

YOUR APP BACKEND FROM SCRATCH



BY STOCARD

About us



Stocard



Lukas



Flo

About you

What are you **interested in** today?

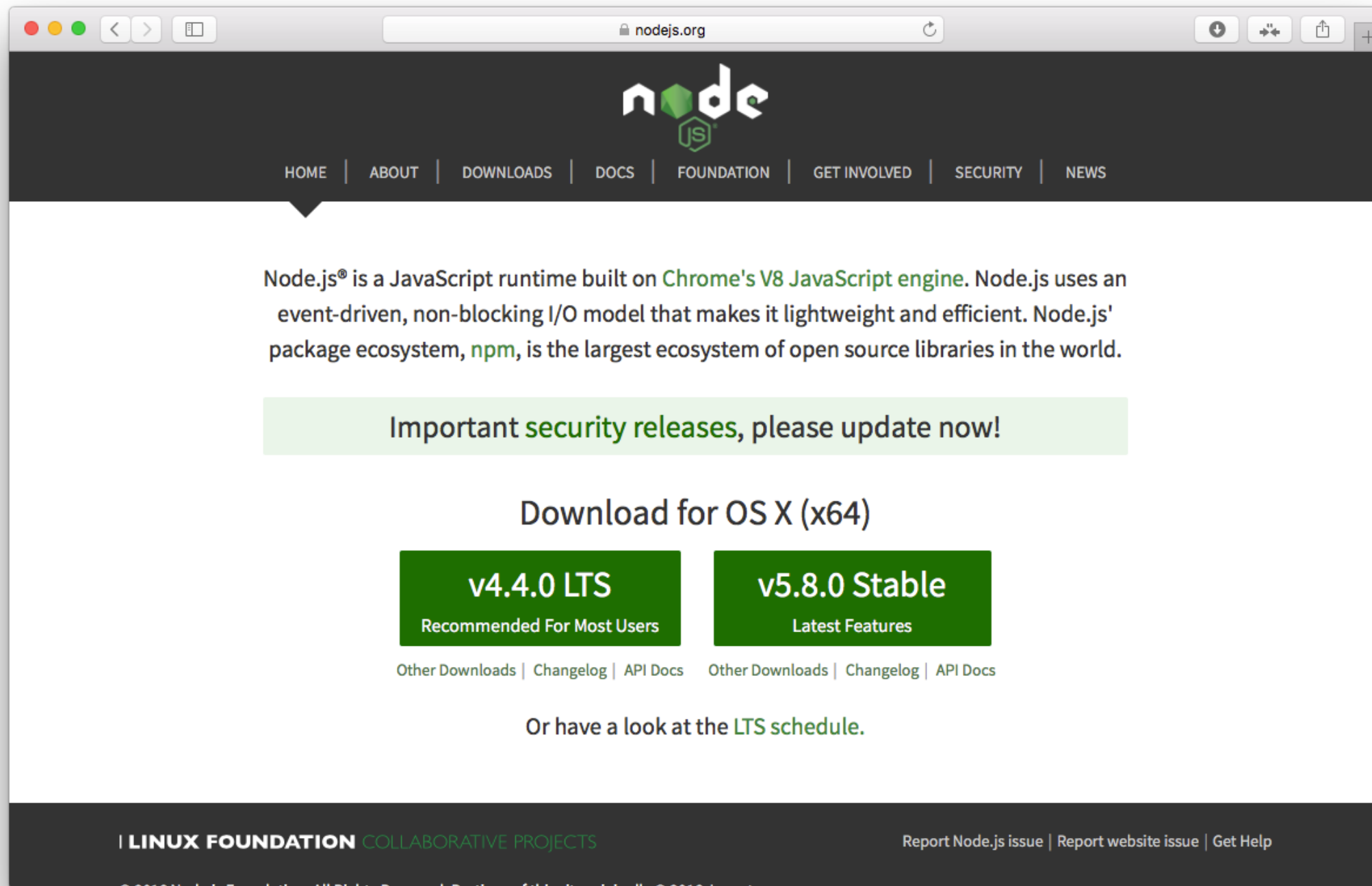
How good are your **programming skills**?

What **programming languages** do you know?

Have you **worked with JavaScript** already?

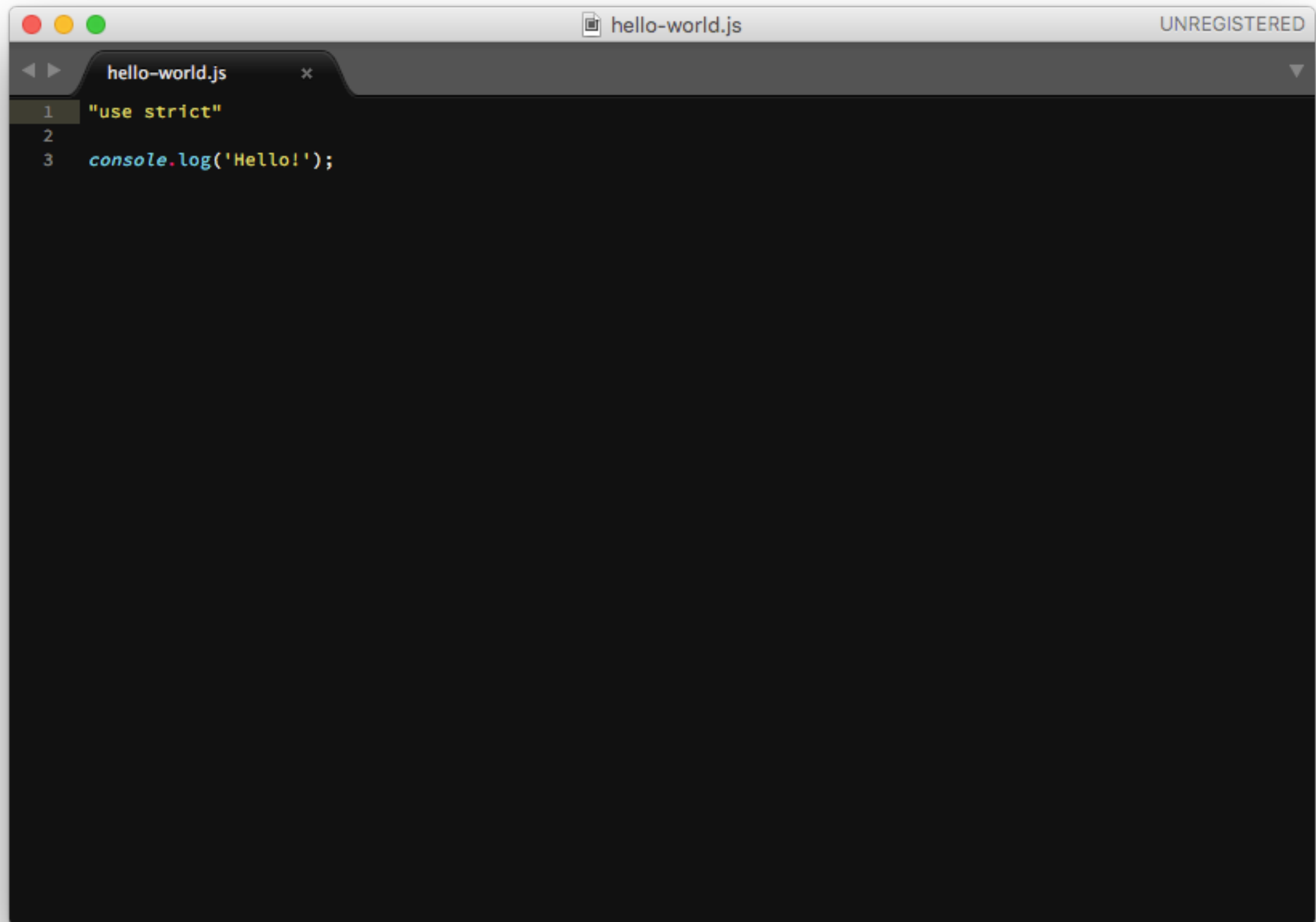
Lets get started!

node.js



<https://nodejs.org>

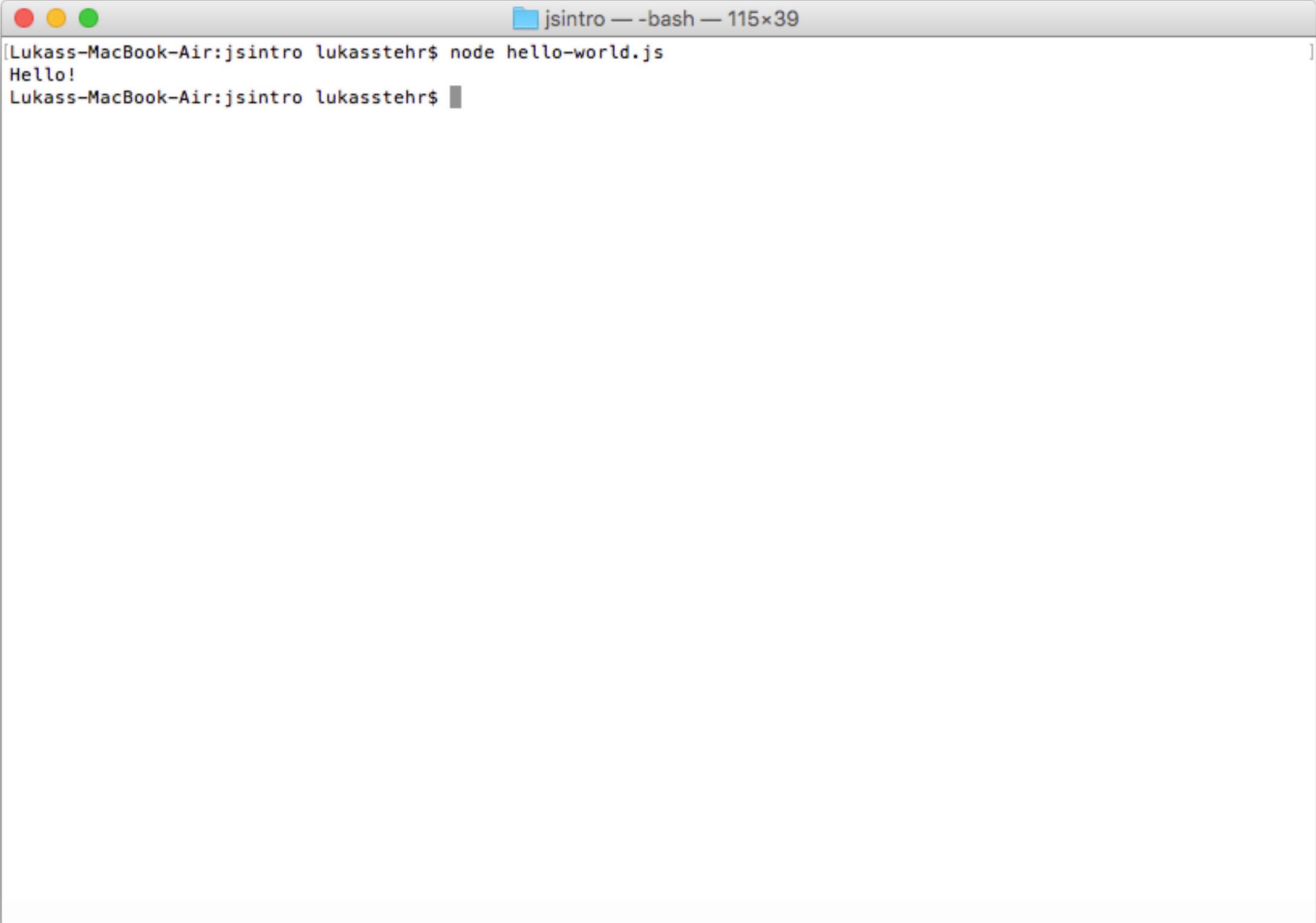
Hello World



A screenshot of a code editor window. The title bar at the top shows three colored window control buttons (red, yellow, green) on the left, the filename 'hello-world.js' in the center, and the word 'UNREGISTERED' on the right. Below the title bar is a tab bar with a single tab labeled 'hello-world.js' and a close button 'x'. The main editing area has a dark background and contains the following code:

```
1  "use strict"  
2  
3  console.log('Hello!');
```

Running the script



```
jsintro — -bash — 115x39
[Lukass-MacBook-Air:jsintro lukasstehr$ node hello-world.js
Hello!
Lukass-MacBook-Air:jsintro lukasstehr$
```

A screenshot of a macOS terminal window. The title bar at the top shows three colored window control buttons (red, yellow, green) on the left, followed by a folder icon and the text "jsintro — -bash — 115x39". The terminal content shows a command prompt where the user has entered "node hello-world.js", resulting in the output "Hello!". The prompt then returns to "Lukas-MacBook-Air:jsintro lukasstehr\$".

Thanks!

