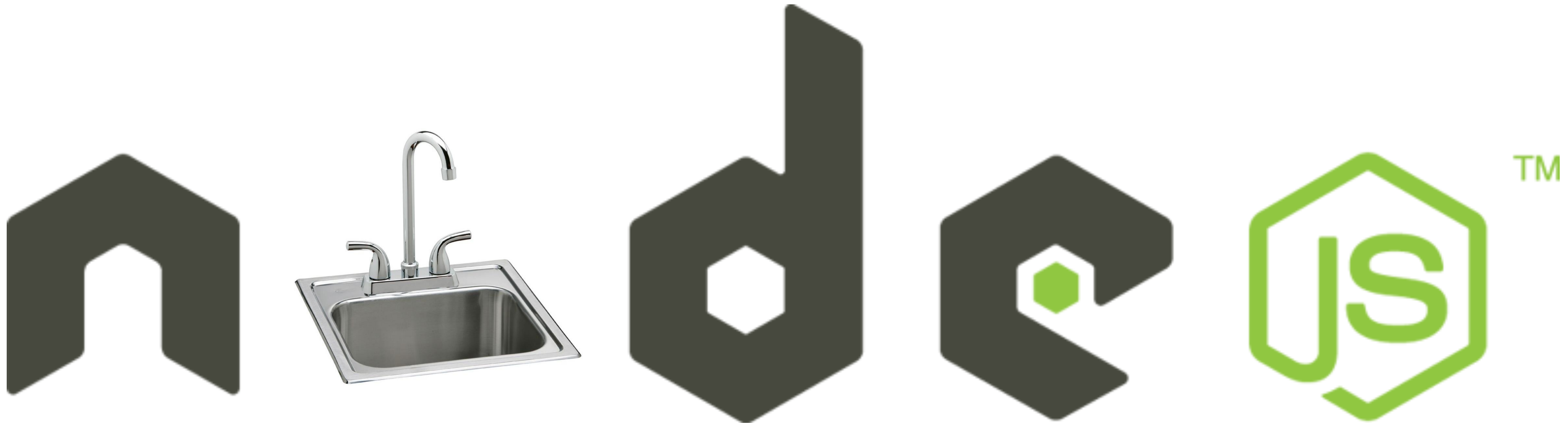
Getting Asynchronous in NodeJS

ALEX SCHWARTZ | MADISON WI

About Me

- Full-Stack Developer/Consultant with Robert Half Technology
- QA Automation Engineer for Sony
- 12+ Years experience
- Geek/Nerd
- First Time Speaking at Conference
- Only person who thinks I’m funny


n
d
e
JS
™

Questions?

JS
TM

Questions?

Like | Comment | Subscribe

What This Talk is About

- JavaScript Basics
- NodeJS
- Asynchronous Programming Concept
- Callback Methods
- Promises
- Advantages/Disadvantages
- Common Pitfalls
- Adding Async To Your Code
- Future State

