Q1 Explain the process of setting up a Node.JS environment, including installing Node.JS and npm, and creating a simple "Hello world" application.

Answer:

(i) Install Node.JS and npm:

- Download the latest node.JS installer from https://nodejs.org.
- Run the installer, which will automatically install both Node JS and Npm (Node package manager).
- verify installation using:

```
node -v
npm -v
```

(ii) Create a project folder:

```
Mkdir myapp
cd myapp
```

(iii) Initialize npm (optional):

```
npm init -y
```

(iv) create a "Hello world" application:

- create a file named app.JS.
- write:

```
console.log("Hello world from node.JS!");
```

(v) Run the application:

```
node app.JS
```

Q2. Discuss the role of the fs (file system) module in node.js Provide an example of how to read a file's content asynchronously using the fs module.

Answer:

- The fs module in node.js allows us to interact with the file system – creating, reading, updating, and deleting files.
- It provides both synchronous and asynchronous methods, but asynchronous is preferred to avoid blocking the event loop.

Example:

```
const fs = require("fs");
fs.readFile("example.txt", "utf8", (err, data) => {
  if (err) {
      console.error("Error reading file:", err);
  return;
  }
  console.log("file contents :", data);
  });
```

Q3. Describe the event loop in node.js and its importance. How does the event-driven architecture of node.js differ from traditional synchronous programming?

Answer:

- The event loop is the core of node.js's runtime. It allows node.js to perform non-blocking I/O operations despite javascript being single-threaded.
- The event loop continuously checks the callback queue and executes pending tasks, enabling asynchronous programming.

importance:

- prevents blocking of the main thread.
- Handles thousands of concurrent connections efficiently.
- Enable real-time, scalable applications.

Difference from Synchronous programming:

- Traditional Synchronous: Tasks execute line by line, blocking the program until one finishes.
- Node.js asynchronous (event-driven): Tasks (like file read, network requests) are offloaded, and once they finish, the callbacks/events are pushed to the event loop queue for execution.

Q4. Describe the process of working with the file system (fs) module in node.js. provide an example where you read the contents of a file asynchronously and handle potential errors during the read operation.

Answer:

- The fs module lets developers perform file operations such as reading, writing, deleting, and updating files.
- Using asynchronous functions avoids blocking the event loop.

Example:

```
const fs = respire ("fs");
fs.readfile ("info.txt", "utf8", (err, data) => {
 if (err) {
  console.error ("Error reading:", err);
 return;
}
console.~~error~~ log ("~~Error reading:~~", ~~err~~); ("file content:", data);
});
```

Here:

- if info.txt does not exist or has permission issue, the error will be caught.
- If success, file contents will be printed.

q.5. Explain the concept of asynchronous programming in Node.js How do callbacks work in Node.js and what are some of the common pitfalls when using callbacks for asynchronous operations? Illustrate with an example.

⇒ Asynchronous programming in node.js means operations don't block program execution. Instead of waiting for one task to finish. Node.js continues running other code.

- Callbacks are functions passed an arguments to asynchronous functions. Once the async task completes, the callback is executed.

Pitfalls of callbacks:

(i) Callback Hell (nested callbacks, hard to read/maintain).
(ii) Error handling issues if not structured properly.
(iii) Inversion of control - giving too much control to callback functions.

Ex-

```
const fs = require("fs");
fs.readFile("info.txt", "utf8", (err, data) => {
    if(err) {
        console.err("Error reading:", err);
        return;
    }
    console.log("file content:", data);
});
```

Better solutions:

Use promises or async/await instead of deep callbacks