Lukas Stehr

stehr@stocard.de

Florian Barth

barth@stocard.de

Introduction

JS



+ 10 days =





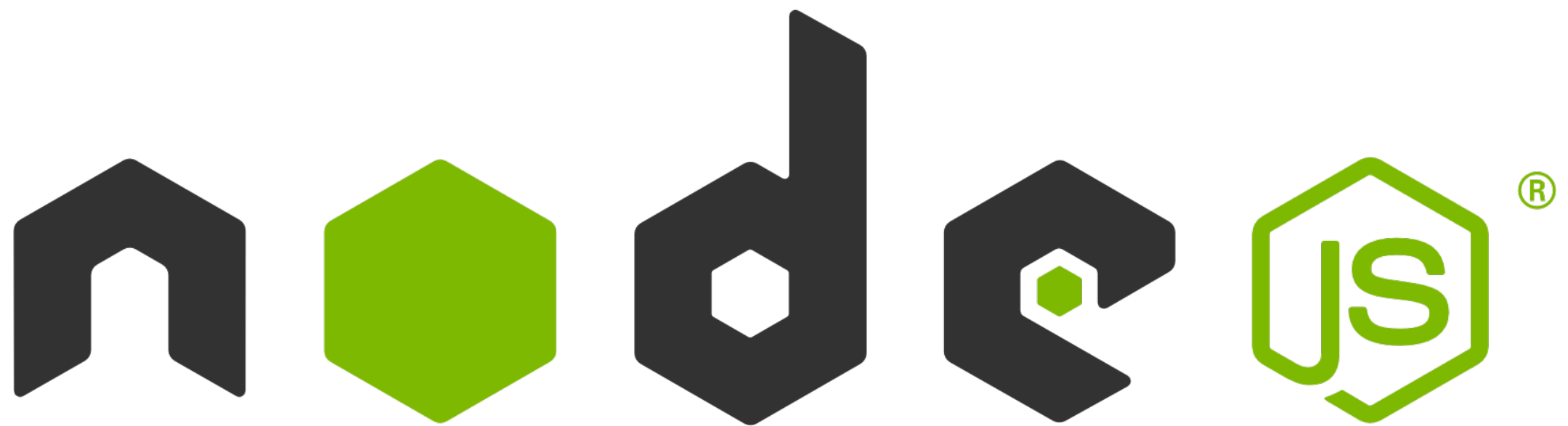
≠



ES6

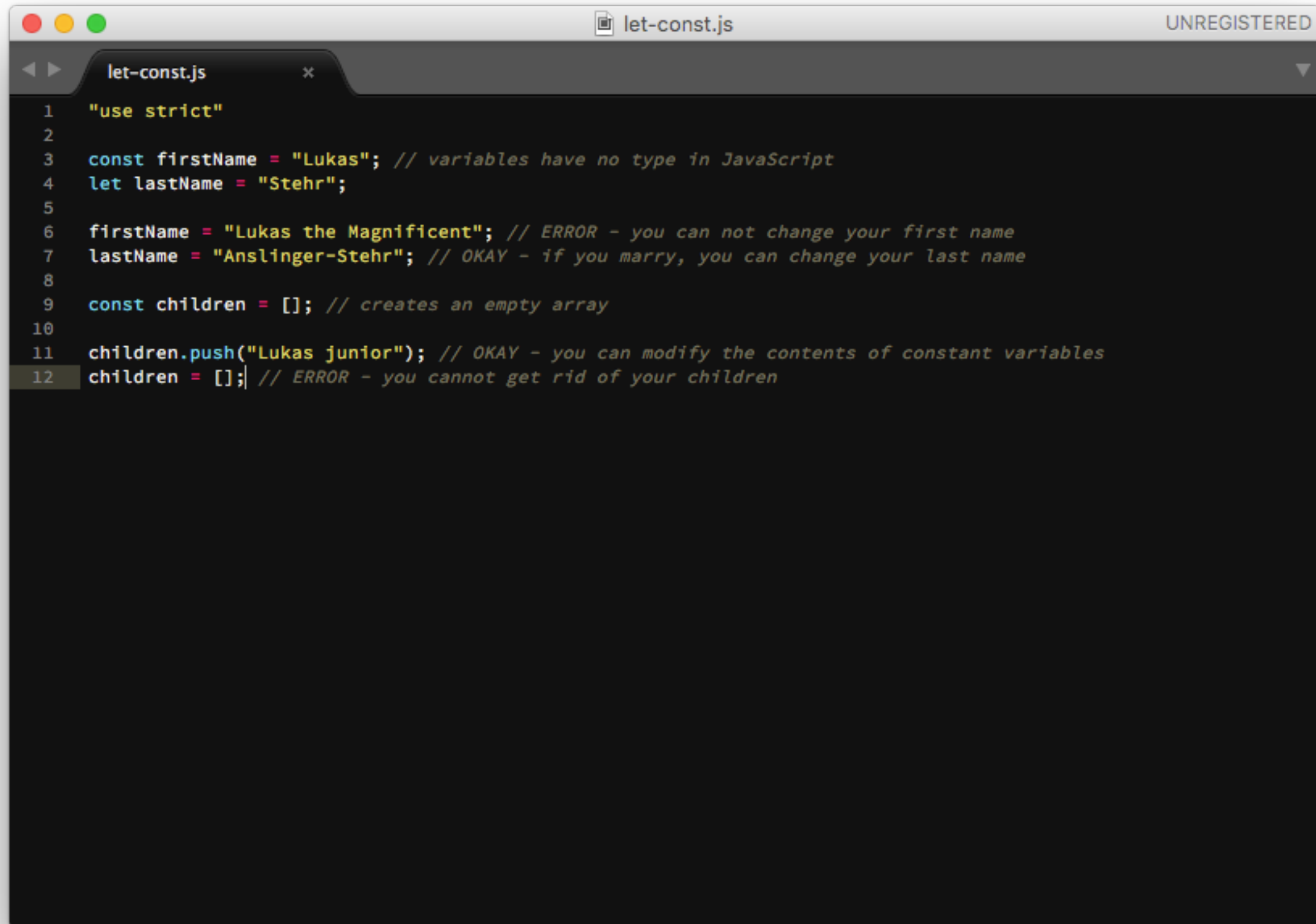
=





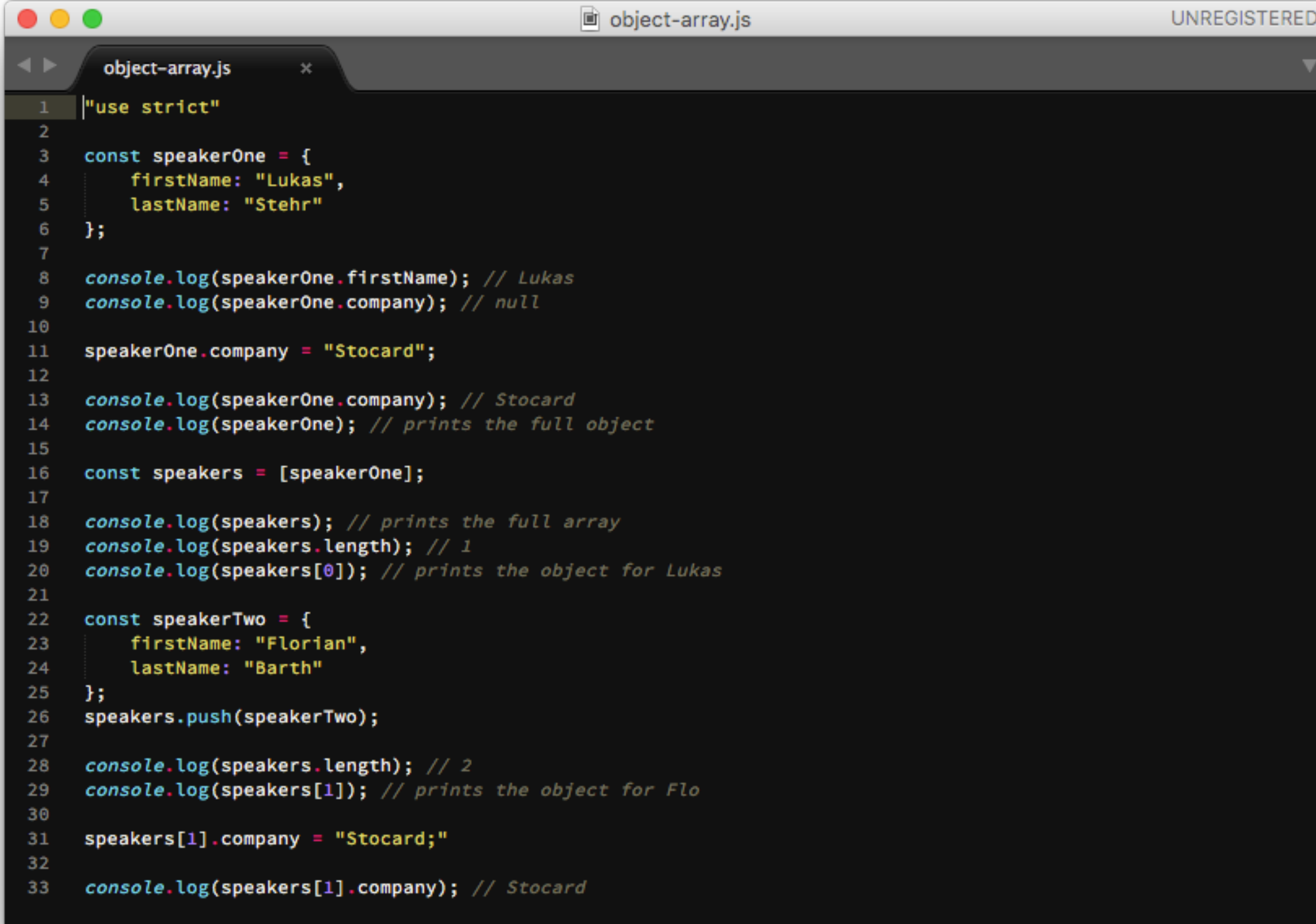
Syntax

let and const



```
1  "use strict"
2
3  const firstName = "Lukas"; // variables have no type in JavaScript
4  let lastName = "Stehr";
5
6  firstName = "Lukas the Magnificent"; // ERROR - you can not change your first name
7  lastName = "Anslinger-Stehr"; // OKAY - if you marry, you can change your last name
8
9  const children = []; // creates an empty array
10
11 children.push("Lukas junior"); // OKAY - you can modify the contents of constant variables
12 children = []; // ERROR - you cannot get rid of your children
```


objects and arrays

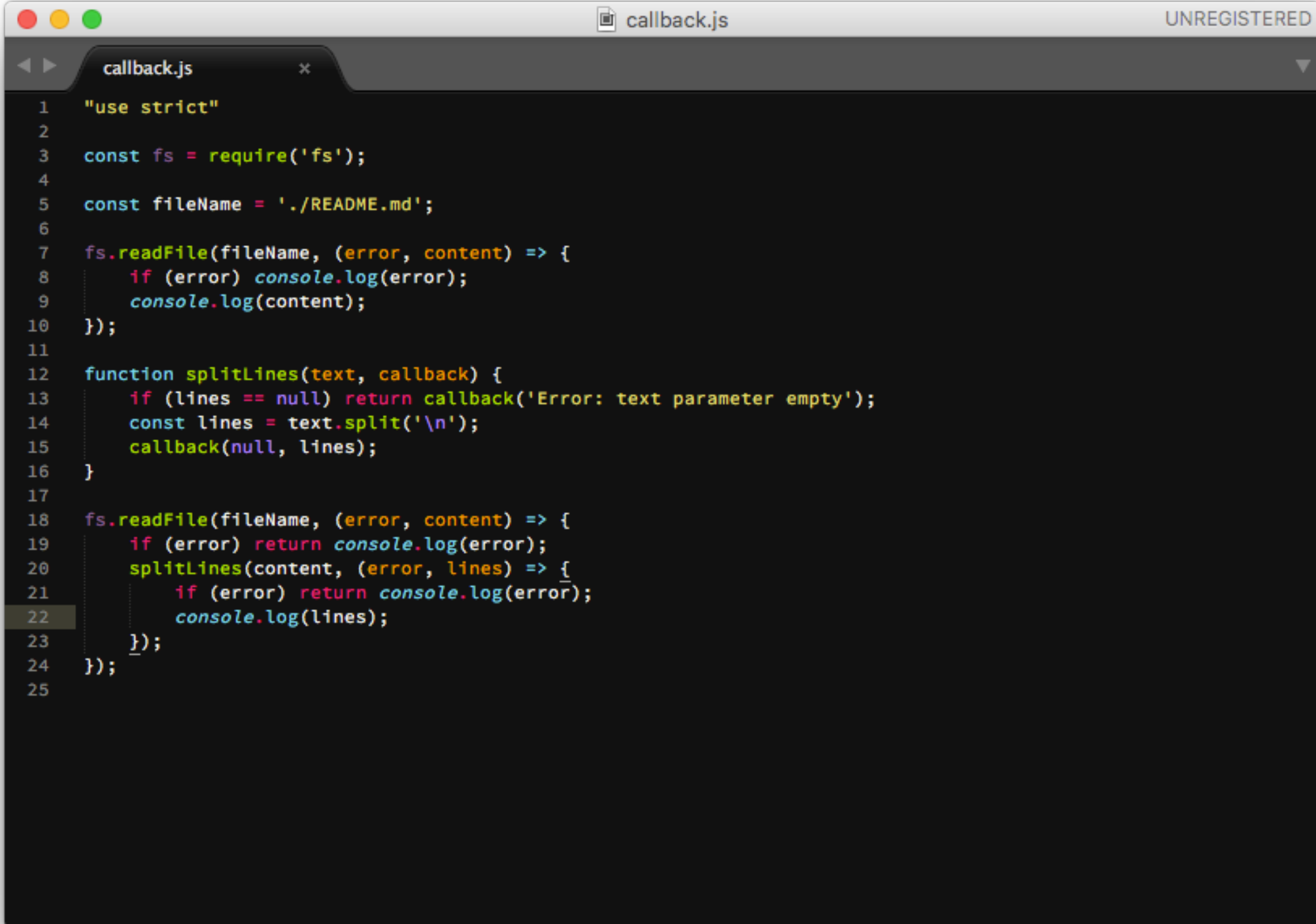


```
1 | "use strict"
2 |
3 | const speakerOne = {
4 |   firstName: "Lukas",
5 |   lastName: "Stehr"
6 | };
7 |
8 | console.log(speakerOne.firstName); // Lukas
9 | console.log(speakerOne.company); // null
10 |
11 | speakerOne.company = "Stocard";
12 |
13 | console.log(speakerOne.company); // Stocard
14 | console.log(speakerOne); // prints the full object
15 |
16 | const speakers = [speakerOne];
17 |
18 | console.log(speakers); // prints the full array
19 | console.log(speakers.length); // 1
20 | console.log(speakers[0]); // prints the object for Lukas
21 |
22 | const speakerTwo = {
23 |   firstName: "Florian",
24 |   lastName: "Barth"
25 | };
26 | speakers.push(speakerTwo);
27 |
28 | console.log(speakers.length); // 2
29 | console.log(speakers[1]); // prints the object for Flo
30 |
31 | speakers[1].company = "Stocard";
32 |
33 | console.log(speakers[1].company); // Stocard
```