What This Talk is Not About

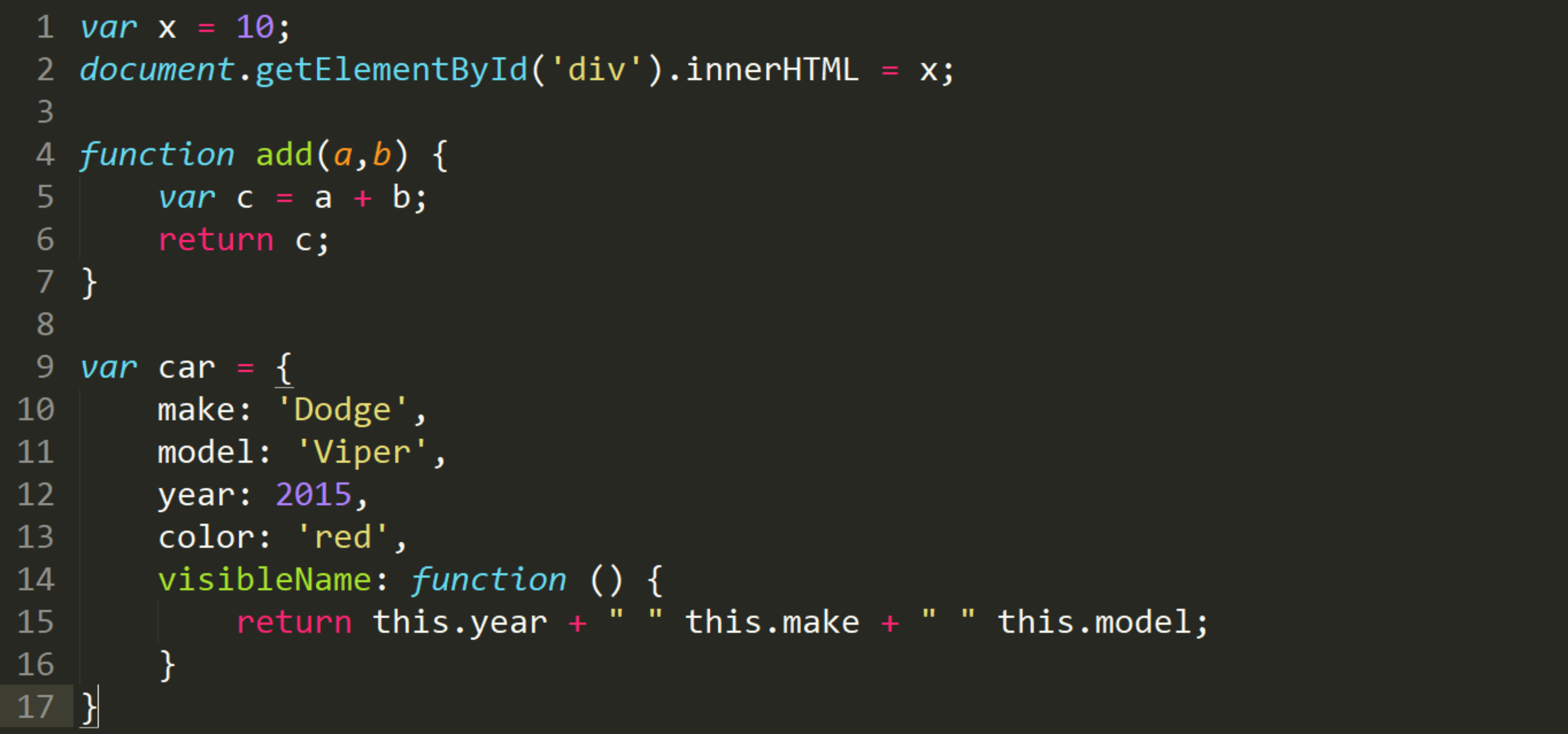
- Installation/Execution of Node
- Building Node Projects
- Working with NPM
- Best Coding Practices for JavaScript
- Every Way to Do Async

Setting the Stage with JavaScript

- Introduced 1995 – Currently on ES8
- Initially developed for client-side practices
    - Event-Driven
    - Standard API
    - I/O Not Part of Standard Package
- Influenced numerous other languages/scripts
    - ActionScript
    - CoffeeScript
    - Jscript
    - .NET

JavaScript Basics

- Event-Driven Loop
  - Odd Scoping/Context
  - Seamless Connection between Server/Client
  - Difficult to Follow | Difficult to Debug
- Framework-Driven
  - Package Hell
  - https://hackernoon.com/how-it-feels-to-learn-javascript-in-2016-d3a717dd577f
  - Newest = Better



```javascript
var x = 10;
document.getElementById('div').innerHTML = x;

function add(a,b) {
    var c = a + b;
    return c;
}

var car = {
    make: 'Dodge',
    model: 'Viper',
    year: 2015,
    color: 'red',
    visibleName: function () {
        return this.year + " " this.make + " " this.model;
    }
}
```

Why Is JavaScript Popular?

