Node.js Tutorial for Beginners

Welcome to the Node.js tutorial for beginners! This guide is designed to help you learn Node.js from scratch. We'll cover the basics, provide coding examples, and include quiz

questions to test your understanding. By the end of this tutorial, you'll be able to build simple server-side applications using Node.js. Let's get started!

Introduction to Node.js **3**
- Why Learn Node.js? 3

Setting Up Node.js **3**
- Download and Install Node.js 3
- Verify Installation 3
- Installing a Code Editor 4

Your First Node.js Program **4**
- Hello World Example 4
- Quiz Question 4

Modules in Node.js **5**
- Built-in Modules 5
- Using the require Function 5
- Creating Custom Modules 5
- Quiz Question 6

File System Module **6**
- Reading Files 7
- Writing Files 7
- Quiz Question 7

Events in Node.js **7**
- Using the EventEmitter Class 7
- Quiz Question 8

Creating a Web Server **8**
- Simple HTTP Server 9
- Quiz Question 9

npm and Packages **10**
- Initializing a Project 10
- Installing Packages 10
- Using Installed Packages 10
- Quiz Question 11

Asynchronous Programming **11**
- Callbacks 11
- Promises 12
- Async/Await 12
- Quiz Question 12

Building a Simple API with Express.js **13**
Setting Up Express.js 13
Creating Routes 13
Testing the API 14
Quiz Question 14
Conclusion **15**
Final Quiz **15**

Introduction to Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside of a web browser. It's built on Chrome's V8 JavaScript engine and allows developers to use JavaScript for server-side scripting.

Why Learn Node.js?

- **Unified Language**: Use JavaScript on both the front-end and back-end.
- **Scalability**: Ideal for building scalable network applications.
- **Rich Ecosystem**: Thousands of packages available through npm.
- **Asynchronous and Event-Driven**: Efficient handling of multiple connections.

Setting Up Node.js

Download and Install Node.js

1. Visit the official Node.js website.
2. Download the **LTS (Long Term Support)** version for your operating system.
3. Run the installer and follow the prompts.

Verify Installation

Open your terminal or command prompt and type:

```
node -v
```

This should display the installed Node.js version.

Installing a Code Editor

Choose a code editor like **Visual Studio Code**, **Sublime Text**, or **Atom**.

Your First Node.js Program

Let's create a simple Node.js application.

Hello World Example

1. Create a new directory for your project.
2. Inside the directory, create a file named `app.js`.

app.js

```
console.log('Hello, Node.js!');
```

3. Run the application:

```
node app.js
```

Output:

```
Hello, Node.js!
```

Quiz Question

Q1: Which command is used to run a Node.js file named `app.js`?

A. `node app.js`
B. `run app.js`

C. `execute app.js`
D. `start app.js`

Answer: A. `node app.js`

Modules in Node.js

Modules are reusable blocks of code whose existence does not accidentally impact other code.

Built-in Modules

Node.js has several built-in modules, such as `fs` (file system), `http`, `url`, etc.

Using the `require` Function

To include a module, use the `require` function.

Example:

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

console.log('Platform:', os.platform());
console.log('Architecture:', os.arch());
```

Output:

```
Platform: win32
Architecture: x64
```

Creating Custom Modules

module1.js

```
function greet(name) {
  return `Hello, ${name}!`;
}

module.exports = greet;
```

app.js

```
const greet = require('./module1');

console.log(greet('Alice'));
```

Output:

```
Hello, Alice!
```

Quiz Question

Q2: How do you export a function from a module in Node.js?

A. `export functionName;`
B. `exports.functionName = functionName;`
C. `module.exports = functionName;`
D. `require('functionName');`

Answer: C. `module.exports = functionName;`

File System Module

The `fs` module allows you to work with the file system on your computer.

Reading Files

Example:

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

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

Writing Files

```
fs.writeFile('output.txt', 'This is some content.', (err) => {
  if (err) throw err;
  console.log('File has been saved!');
});
```

Quiz Question

Q3: Which method is used to read a file asynchronously in Node.js?

A. `fs.readFile()`
B. `fs.readFileSync()`
C. `fs.openFile()`
D. `fs.read()`

Answer: A. `fs.readFile()`

Events in Node.js

Node.js is event-driven, meaning it uses events heavily and is asynchronous.

Using the EventEmitter Class

Example:

```
const EventEmitter = require('events');
const eventEmitter = new EventEmitter();

// Create an event handler
function myEventHandler() {
  console.log('An event occurred!');
}

// Assign the event handler to an event
eventEmitter.on('myEvent', myEventHandler);

// Fire the event
eventEmitter.emit('myEvent');
```

Output:

```
An event occurred!
```

Quiz Question

Q4: Which method is used to bind an event handler to an event in Node.js?

A. `eventEmitter.emit()`
B. `eventEmitter.on()`
C. `eventEmitter.bind()`
D. `eventEmitter.addEventListener()`

Answer: B. `eventEmitter.on()`

Creating a Web Server

Node.js provides the `http` module to create web servers.

Simple HTTP Server

Example:

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

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

const PORT = 3000;

server.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}/`);
});
```

Run the server:

```
node app.js
```

Visit `http://localhost:3000/` in your browser to see the output.

Quiz Question

Q5: Which module is used to create a web server in Node.js?

A. `http`
B. `web`
C. `server`
D. `net`

Answer: A. `http`

npm and Packages

npm (Node Package Manager) is the default package manager for Node.js.

Initializing a Project

```
npm init -y
```

This command creates a `package.json` file.

Installing Packages

```
npm install express
```

This installs the `express` package locally.

Using Installed Packages

Example:

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello, Express.js!');
});

const PORT = 3000;

app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}/`);
});
```

Quiz Question

Q6: What command is used to install a package using npm?

A. `npm get [package-name]`
B. `npm install [package-name]`
C. `npm add [package-name]`
D. `npm new [package-name]`

Answer: B. `npm install [package-name]`

Asynchronous Programming

Node.js uses an asynchronous, non-blocking I/O model.

Callbacks

Functions passed as arguments to other functions.

Example:

```
function fetchData(callback) {
  setTimeout(() => {
    callback('Data received');
  }, 2000);
}

fetchData((data) => {
  console.log(data);
});
```

Output:

```
Data received
```

Promises

An alternative to callbacks for handling asynchronous operations.

Example:

```
function fetchData() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve('Data received');
    }, 2000);
  });
}

fetchData().then((data) => {
  console.log(data);
});
```

Async/Await

Simplifies working with Promises.

Example:

```
async function getData() {
  const data = await fetchData();
  console.log(data);
}

getData();
```

Quiz Question

Q7: Which of the following is NOT a way to handle asynchronous code in Node.js?

A. Callbacks
B. Promises
C. Async/Await
D. Synchronous loops

Answer: D. Synchronous loops

Building a Simple API with Express.js

Express.js is a minimal and flexible Node.js web application framework.

Setting Up Express.js

Install Express:

```
npm install express
```

1.

Create `app.js`:

```
const express = require('express');
const app = express();

app.use(express.json());

const PORT = 3000;

app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}/`);
});
```

2.

Creating Routes

GET Route:

```
app.get('/', (req, res) => {
  res.send('Welcome to my API!');
});
```

POST Route:

```
app.post('/data', (req, res) => {
  const receivedData = req.body;
  res.send(`Data received: ${JSON.stringify(receivedData)}`);
});
```

Testing the API

Use tools like **Postman** or **curl** to test your API endpoints.

Example using curl:

```
curl -X POST -H "Content-Type: application/json" -d
'{"name":"Alice"}' http://localhost:3000/data
```

Output:

```
Data received: {"name":"Alice"}
```

Quiz Question

Q8: Which middleware is used in Express.js to parse JSON bodies?

A. `express.urlencoded()`
B. `express.static()`
C. `express.json()`
D. `bodyParser.text()`

Answer: C. `express.json()`

Conclusion

Congratulations! You've covered the basics of Node.js, including setting up your environment, understanding modules, working with the file system, handling events, creating a web server, using npm, asynchronous programming, and building a simple API with Express.js. Keep practicing and building projects to reinforce your learning.

Final Quiz

Q9: What does the `require` function do in Node.js?

A. Imports modules
B. Exports modules
C. Reads files
D. Creates servers

Answer: A. Imports modules

Q10: Which of the following is a correct way to start a Node.js application named `server.js`?

A. `start server.js`
B. `node server.js`
C. `run server.js`
D. `execute server.js`

Answer: B. `node server.js`