Asynchronicity

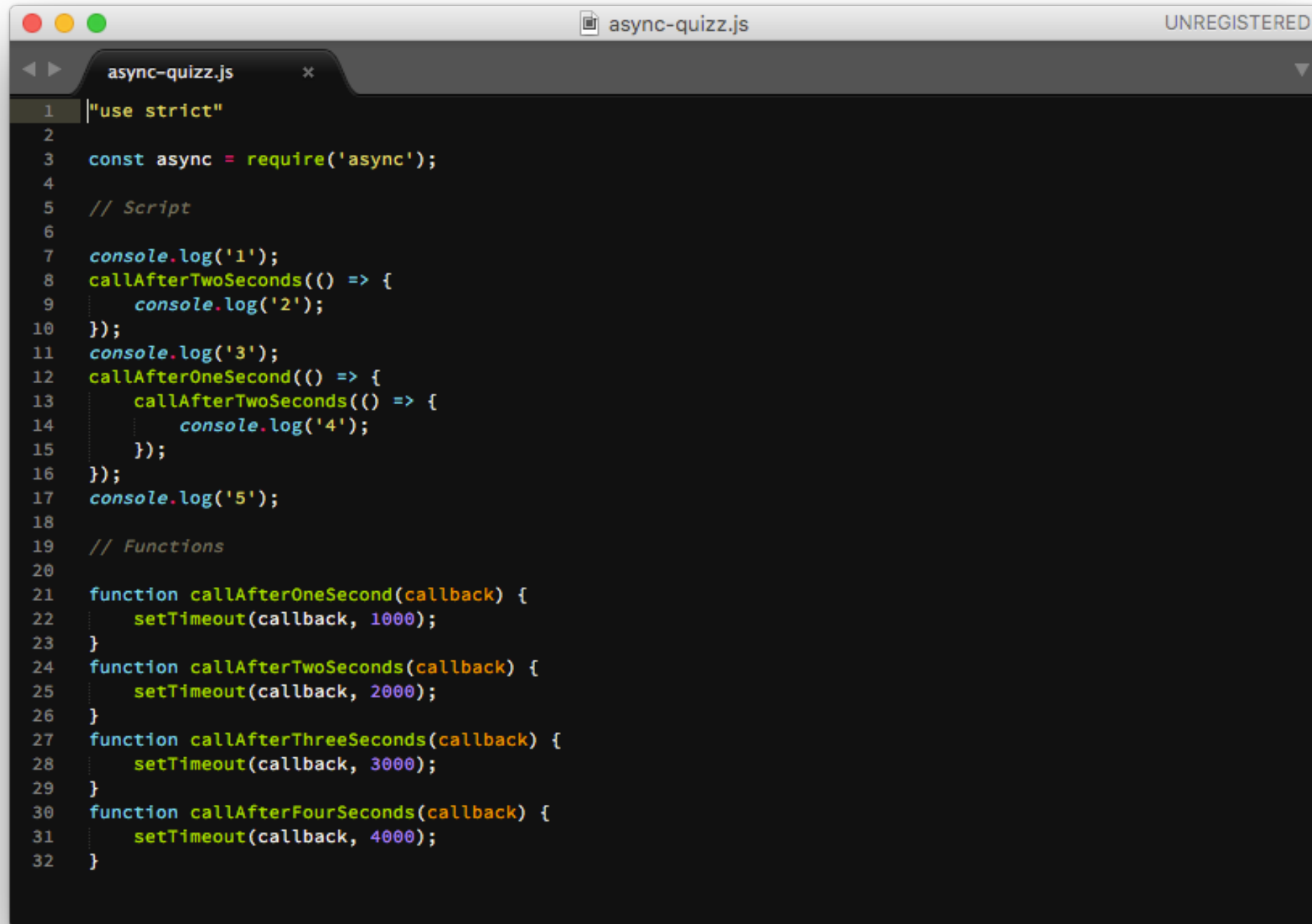
single-threaded
callback
non-blocking
asynchronicity
async *event-loop*
performant
interrupt
parallel

callbacks



```
1  "use strict"
2
3  const fs = require('fs');
4
5  const fileName = './README.md';
6
7  fs.readFile(fileName, (error, content) => {
8    if (error) console.log(error);
9    console.log(content);
10 });
11
12 function splitLines(text, callback) {
13   if (lines == null) return callback('Error: text parameter empty');
14   const lines = text.split('\n');
15   callback(null, lines);
16 }
17
18 fs.readFile(fileName, (error, content) => {
19   if (error) return console.log(error);
20   splitLines(content, (error, lines) => {
21     if (error) return console.log(error);
22     console.log(lines);
23   });
24 });
25
```

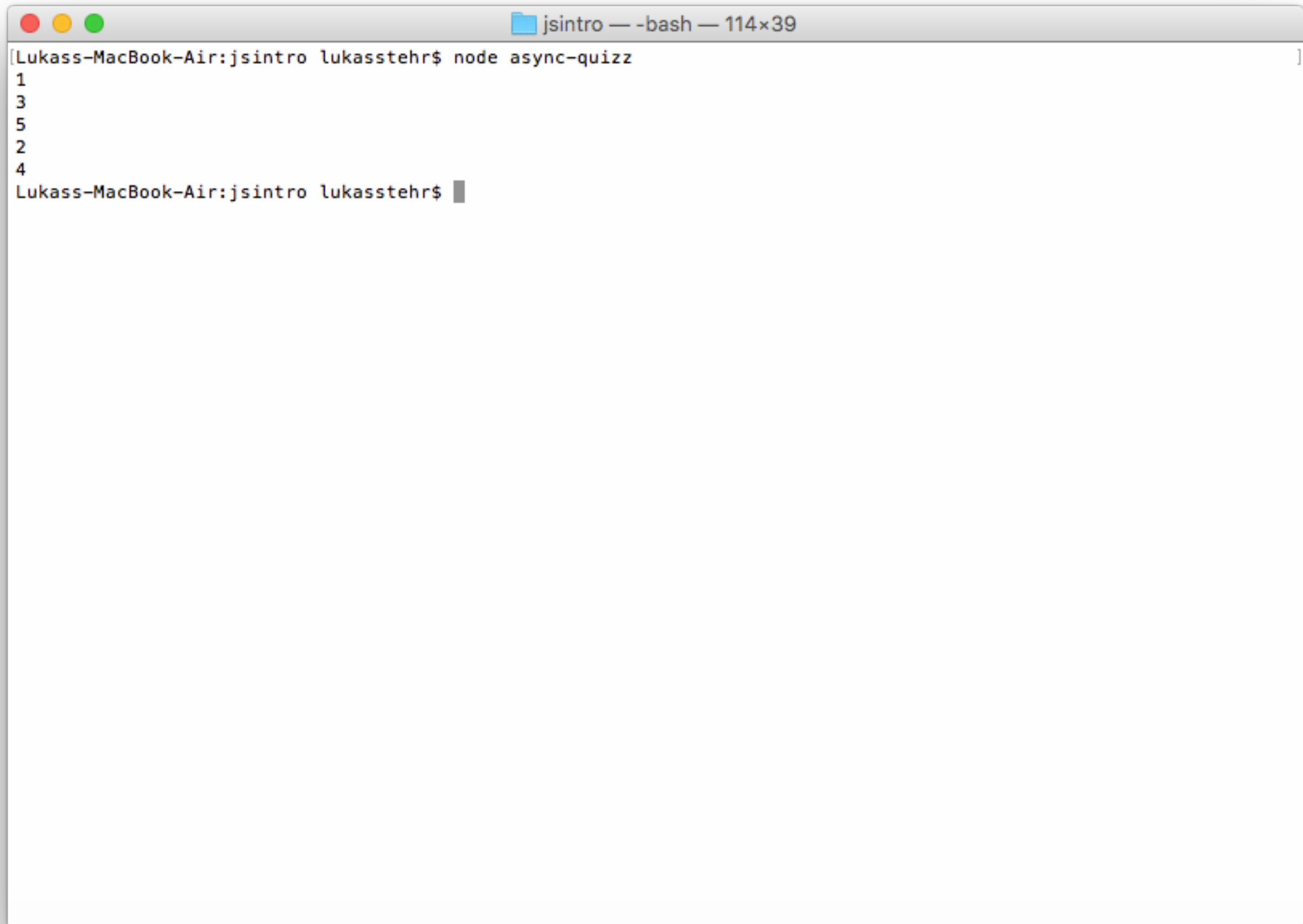
Async quiz

A screenshot of a code editor window titled 'async-quizz.js' with a status bar indicating 'UNREGISTERED'. The code is written in JavaScript and includes a 'use strict' directive, a require statement for 'async', and several console.log statements and setTimeout calls. The code is as follows:

```
1 "use strict"
2
3 const async = require('async');
4
5 // Script
6
7 console.log('1');
8 callAfterTwoSeconds(() => {
9   console.log('2');
10 });
11 console.log('3');
12 callAfterOneSecond(() => {
13   callAfterTwoSeconds(() => {
14     console.log('4');
15   });
16 });
17 console.log('5');
18
19 // Functions
20
21 function callAfterOneSecond(callback) {
22   setTimeout(callback, 1000);
23 }
24 function callAfterTwoSeconds(callback) {
25   setTimeout(callback, 2000);
26 }
27 function callAfterThreeSeconds(callback) {
28   setTimeout(callback, 3000);
29 }
30 function callAfterFourSeconds(callback) {
31   setTimeout(callback, 4000);
32 }
```

What is printed to the console?