- Client/Sever Connection
- Fast
- Community-Driven Packages and Modules
- Cross-Browser Support
- Responsive Design
- NodeJS | Server-Side
- OOP Prototyping
- Optional JIT

Dark Side of JavaScript

- Event-Driven Loop
- Framework Hell (Third-Party Issues)
- Changing ES versions
- Versioning
- Security
- Loose-Typing (Typescript Fixes This)

NodeJS

- Server-Side JavaScript
- Event-Driven, Non-Blocking IO Model
- Fast, No Buffer
- Highly Scalable
- Utilizes ES syntax and Standard JavaScript
- Development and Automated Testing

When To Use NodeJS

- Best Utilized In:
  - I/O Intensive Applications
  - Data Intensive Applications
  - API Intensive Applications
  - High-Speed Web Applications
- Pairs well with numerous front-end frameworks
  - Ember
  - React
  - Angular
  - Backbone

Utilizing NodeJS

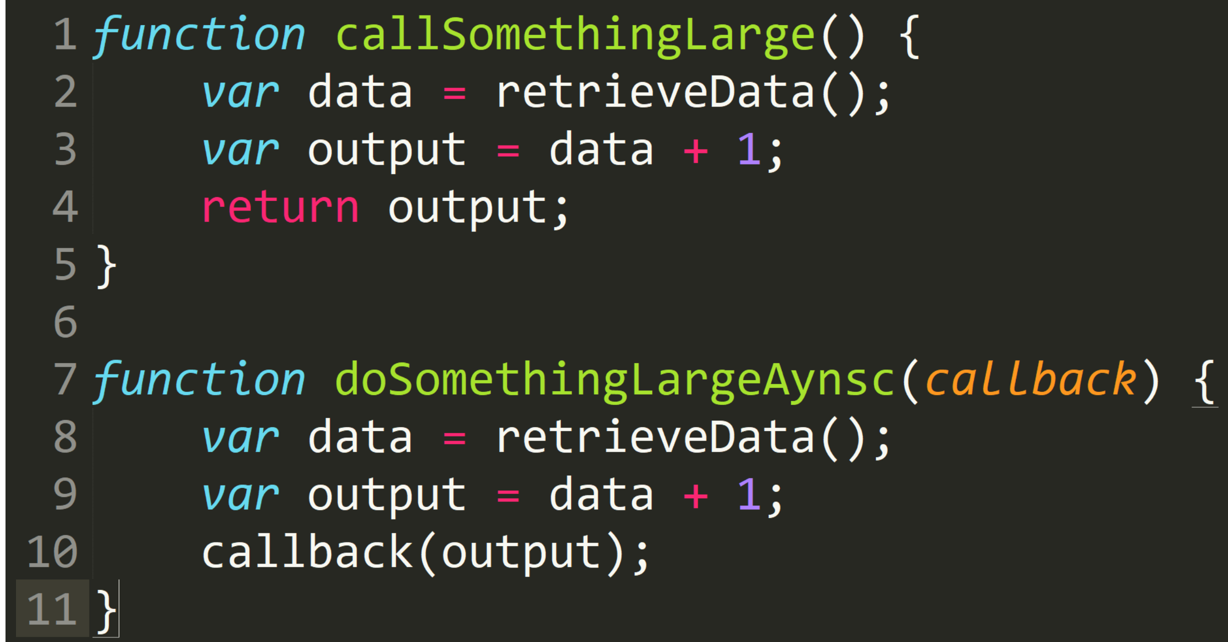
- Express WebServer
- Meteor (Full Stack Design)
- Sails
- KOA
- Useful Frameworks:
  - Babel – Transpile
  - Lodash – File Handling

What is Asynchronous Programming?

