

Node.js & JavaScript Learning Notes (Beginner Friendly)

This file explains everything in the simplest possible way—so even if you're just starting out, you're not lost. Expect simple language, easy examples, and a little humor to keep you awake. 😊



Day 1: Getting Started with Node.js & JavaScript

What is Node.js?

Node.js is like a superhero that lets JavaScript work **outside the browser**. Normally, JavaScript needs a browser like Chrome. But Node.js lets you run .js files directly from your computer's terminal.



Try this:

```
node app.js
```

Is React also a runtime like Node.js?

Nope! React is just a **toolbox** to help you build user interfaces (like buttons, forms, etc.) inside the browser. It reacts (get it?) when your data changes, and updates the page automatically.

Setup Checklist

- Make a folder for your code.
- Open it in VS Code.
- Run `npm init -y` to create a starter file.
- Make a file like `hello.js` and write:

```
console.log("Hello Node.js");
```


Boom! You just ran backend code! 🎉



Day 2: JavaScript Functions & Git Basics

What is `(err) => {}`?


It's a **shortcut** for writing a function. You don't need the word `function`, you just write the arrow.

 Example:

```
const sayHi = (name) => {  
  console.log(` Hello, ${name} `);  
}  
sayHi("You");
```

What does `path.join()` do?


It glues file paths together in a safe way, no matter what operating system you're using (Windows, Mac, Linux).

 Example:

```
const path = require("path");  
const fullPath = path.join("folder", "file.txt");  
console.log(fullPath);
```

What is UTF-8?


It's just a way to **encode text** so computers can read/write files properly. Without it, reading files might look like alien code. 🙄

 Example:

```
fs.readFile("notes.txt", "utf-8", (err, data) => {  
  console.log(data);  
});
```

Where does `data` come from in functions like `readFile()`?

You write the variable name in your function, and Node magically gives it to you. 🤖

 Example:

```
fs.readFile("file.txt", "utf-8", (err, data) => {  
  console.log(data); // Node gives you this 'data'  
});
```

What does `git branch -M main` do?

It **renames your main branch** to main (instead of the older default master). Keeps things modern.



Day 3: The Magic of the File System (fs)

What can you do with fs?

Think of fs like a librarian that lets you:

- **Write to files**

```
fs.writeFile("notes.txt", "Hello world!", (err) => {});
```

- **Add more text** (like writing in a diary)

```
fs.appendFile("notes.txt", "\nAnother line", (err) => {});
```

- **Read files** (duh)

```
fs.readFile("notes.txt", "utf-8", (err, data) => console.log(data));
```

- **Delete files** (carefully!)

```
fs.unlink("notes.txt", (err) => {});
```

- **Make a folder**

```
fs.mkdir("myFolder", (err) => {});
```

- **List what's inside a folder**

```
fs.readdir("./", (err, files) => console.log(files));
```

- **Rename or move files**

```
fs.rename("old.txt", "new.txt", (err) => {});
```

Can I use import instead of require?

Yes, if you either:

- Rename the file to .mjs, or
- Add "type": "module" in your package.json



Example:

```
import fs from "fs";
```

Mini Task

1. Make a folder called myFiles
2. Write to notes.txt
3. Read & print it
4. Add more lines
5. Rename the file
6. Delete it like a boss 😎

Day 4: Creating a Simple Web Server with Node.js

What is a server?

A server is just a computer that waits for requests and responds to them. It's like a restaurant waiter: you ask for something, and it brings it to you.

What does `http.createServer()` do?


It creates a basic server that listens for requests and sends back responses.

Example:

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

```
const server = http.createServer((req, res) => {  
  res.end('Hello from the server!');  
});
```

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

Open your browser and go to `http://localhost:3000` — you'll see “Hello from the server!” 

4

What is a port?

A port is like a door number for your app on your computer. You can have many apps running, and each

one listens on a different port.

What is a request and response?

- Request = What the user (browser) asks for

- Response = What the server sends back

What if I want to do calculations on the backend? Do the calculation in Node and return the result!

Example:

```
const server = http.createServer((req, res) => {
  if (req.url === '/sum') {
    const result = 40 + 2;
    res.end(`The sum is ${result}`);
  } else {
    res.end("Hello from the server");
  }
});
```

How does frontend talk to backend?

- Frontend makes an HTTP request (using fetch, axios, etc.)
- Backend sends a response with data
- Frontend updates the UI with it

It's like a waiter bringing food from the kitchen!

Task

1. Create a file called server.js
2. Create different routes (/ , /about , /sum)
3. Do a calculation on /sum
4. Run and test in browser

🎯 Stretch goal: Connect frontend fetch() with your backend and show data in the UI You've officially made your own server today — not bad for Day 4!



Day 5: Building a Dynamic Server



What is a dynamic server?

- A server that **changes its response** based on the request.
- Can read URLs and **respond differently** to different paths or query parameters.