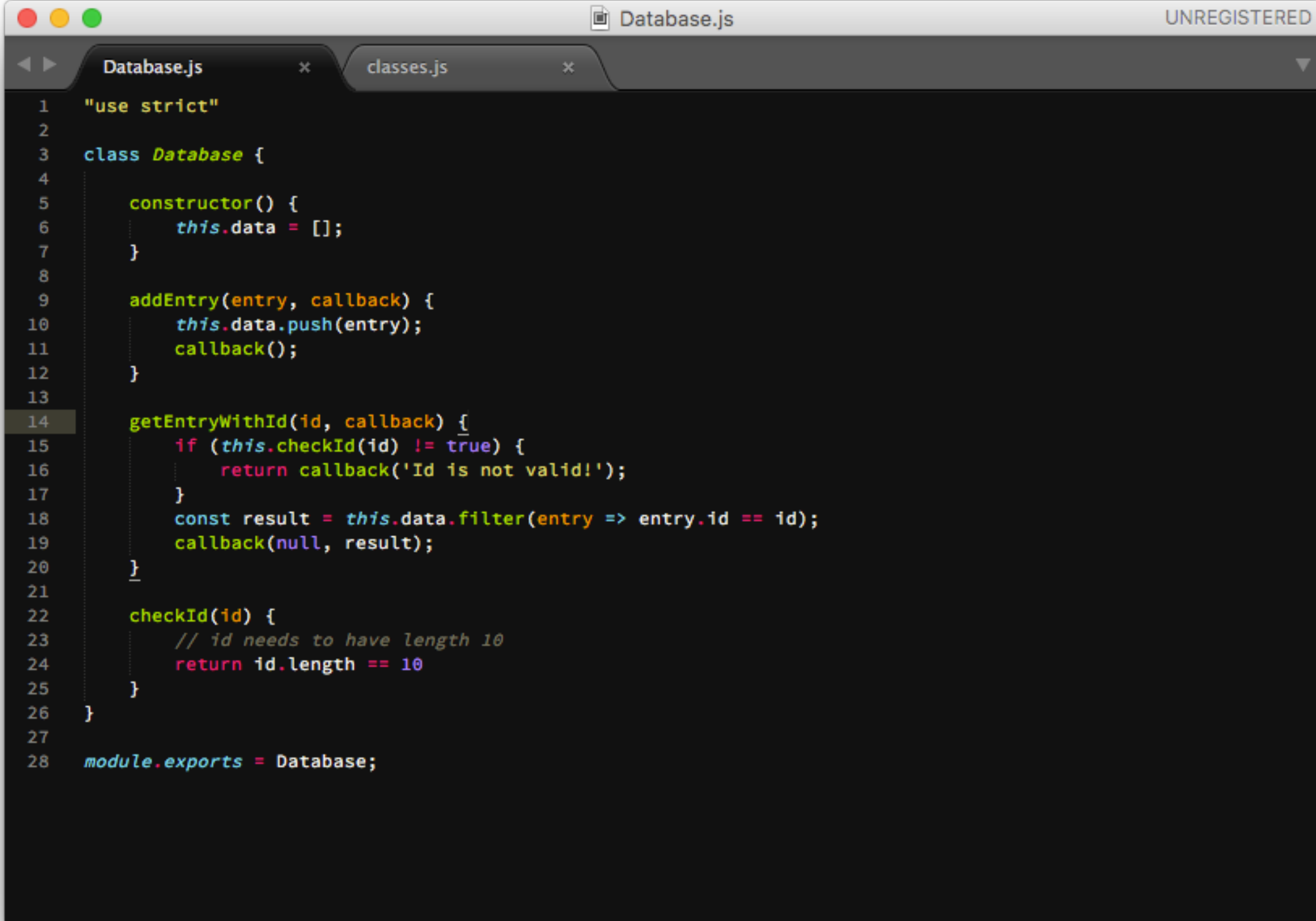
Async quiz



```
jsintro — -bash — 114x39
[Lukass-MacBook-Air:jsintro lukassteahr$ node async-quizz
1
3
5
2
4
Lukass-MacBook-Air:jsintro lukassteahr$
```

A terminal window titled "jsintro — -bash — 114x39" on a Mac. The prompt is "[Lukass-MacBook-Air:jsintro lukassteahr\$]". The command "node async-quizz" has been executed, resulting in the output: "1", "3", "5", "2", and "4" on separate lines. The prompt is now "Lukass-MacBook-Air:jsintro lukassteahr\$".

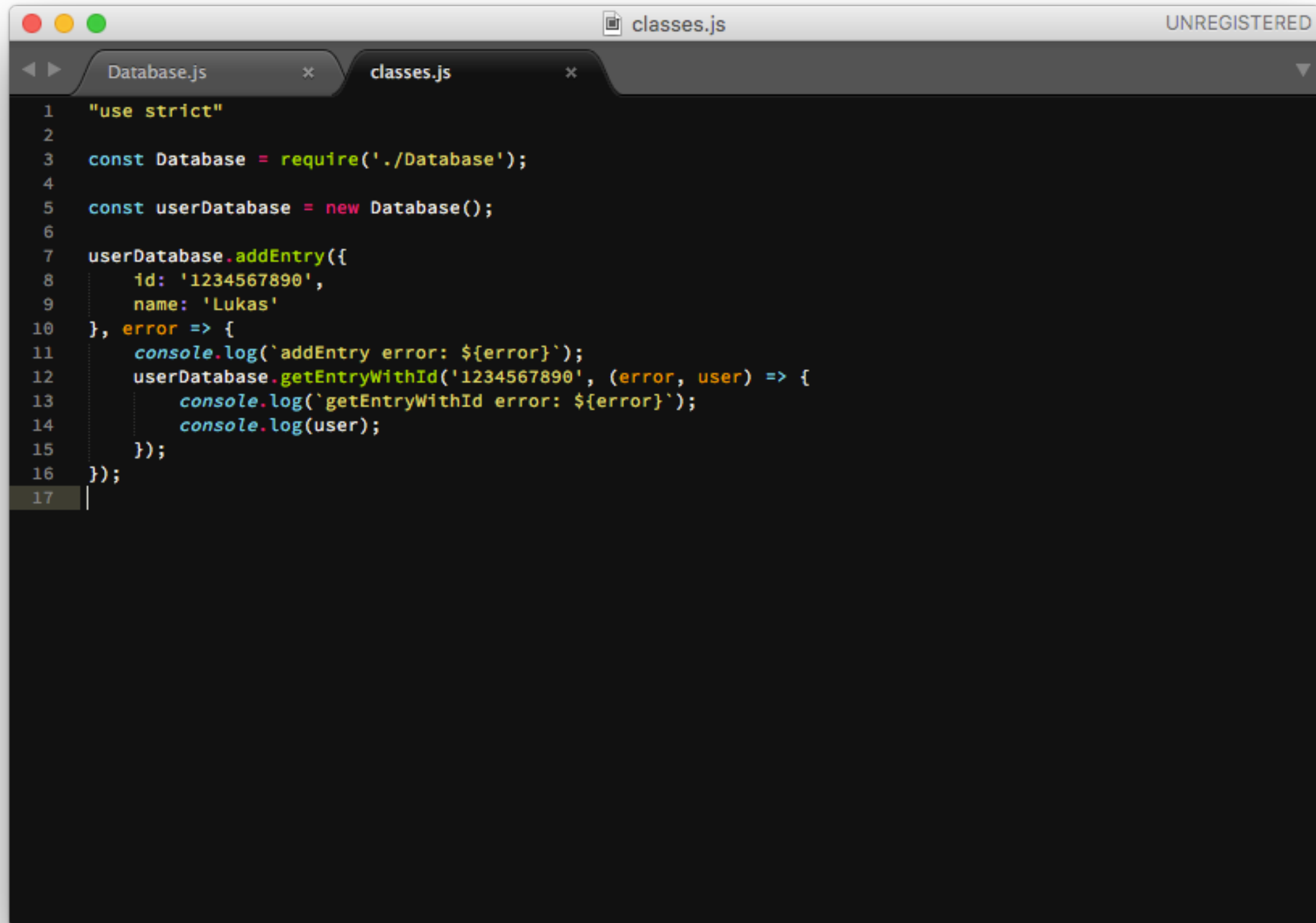
class syntax



The image shows a code editor window with a dark theme. The title bar at the top indicates the file is 'Database.js' and the status is 'UNREGISTERED'. The editor has two tabs: 'Database.js' (active) and 'classes.js'. The code in 'Database.js' is as follows:

```
1  "use strict"
2
3  class Database {
4
5      constructor() {
6          this.data = [];
7      }
8
9      addEntry(entry, callback) {
10         this.data.push(entry);
11         callback();
12     }
13
14     getEntryWithId(id, callback) {
15         if (this.checkId(id) !== true) {
16             return callback('Id is not valid!');
17         }
18         const result = this.data.filter(entry => entry.id == id);
19         callback(null, result);
20     }
21
22     checkId(id) {
23         // id needs to have length 10
24         return id.length == 10
25     }
26 }
27
28 module.exports = Database;
```

class syntax



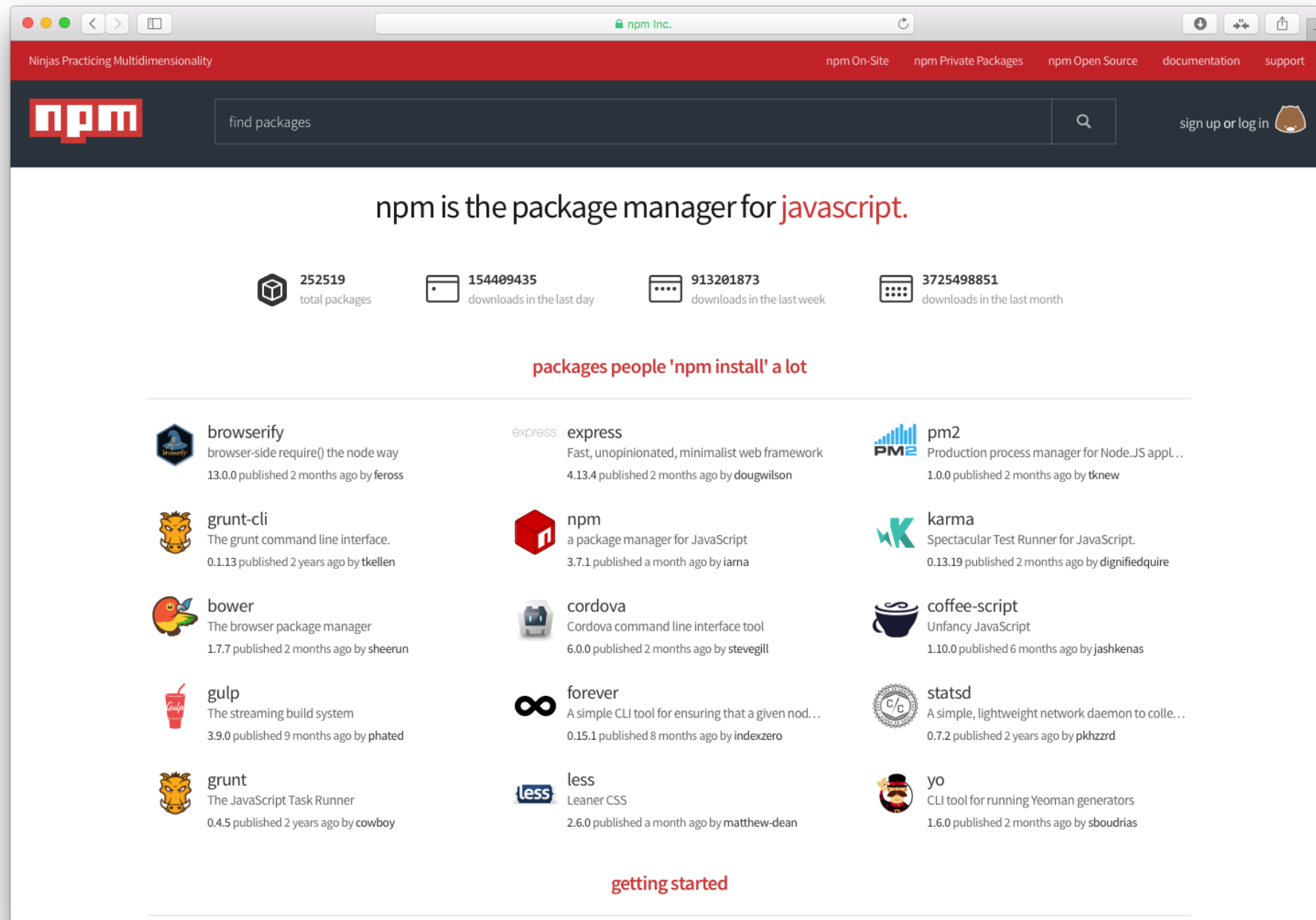
The image shows a code editor window with two tabs: 'Database.js' and 'classes.js'. The 'classes.js' tab is active, displaying the following JavaScript code:

```
1  "use strict"
2
3  const Database = require('./Database');
4
5  const userDatabase = new Database();
6
7  userDatabase.addEntry({
8      id: '1234567890',
9      name: 'Lukas'
10 }, error => {
11     console.log(`addEntry error: ${error}`);
12     userDatabase.getEntryWithId('1234567890', (error, user) => {
13         console.log(`getEntryWithId error: ${error}`);
14         console.log(user);
15     });
16 });
17
```

The code demonstrates the use of the 'class' syntax in JavaScript. It starts with 'use strict', then imports a 'Database' module. A 'userDatabase' instance is created using 'new Database()'. The 'addEntry' method is called with an object containing 'id' and 'name' properties. The code uses arrow functions for callbacks and includes console logs for error handling.