- Parallel Execution of Code
    - Unit of Work Independent of main thread
    - Multi-Threaded Execution
- Responsive User Response with Background Activity
    - API Calls
    - Database Writing
    - File Reading/Writing
- Non-Blocking Event-Driven Programming

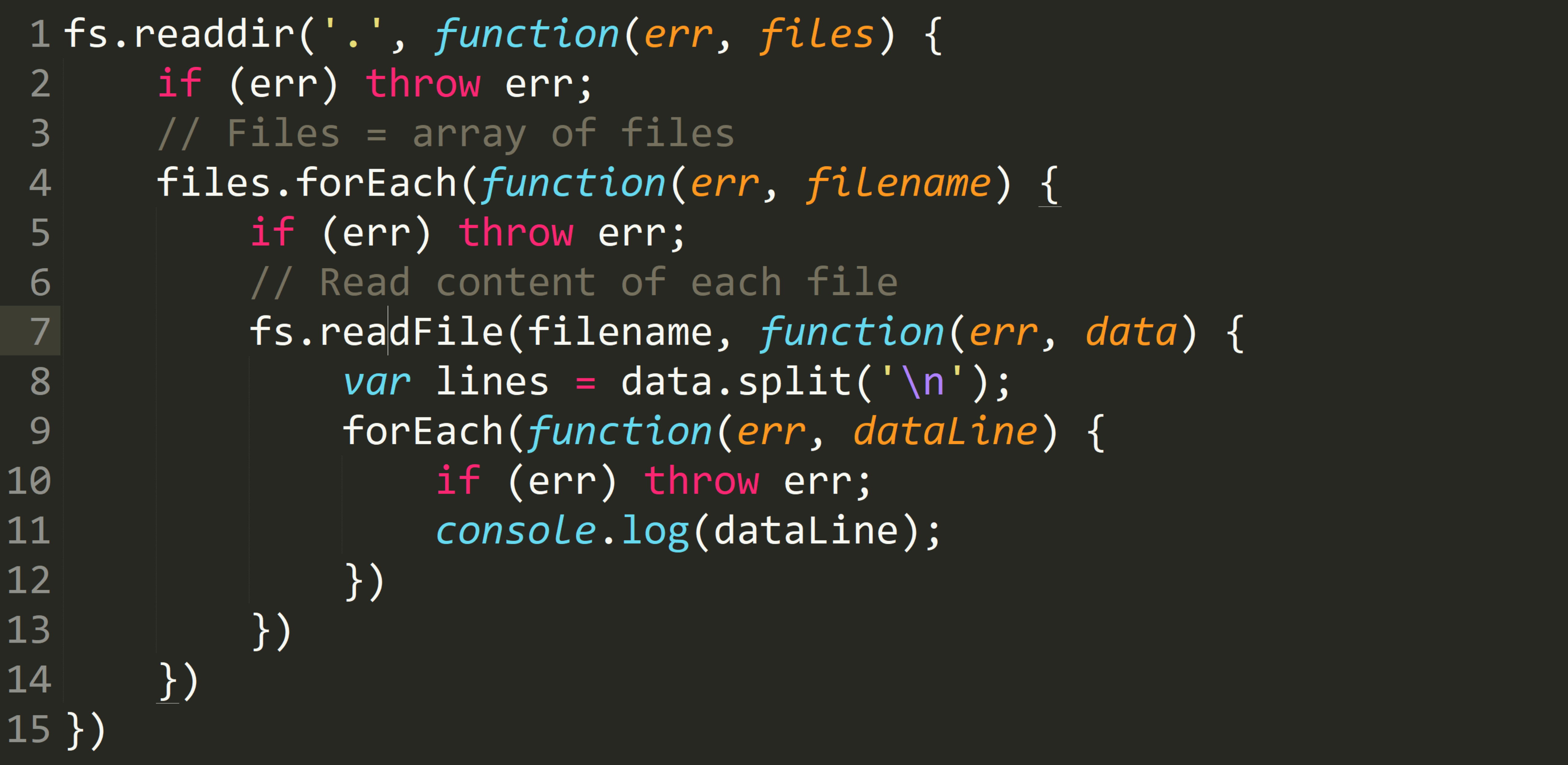
History of NodeJS Async – Feedback Loop

- Callback called when “done”
- Synchronous Inside Function
- Callbacks tell Event Loop when it is “Ready” and “Done”
- Event Handlers Perform Same Function



```
function callSomethingLarge() {
    var data = retrieveData();
    var output = data + 1;
    return output;
}

function doSomethingLargeAynsc(callback) {
    var data = retrieveData();
    var output = data + 1;
    callback(output);
}
```



```javascript
fs.readdir('.', function(err, files) {
    if (err) throw err;
    // Files = array of files
    files.forEach(function(err, filename) {
        if (err) throw err;
        // Read content of each file
        fs.readFile(filename, function(err, data) {
            var lines = data.split('\n');
            forEach(function(err, dataLine) {
                if (err) throw err;
                console.log(dataLine);
            })
        })
    })
})
```