Example:

```
js
CopyEdit
const http = require('http');
const url = require('url');
```

```
const server = http.createServer((req, res) => {
  const parsedUrl = url.parse(req.url, true);

  if (parsedUrl.pathname === '/') {
    res.end('Welcome to the homepage!');
  } else if (parsedUrl.pathname === '/about') {
    res.end('This is the about page. ');
  } else if (parsedUrl.pathname === '/sum') {
    const a = parseInt(parsedUrl.query.a) || 0;
    const b = parseInt(parsedUrl.query.b) || 0;
    res.end(`The sum is ${a + b}`);
  } else {
    res.statusCode = 404;
    res.end('Page not found');
  }
});

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

✅ Try these URLs:

- /
- /about
- /sum?a=5&b=10

✅ Task:

- Add your own route /greet?name=Supriya that returns “Hello, Supriya!”
-



Day 6: Understanding the Event Loop

✅ The Event Loop is how Node.js handles **asynchronous tasks** without blocking.

✅ Node is **single-threaded** but can manage many operations **in parallel** via the event loop.

✅ Phases of the Event Loop

- **Timers:** setTimeout, setInterval
- **Pending Callbacks:** System callbacks
- **Idle/Prepare:** Internal

- **Poll:** Waiting for new I/O
- **Check:** setImmediate
- **Close Callbacks:** socket.on('close')

✓ Example:

```
js
CopyEdit
console.log('Start');

setTimeout(() => {
  console.log('setTimeout');
}, 0);

setImmediate(() => {
  console.log('setImmediate');
});

process.nextTick(() => {
  console.log('process.nextTick');
});

console.log('End');
```

✓ Expected Output:

```
arduino
CopyEdit
Start
End
process.nextTick
setTimeout
setImmediate
```

✓ Explanation:

- process.nextTick runs *before* the next phase.
- setTimeout waits for the **Timers phase**.
- setImmediate runs in the **Check phase**.

✓ Why learn this?

- Understand when your callbacks run.
- Avoid blocking the event loop.
- Write responsive servers.

✅ Mini Task:

- Write your own script using `setTimeout`, `setImmediate`, and `process.nextTick`.
 - Predict the output before running!
-



Day 7: Callbacks and Promises

✅ Today you'll learn how to handle **asynchronous code** in Node.js using **Callbacks** and **Promises**.

✅ What is a Callback?

- A function **passed as a parameter** to another function.
- It is **called later** when work is done.



Example:

```
js
CopyEdit
function greet(name, callback) {
  console.log('Hello', name);
  callback();
}

greet('Supriya', () => console.log('Goodbye!'));
```

✅ Key point:

A function becomes a callback when you pass it as a parameter and call it inside another function.

✅ Are callbacks always async?

❌ No!

✅ Synchronous callback:

```
js
```


CopyEdit

```
[1, 2, 3].forEach(num => console.log(num));
```

✅ Asynchronous callback:

js

CopyEdit

```
setTimeout(() => console.log('Later!'), 1000);
```

✅ What is "callback hell"?

✅ Example:

js

CopyEdit

```
getUser(id, (user) => {  
  getOrders(user.id, (orders) => {  
    getItems(orders[0], (items) => {  
      console.log(items);  
    });  
  });  
});
```

🤖 Hard to read and maintain.

✅ Promises to the rescue!

✅ A **Promise** is an object representing a **future value**.

✅ States:

- Pending
 - Fulfilled
 - Rejected
-

✅ Basic Promise Example:

js

CopyEdit

```
const myPromise = new Promise((resolve, reject) => {  
  resolve('It worked!');  
});
```

```
myPromise.then(console.log);
```

✓ Output:

```
nginx  
CopyEdit  
It worked!
```

✓ Using setTimeout with a Promise

```
js  
CopyEdit  
const waitTwoSeconds = new Promise((resolve, reject) => {  
  setTimeout(() => {  
    resolve('Finished waiting 2 seconds!');  
  }, 2000);  
});  
  
waitTwoSeconds.then(console.log);
```

✓ Using resolve and reject

```
js  
CopyEdit  
function doubleAsync(num) {  
  return new Promise((resolve, reject) => {  
    if (num) {  
      setTimeout(() => resolve(num * 2), 1000);  
    } else {  
      reject('Please provide a number!');  
    }  
  });  
}  
  
doubleAsync(5)  
  .then(console.log)  
  .catch(console.error);
```

✓ Handles **success** with `.then()`.

✓ Handles **error** with `.catch()`.

✓ Async/Await — Syntactic Sugar for Promises

✓ `async` = returns a Promise automatically.

✓ `await` = pause until Promise resolves.

✓ Simple Example:

```
js
CopyEdit
async function sayHello() {
  return 'Hello!';
}

sayHello().then(console.log);
```

✓ Output:

```
CopyEdit
Hello!
```

✓ Using await:

```
js
CopyEdit
async function run() {
  console.log('Start');
  await wait(2000);
  console.log('After 2 seconds');
}

function wait(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

run();
```

✓ Output:

```
bash
CopyEdit
Start
...wait 2 seconds...
After 2 seconds
```

✓ Using await with try/catch:

```
js
CopyEdit
async function doubleAndLog(num) {
  try{
```

```
const result = await doubleAsync(num);
console.log(result);
} catch (err) {
  console.error(err);
}
}
```

```
doubleAndLog(5);
```

✅ Error automatically goes to catch.

✅ Key differences:

Feature	Promise	Async/Await
Returns	Promise	Promise (automatically)
Handle result	.then()	await
Handle errors	.catch()	try/catch
Code style	Chained	Linear, easy to read

✅ Why use async/await?

- Avoid .then() chains.
 - Looks synchronous.
 - Easier error handling.
-

✅ Mini Task:

Write an async function called tripleAsync that:

- Takes a number
 - Returns a Promise that resolves with number * 3 after 1 second
 - Use await to get the result and log it
-



You've finished:

- **Day 5:** Building a dynamic server
 - **Day 6:** Understanding the event loop
 - **Day 7:** Callbacks and Promises
-

 Next Steps?

- Build more routes.
 - Use Promises everywhere.
 - Refactor to async/await.
 - Connect to databases!
-