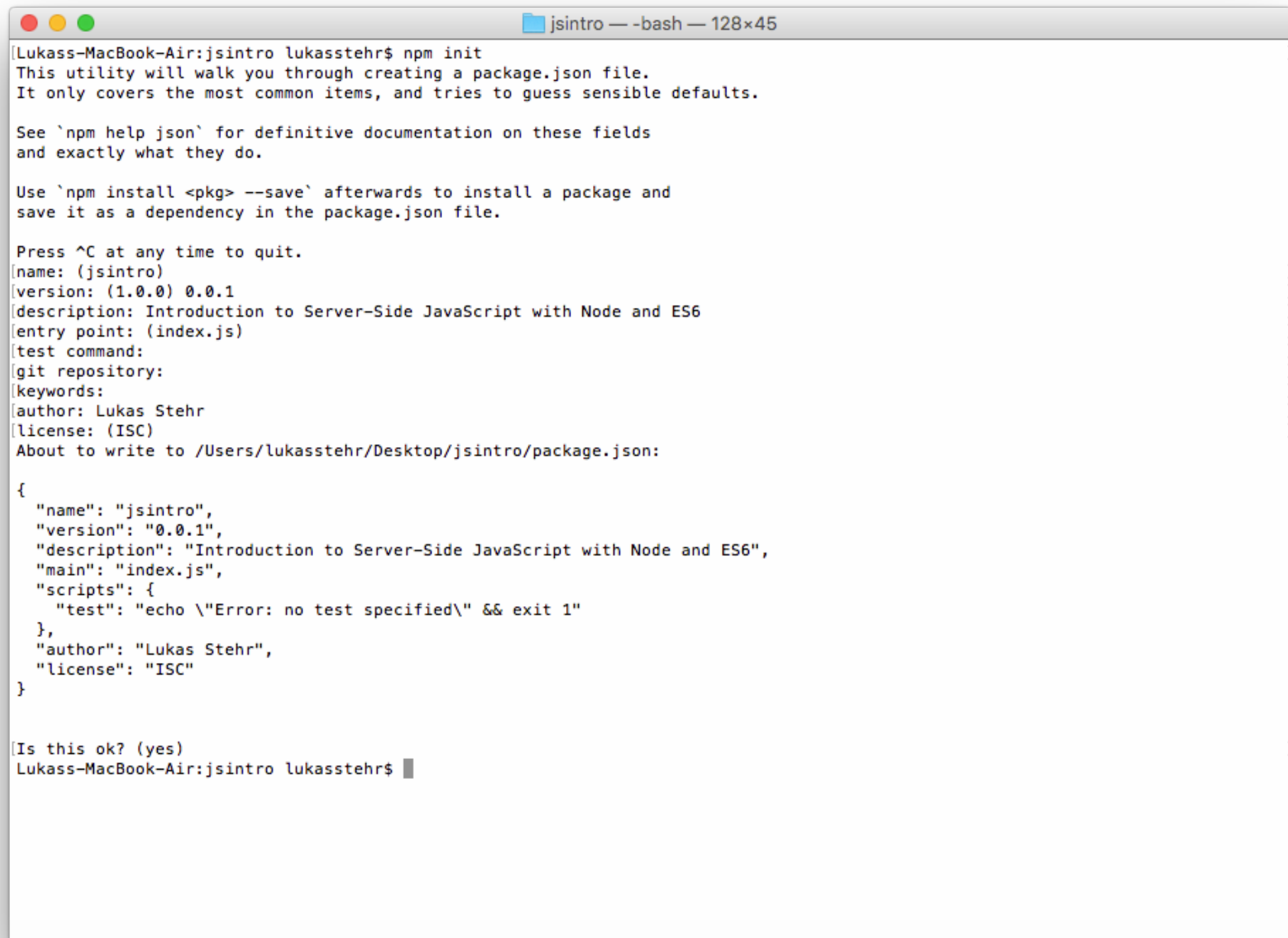
Libraries

NPM



<https://npmjs.com>

npm init



```
[Lukass-MacBook-Air:jsintro lukasstehr$ npm init]
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

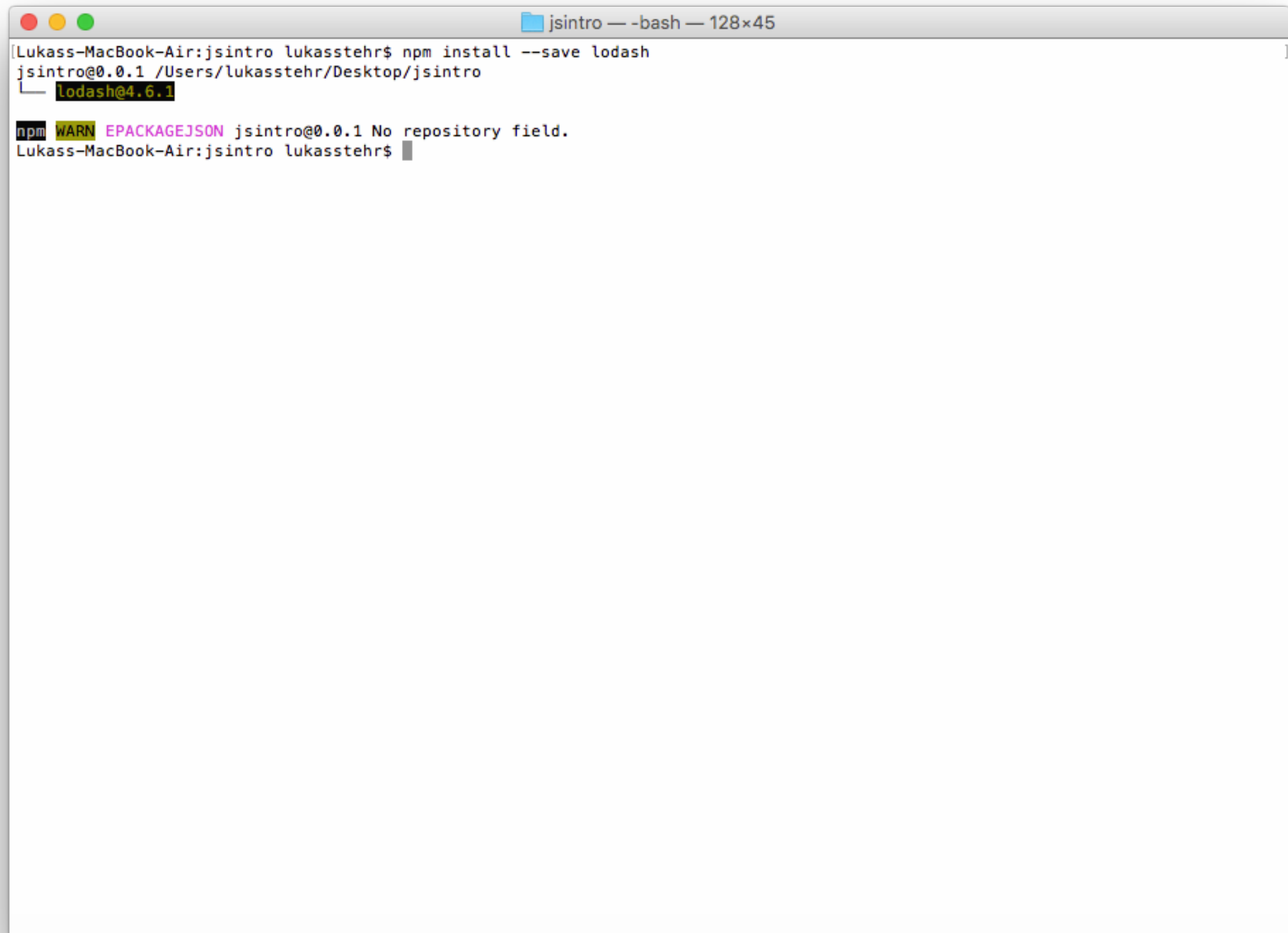
Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
[name: (jsintro)]
[version: (1.0.0) 0.0.1]
[description: Introduction to Server-Side JavaScript with Node and ES6]
[entry point: (index.js)]
[test command:]
[git repository:]
[keywords:]
[author: Lukas Stehr]
[license: (ISC)]
About to write to /Users/lukasstehr/Desktop/jsintro/package.json:

{
  "name": "jsintro",
  "version": "0.0.1",
  "description": "Introduction to Server-Side JavaScript with Node and ES6",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Lukas Stehr",
  "license": "ISC"
}

[Is this ok? (yes)]
[Lukass-MacBook-Air:jsintro lukasstehr$ ]
```

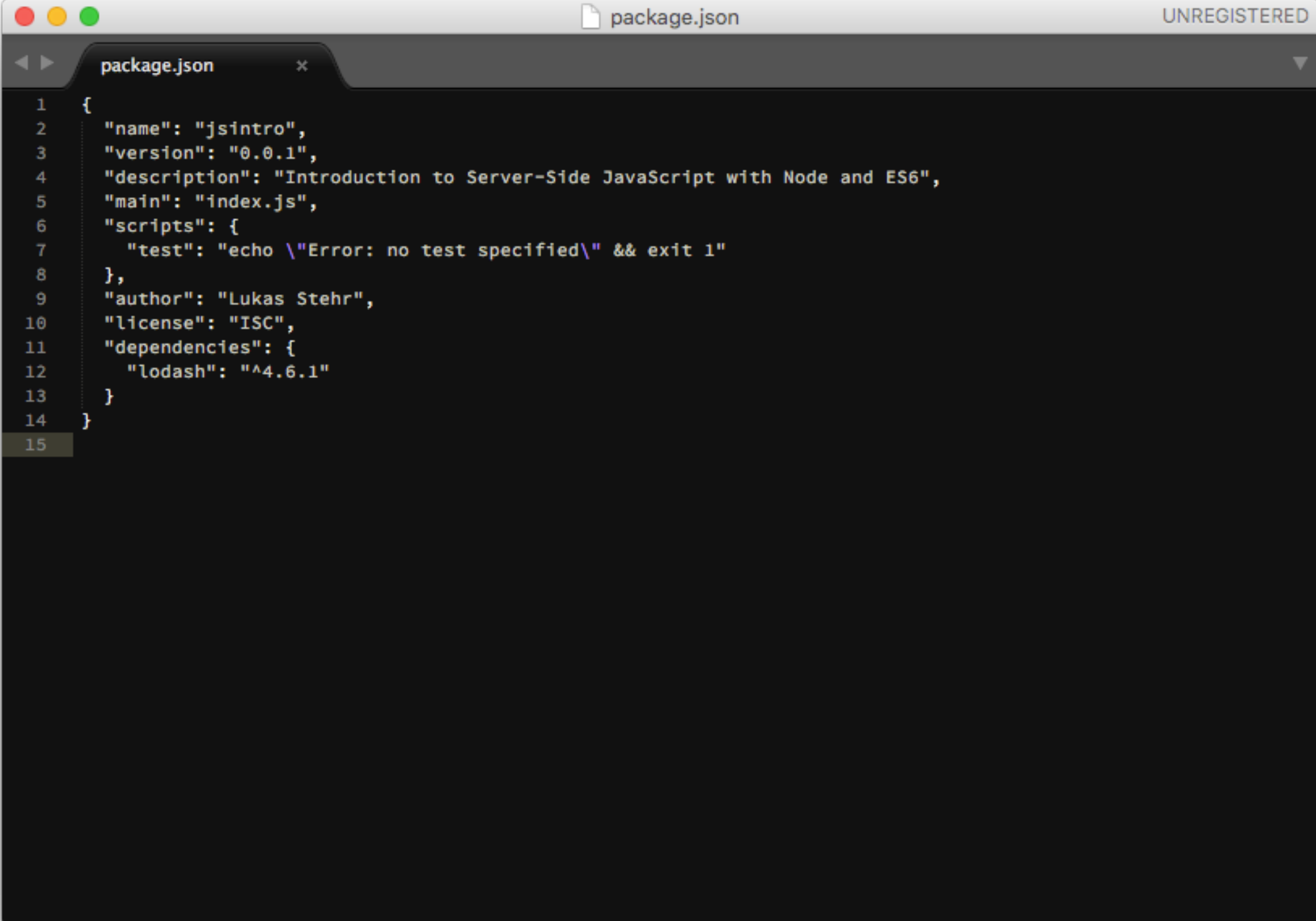
npm install



```
jsintro — -bash — 128x45
[Lukass-MacBook-Air:jsintro lukasstehr$ npm install --save lodash
jsintro@0.0.1 /Users/lukasstehr/Desktop/jsintro
└─┬─> lodash@4.6.1

npm WARN EPACKAGEJSON jsintro@0.0.1 No repository field.
Lukass-MacBook-Air:jsintro lukasstehr$
```