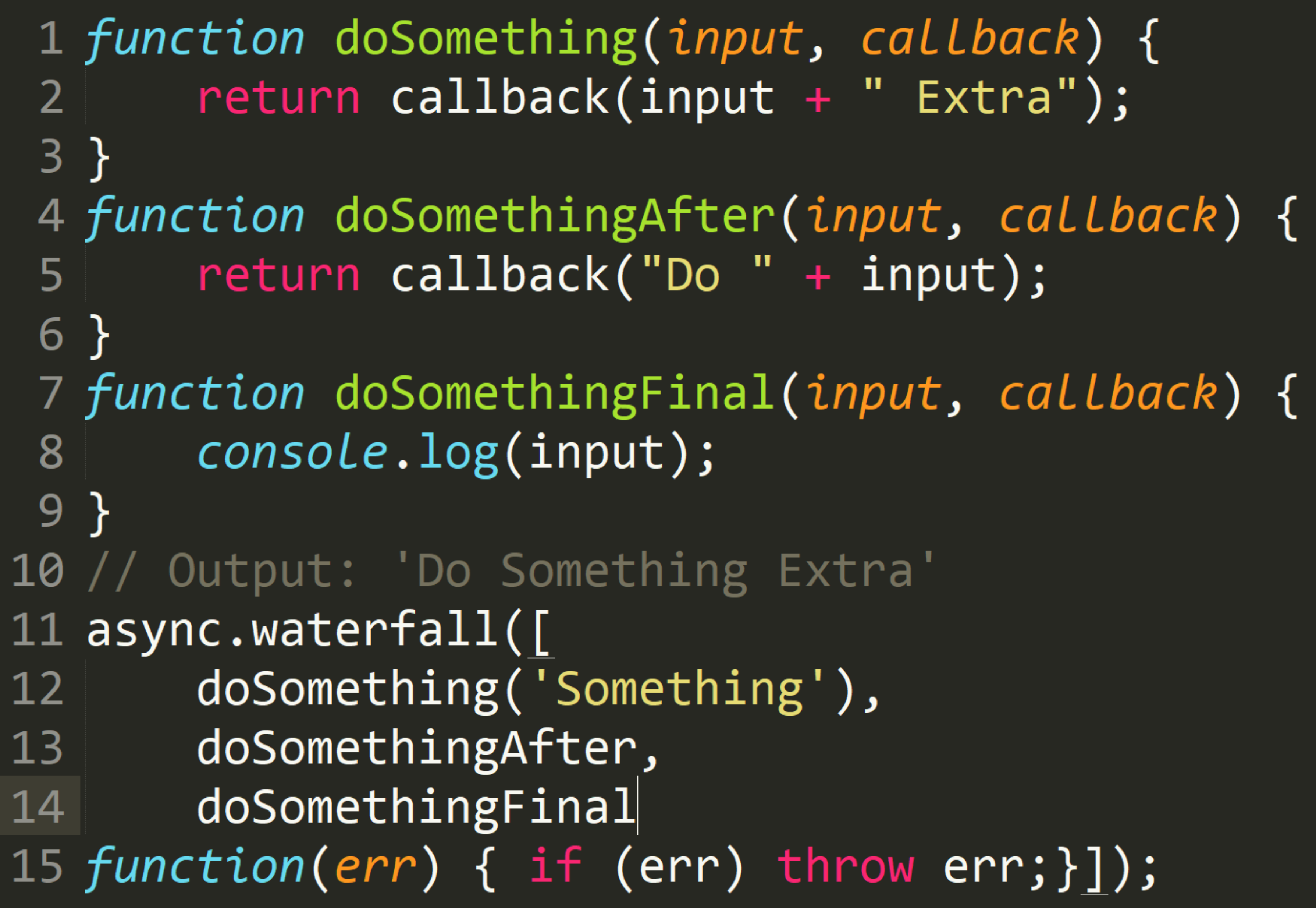
```
doSomething(a) {
    doSomethingElse(a, function(b) {
        doSomethingElse(b, function(c) {
            stillSomethingElse(c, function(d) {
                willThisEverEnd(d, function(e) {
                    nopeStillGoing(e, function(f) {
                        var result = "Where Will This End Up?";
                        return result;
                    });
                });
            });
        });
    })
}
```

