they go into this folder. It's important because it keeps all the code your project needs to work.

10. How can you check the version of Node.js and NPM installed on your system?

Using the `node -v` command.

11. How does Node.js handle concurrency and what are the benefits of this approach?

Node.js uses an event-driven, non-blocking I/O model to handle concurrency. This means it can handle many tasks at once without waiting for one task to finish before starting another.

Benefits:

- **Efficiency:** It can handle many connections at the same time.
- **Speed:** Tasks can be processed quickly without delays.
- **Scalability:** It works well for real-time applications like chat apps or online games.

12. How does Node.js handle file I/O? Provide an example of reading a file asynchronously.

Node.js handles file I/O using its built-in `fs` (file system) module. It can read and write files.

Example:

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

fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) {
    console.error(err);
    return;
  }
  console.log(data);
});
```

13. What are streams in Node.js and how are they useful?

Streams are objects that let you read or write data continuously.

Uses:

- **Efficiency: Handles large data piece by piece.**
- **Performance: Starts processing data as it comes in, which is faster.**

Node.js Modules

1. What are modules in Node.js and why are they important?

- **Code Organization:** Modules help keep your codebase clean and organized by breaking it into smaller files.
- **Reusability:** They promote code reuse, so you can use the same module in multiple parts of your application.
- **Encapsulation:** Modules encapsulate functionality, making it easier to maintain and debug.

2. How do you create a module in Node.js? Provide a simple example.

To create a module in Node.js, you define functions.

Example:

```
const add = (a, b) => a + b;
const subtract = (a, b) => a - b;

module.exports = {
  add,
  subtract
};
```

3. Explain the difference between `require` and `import` statements in Node.js.

- **require:** Used in CommonJS modules (Node.js default) to import modules.
- **import:** Used in ES modules (supported in recent Node.js versions) for the same purpose.

4. What is the `module.exports` object and how is it used?

`module.exports` is a special object in Node.js used to export variables, functions, or objects from one module to another**.**

5. Describe how you can use the `exports` shorthand to export module contents.

```
exports.add = (a, b) => a + b;
exports.subtract = (a, b) => a - b;
```

6. What is the CommonJS module system?

CommonJS is the module system used by Node.js, focusing on simplicity and synchronous loading of modules using `require` and `module.exports`.

7. How can you import a module installed via NPM in your Node.js application?

You import an NPM module using `require` or `import` statements.

8. Explain how the `path` module works in Node.js. Provide an example of using it.

The `path` module provides utilities for working with file and directory paths.

Example:

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

const fullPath = path.join(__dirname, 'files', 'example.txt');
console.log(fullPath);
```

9. How do you handle circular dependencies in Node.js modules?

Circular dependencies occur when modules depend on each other directly or indirectly. They can be managed by refactoring code to avoid mutual dependencies or using required statements at runtime inside functions.

10. What is a built-in module in Node.js? Name a few and explain their purposes.

Built-in modules are part of Node.js core and provide various functionalities without needing external libraries.

Examples include:

- **fs:** File system operations.
- **http:** HTTP server and client functionality.
- **path:** Path-related utilities.
- **util:** Utility functions.

11. What is the difference between relative and absolute module paths in Node.js?

- Relative paths (./ or ../): Refer to modules relative to the current module's directory.
- Absolute paths (/ or module name without ./ or ../): Refer to modules using the full path from the project's root directory.

12. What is a module wrapper function in Node.js?

In Node.js, each module is wrapped in a function wrapper before being executed.

13. Describe the buffer module and its use in Node.js.

The buffer module in Node.js helps manage and manipulate binary data directly in memory. It's essential for tasks like handling network data, reading/writing files, and performing data transformations like encryption.

Starting an HTTP Server in Node.js

1. How do you create a simple HTTP server in Node.js? Provide a code example.

Create an HTTP server in Node.js, you use the `http` module and its `createServer` method.

Example:

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

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, world!\n');
});

server.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

2. Explain the purpose of the `http` module in Node.js.

The `http` module in Node.js provides functionality to create HTTP servers and make HTTP requests. It's essential for building web servers and handling HTTP operations.

3. What method do you use to start the HTTP server and make it listen on a specific port?

You use the `listen` method on the server object to start the HTTP server and specify the port number to listen on.

4. How can you send a response to the client in an HTTP server created with Node.js?

You use the `res` object (short for response) passed to the request handler function. Methods like `res.writeHead` set response headers, `res.write` sends data, and `res.end` finishes the response.

5. Explain the request and response objects in the context of an HTTP server.

- **Request**: Contains information about the client's request, like URL, HTTP method, headers, and body.
- **Response**: Allows sending back data to the client, including headers and content.

6. How do you handle different HTTP methods (GET, POST, etc.) in a Node.js HTTP server?

You check the `req.method` property in the request handler function to determine the HTTP method used (GET, POST, etc.) and write logic accordingly.

7. What is middleware in the context of a Node.js HTTP server?

Middleware are functions that have access to the request and response objects. They can modify these objects, execute additional code, and end the request-response cycle.

8. How can you serve static files using an HTTP server in Node.js?

You can use the express framework or the `http` module along with the `fs` module to read and send static files like HTML, CSS, and images to clients.

9. Explain how to handle errors in an HTTP server created with Node.js.

You use try-catch blocks or the `error` event on the server object to catch errors. You can also use middleware functions for error handling.

10. How can you implement routing in a Node.js HTTP server without using external libraries?

You can implement routing by checking `req.url` in the request handler function and using conditional statements (if or switch) to route requests to different paths to appropriate logic or resources.