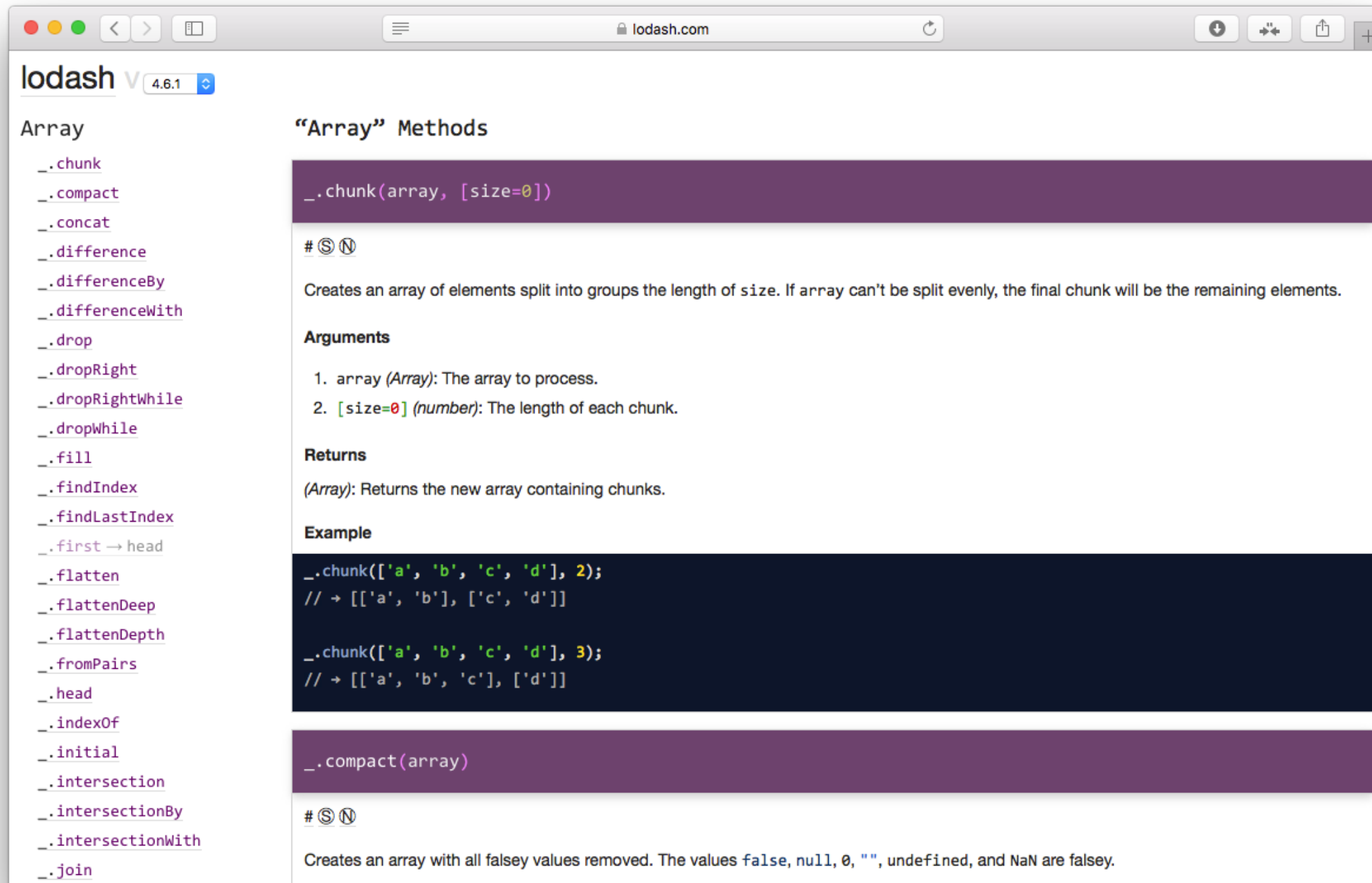
package.json



A screenshot of a code editor window titled "package.json" with a tab labeled "package.json" and a close button. The editor shows the following JSON content:

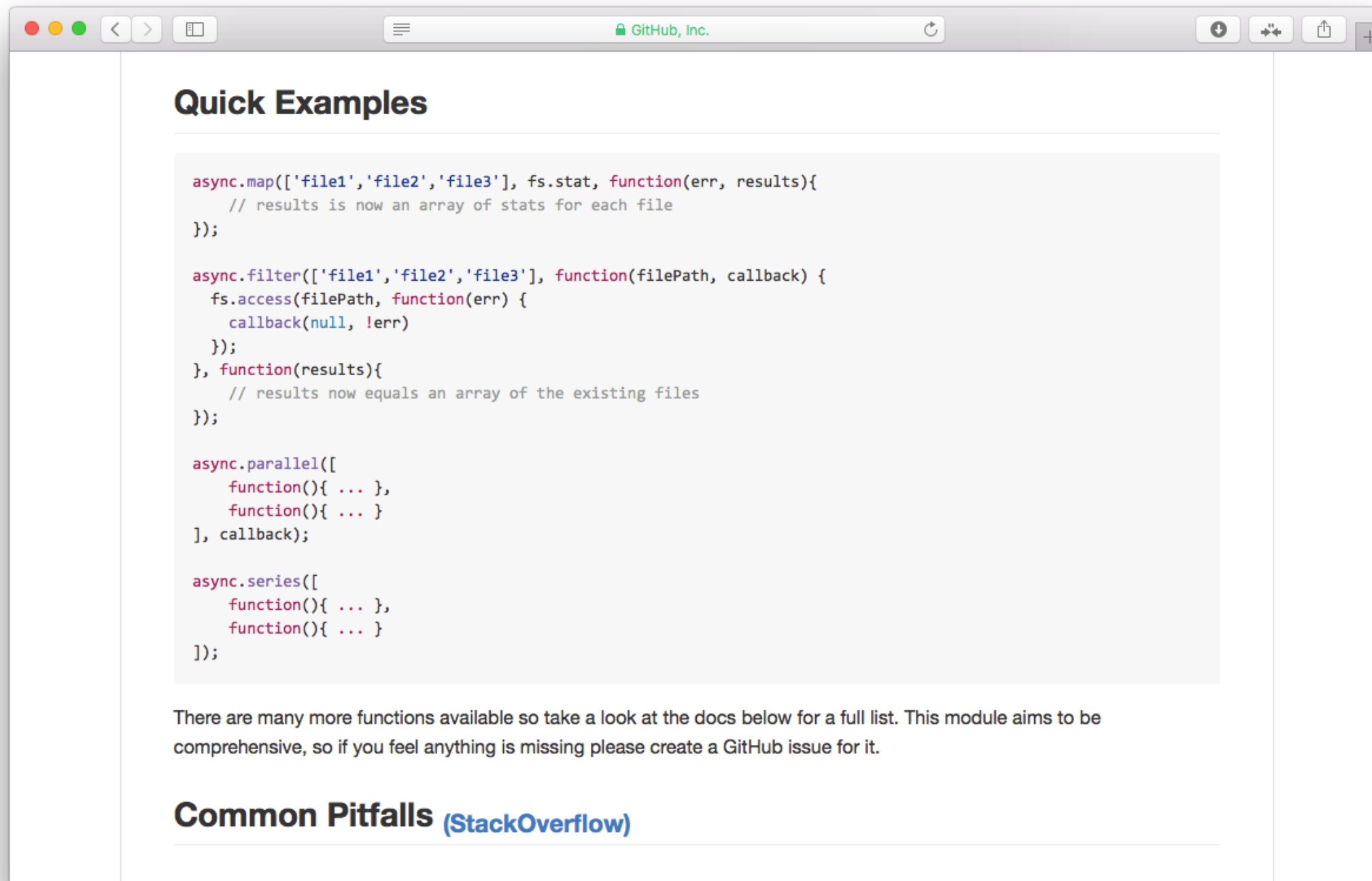
```
1 {
2   "name": "jsintro",
3   "version": "0.0.1",
4   "description": "Introduction to Server-Side JavaScript with Node and ES6",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "Lukas Stehr",
10  "license": "ISC",
11  "dependencies": {
12    "lodash": "^4.6.1"
13  }
14 }
15
```

Lodash



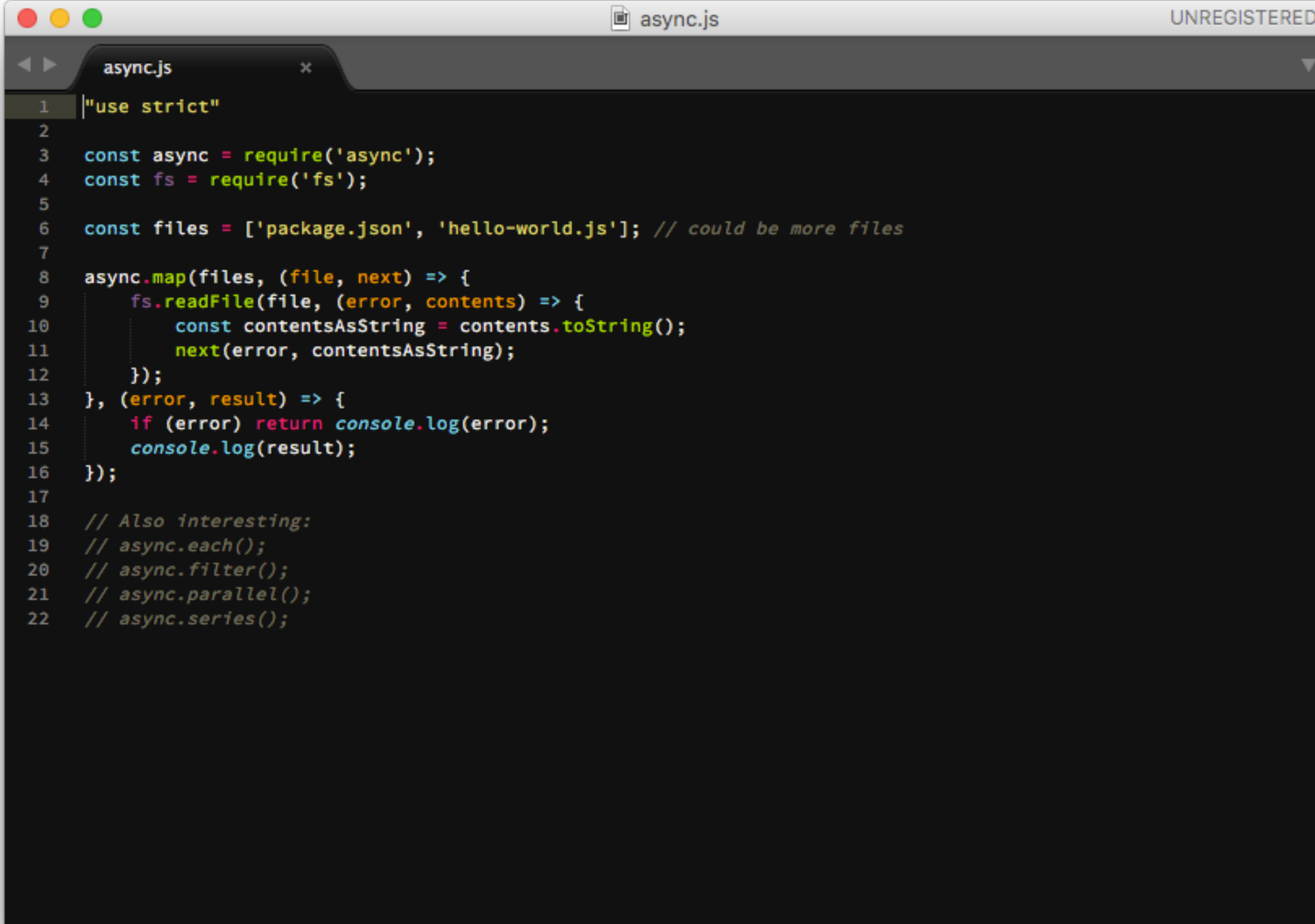
<https://lodash.com>

async



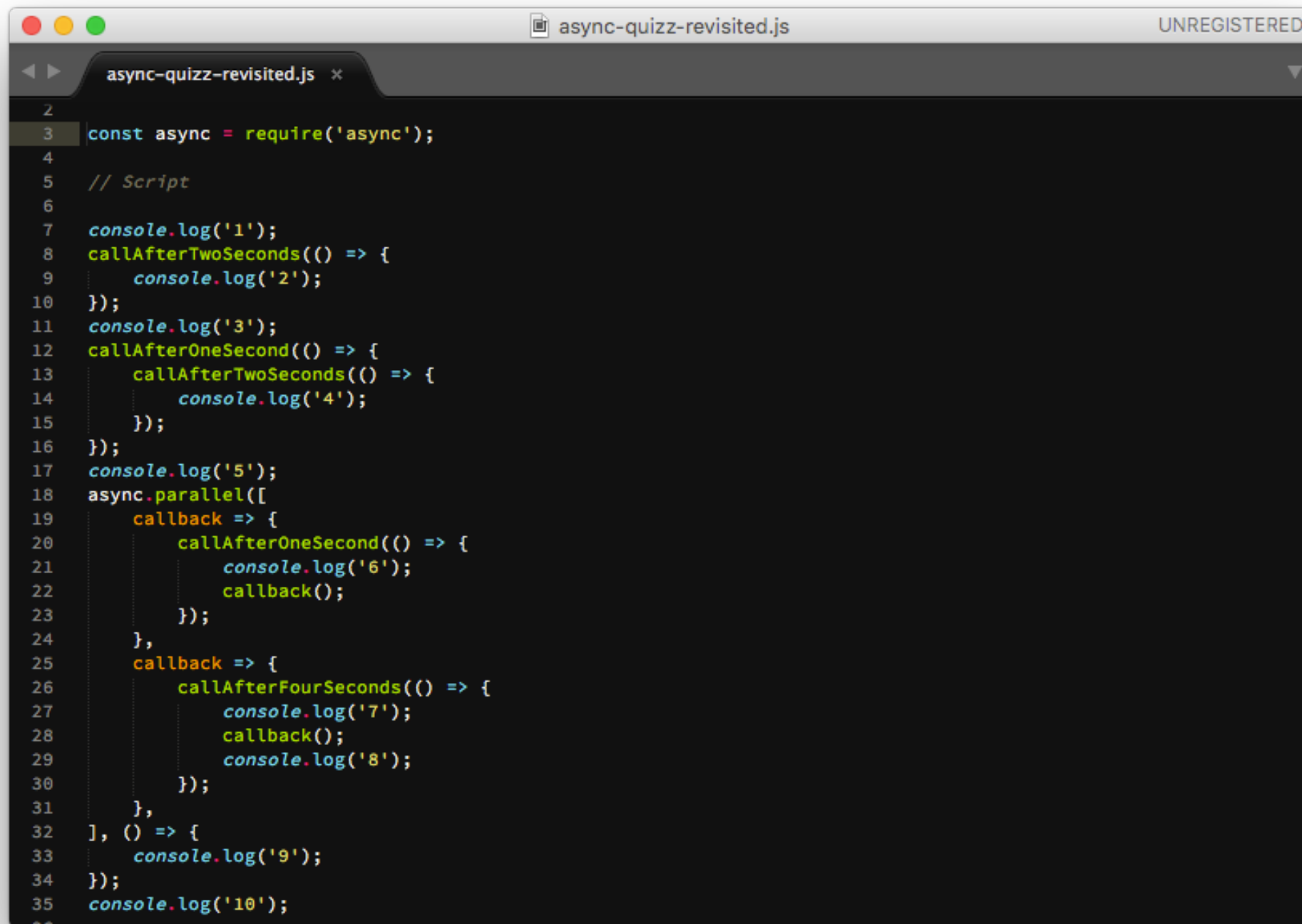
<https://github.com/caolan/async>

Using async



```
1 "use strict"
2
3 const async = require('async');
4 const fs = require('fs');
5
6 const files = ['package.json', 'hello-world.js']; // could be more files
7
8 async.map(files, (file, next) => {
9   fs.readFile(file, (error, contents) => {
10     const contentsAsString = contents.toString();
11     next(error, contentsAsString);
12   });
13 }, (error, result) => {
14   if (error) return console.log(error);
15   console.log(result);
16 });
17
18 // Also interesting:
19 // async.each();
20 // async.filter();
21 // async.parallel();
22 // async.series();
```

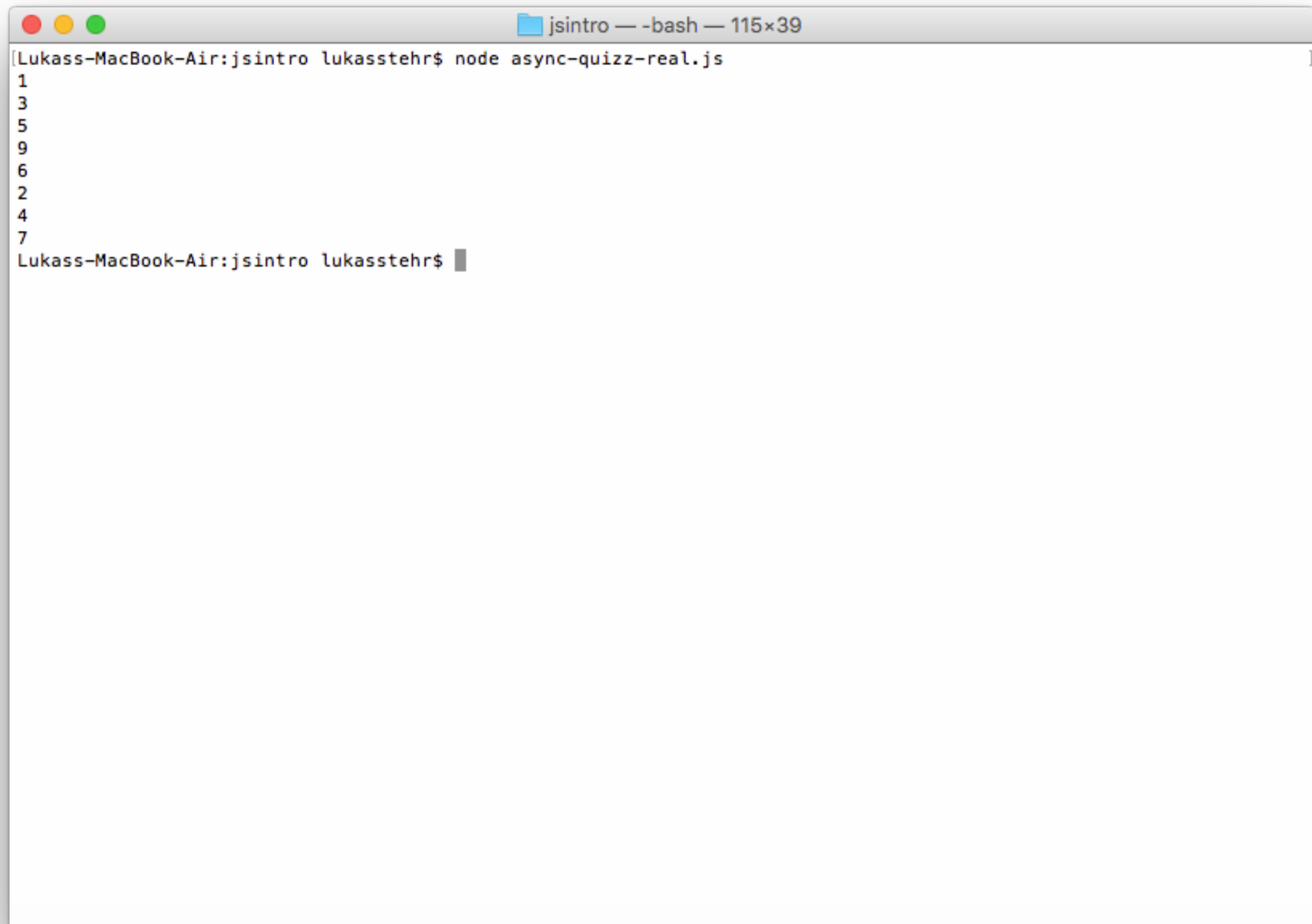
Async quiz revisited



```
2
3 const async = require('async');
4
5 // Script
6
7 console.log('1');
8 callAfterTwoSeconds(() => {
9   console.log('2');
10 });
11 console.log('3');
12 callAfterOneSecond(() => {
13   callAfterTwoSeconds(() => {
14     console.log('4');
15   });
16 });
17 console.log('5');
18 async.parallel([
19   callback => {
20     callAfterOneSecond(() => {
21       console.log('6');
22       callback();
23     });
24   },
25   callback => {
26     callAfterFourSeconds(() => {
27       console.log('7');
28       callback();
29       console.log('8');
30     });
31   },
32 ], () => {
33   console.log('9');
34 });
35 console.log('10');
```

What is printed to the console?

Async quiz revisited

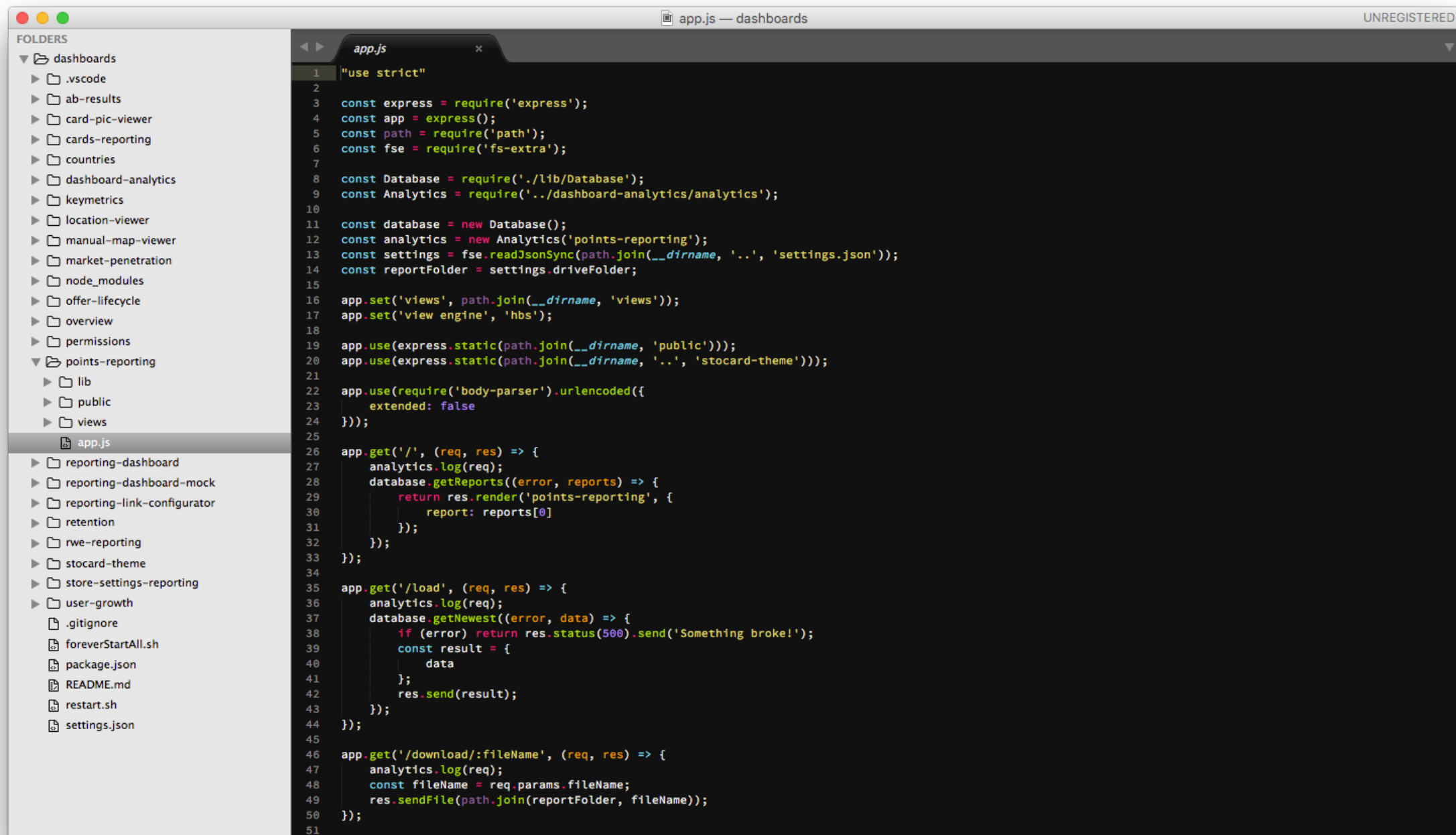


```
jsintro — -bash — 115x39
[Lukass-MacBook-Air:jsintro lukasstehr$ node async-quizz-real.js
1
3
5
9
6
2
4
7
Lukass-MacBook-Air:jsintro lukasstehr$
```

A terminal window titled "jsintro — -bash — 115x39" showing the execution of a Node.js script. The prompt is "[Lukass-MacBook-Air:jsintro lukasstehr\$ node async-quizz-real.js]". The script outputs the numbers 1, 3, 5, 9, 6, 2, 4, and 7, each on a new line. The terminal window has a standard macOS title bar with red, yellow, and green window control buttons.

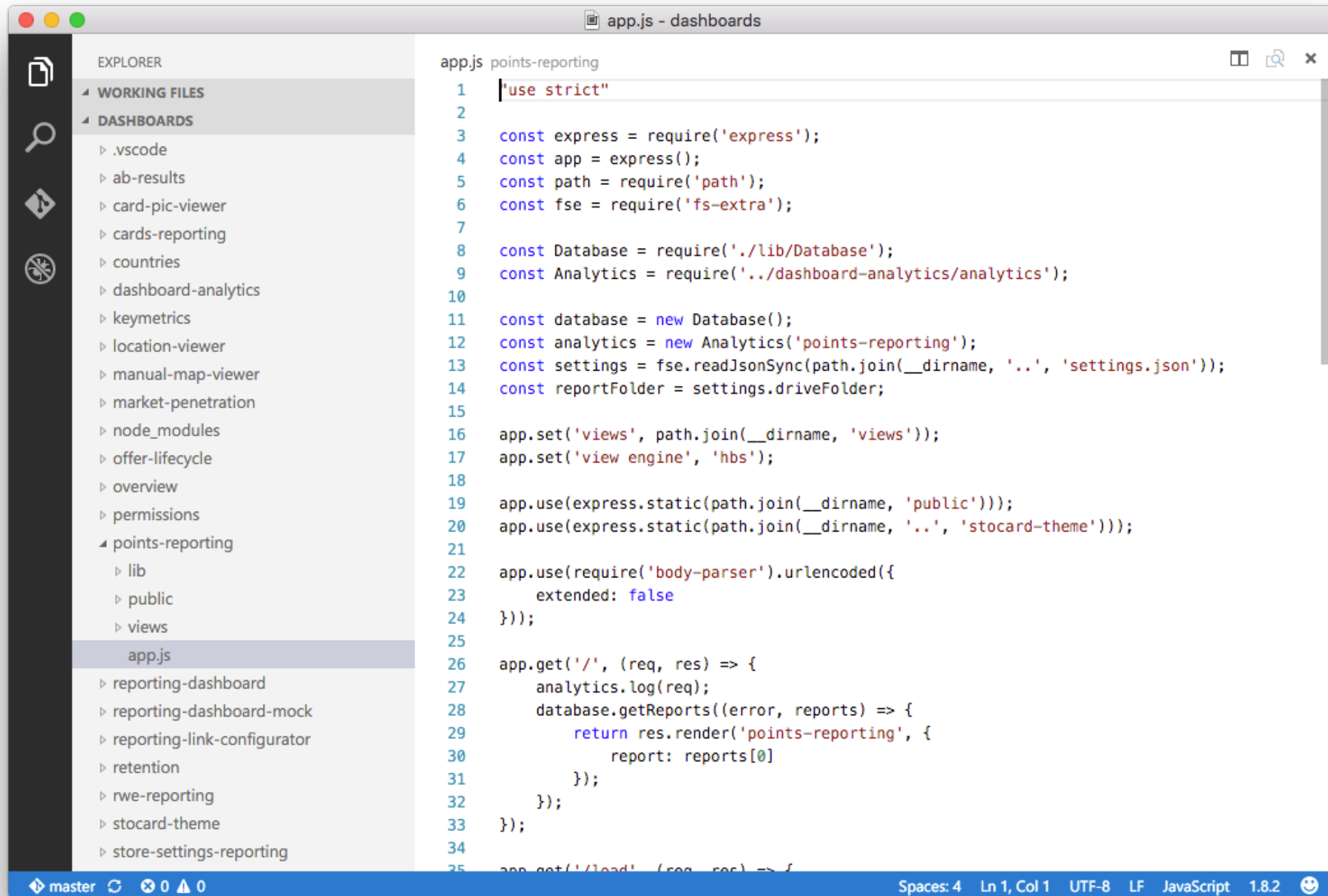
Tools

Sublime Text



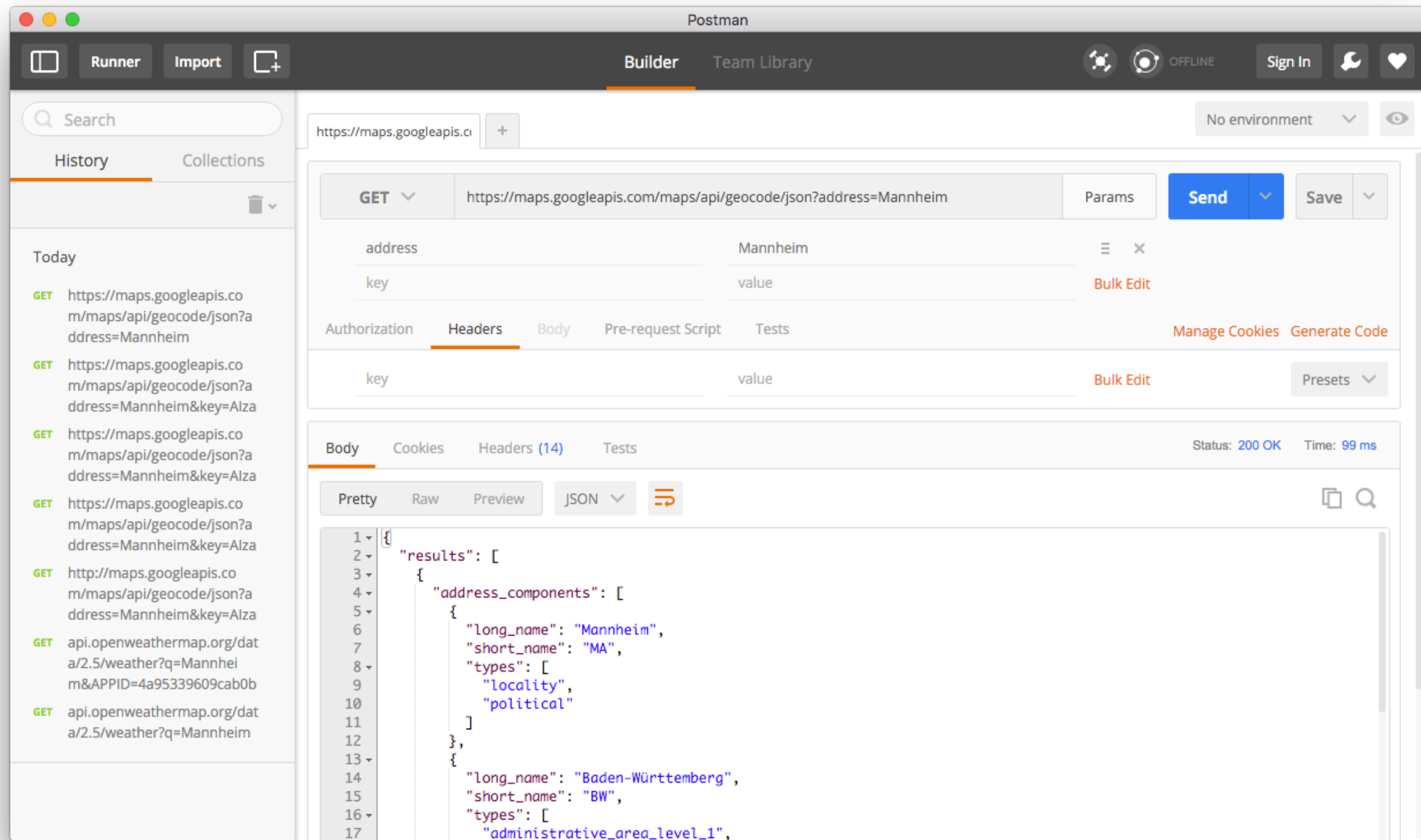
<https://sublimetext.com>

Visual Studio Code



<https://code.visualstudio.com>

Postman



<https://getpostman.com>

Server

12 Factor App

THE TWELVE FACTORS

I. Codebase

One codebase tracked in revision control, many deploys

II. Dependencies

Explicitly declare and isolate dependencies

III. Config

Store config in the environment

IV. Backing Services

Treat backing services as attached resources

V. Build, release, run

Strictly separate build and run stages

VI. Processes

Execute the app as one or more stateless processes

VII. Port binding

Export services via port binding

VIII. Concurrency

Scale out via the process model

IX. Disposability

Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity

Keep development, staging, and production as similar as possible

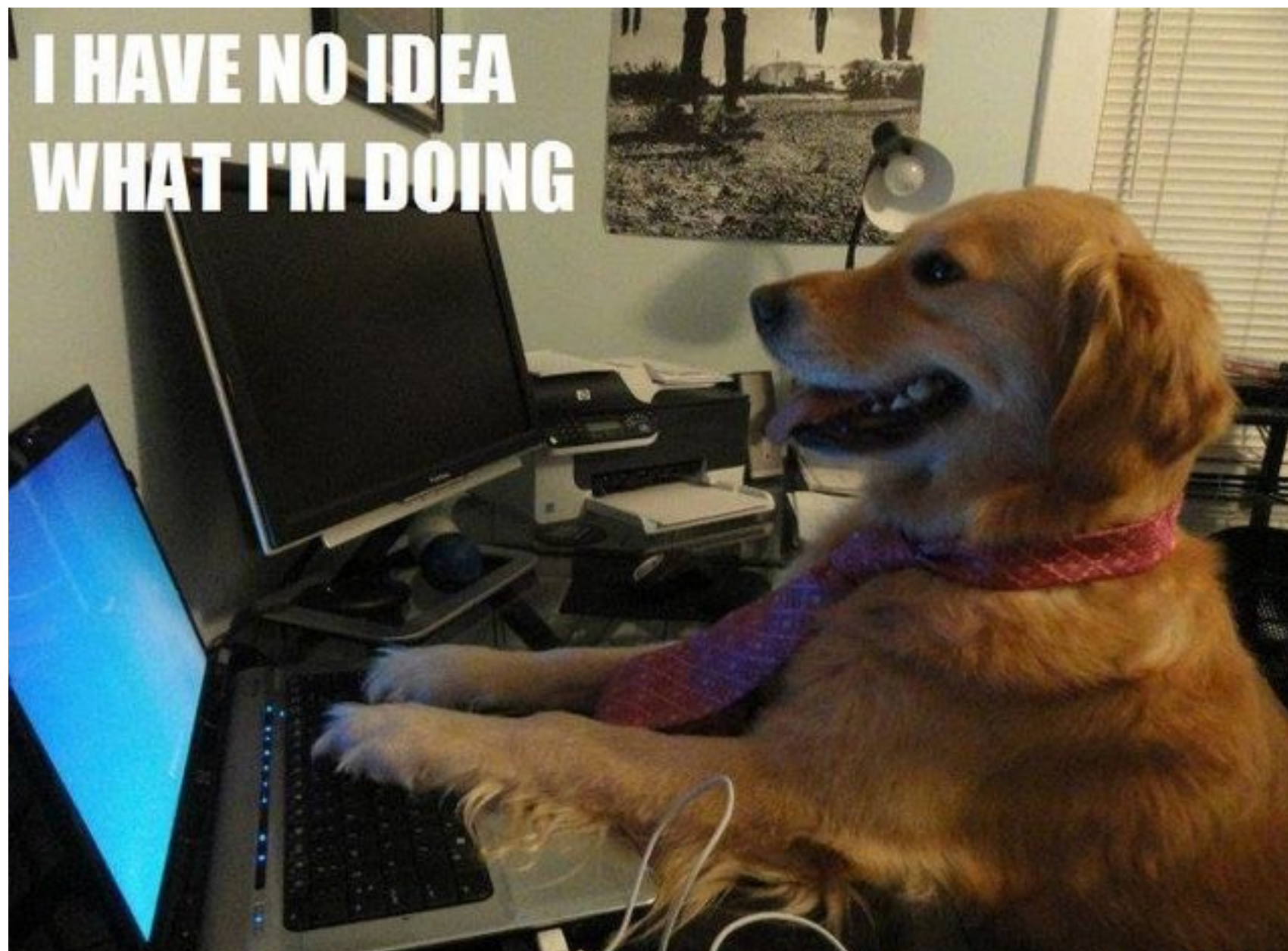
XI. Logs

Treat logs as event streams

XII. Admin processes


Run admin/management tasks as one-off processes

<http://12factor.net/>



**I HAVE NO IDEA
WHAT I'M DOING**

restify



API GUIDE | RESTIFY DOCUMENTATION

Fork me on GitHub

Installation

Server API

Creating a Server

Common handlers: server.use()

Routing

Upgrade Requests

Content Negotiation

Error handling

Socket.IO

Node Server API

Bundled Plugins

Request API

Response API

DTrace

Client API

JsonClient

StringClient

HttpClient

About restify

restify is a node.js module built specifically to enable you to build correct REST web services. It intentionally borrows heavily from **express** as that is more or less the de facto API for writing web applications on top of node.js.

Why use restify and not express?

I get asked this more than anything else, so I'll just get it out of the way up front.

Express' use case is targeted at browser applications and contains a lot of functionality, such as templating and rendering, to support that. Restify does not.

Restify exists to let you build "strict" API services that are maintainable and observable. Restify comes with automatic **DTrace** support for all your handlers, if you're running on a platform that supports DTrace.

In short, I wrote restify as I needed a framework that gave me absolute control over interactions with HTTP and full observability into the latency and characteristics of my applications. If you don't need that, or don't care about those aspect(s), then it's probably not for you.

For real time chat, discussion and support, join the `#restify` IRC channel on `irc.freenode.net`.

About this guide

This guide provides comprehensive documentation on writing a REST api (server) with restify, writing clients that easily consume REST APIs, and on the DTrace integration present in restify.

<http://restify.com/>

Thanks!



Lukas Stehr

stehr@stocard.de

Florian Barth

barth@stocard.de