Async Module – The First Step

- Framework designed to cleanup the codebase
- Handles Internal Promises
- Allows Parallel Execution of Asynchronous Modules
- Cleans up the Callback Hell

```javascript
function theSmiths(name, callback) {
    return callback(name + ' Smith');
}

// Ouput: [John Smith, Andrew Smith, Robert Smith]
async.map(['John', 'Andrew', 'Robert'], theSmiths, function(err, results) {
    console.log(results);
})
```



```javascript
function doSomething(input, callback) {
    return callback(input + " Extra");
}
function doSomethingAfter(input, callback) {
    return callback("Do " + input);
}
function doSomethingFinal(input, callback) {
    console.log(input);
}
// Output: 'Do Something Extra'
async.waterfall([
    doSomething('Something'),
    doSomethingAfter,
    doSomethingFinal
function(err) { if (err) throw err;}]);
```

Problems

- Current method requires modularization
- Upfront costs are high
- Doesn’t only “Wait” when necessary
- Need “Pending” State

Promises – A Better Way

- Method to “wait” for a value to return
  - Failure Handling
- Four states
  - Fulfilled | Rejected
  - Pending
  - Settled
- Allows code execution to continue passing along a Promise
- Wait only when necessary

```javascript
var promise = new Promise(function(resolve, reject) {
    var x = 1;
    if (x === 1) {
        resolve('It Worked');
    }
    else {
        reject('Math is Broken');
    }
});
```

Promise Chaining

- Utilize new ‘then()’ operator to serialize code
  - Chain them together passing promises along the way
- Standard execution occurs in parallel/async as needed
- ‘catch()’ grabs and passes any errors up the stack
- Can now asynchronously chain our methods together in a simpler format

```javascript
doSomethingAsync()      // Returns promise
    .then(function(result) {
        const testItem = result.item;
        return callAPI(testItem);   // Returns promise
    }).then(function(content) {
        console.log(content);
    }).catch(function(err) {    // Catchs any rejections
        if (err) throw err;
    });
doSomethingElseAsync();
```

Advantages

- Performance Boost when Used Correctly
- Strong UI/UX Performance – eCommerce/Stores
- Data-Intensive Applications
- Promises -> Easy Determination of Async/Sync
- Error Handling
- Modular Code Cleanly

Disadvantages

- Race Conditions
- Non-Standard Approach to Development
  - Code executed in “blocks” or “modules”
- Even Senior Devs Have Issues
- Unique Approach from Non-Programmers
- Harder to Debug | Non-Obvious Solutions

```
requestUser(name)
    .then((result) => {
        if (result) {
            userProfile = result.profile;
        }
    }).catch((err) => {
        throw err;
    });
retrieveElements()
    .then((elements) => {
        generateElements(elements);
    });
processRequest(request)
    .then((response) => {
        if (response.status === 'success') {
            logEvent(response.event);
        }
    });
```

```javascript
MongoClient.connect(url)
    .then((db) => {
        const collection = db.collection('items');
        return collection.findOne({user: '12345'})
        .then((results) => {
            if (results.value !== null) {
                response.output = JSON.stringify(results.value);
                response.render('users', response);
            }
            else {
                return createQuickAccount();
            }
        }).catch((error) => {
            response.error = 'MongoDB Error';
            response.render('error', response);

    })
```

Common Pitfalls

- Database Connections
- Variable Assignment
- Returning Promises vs Handling Them
- Form over Function

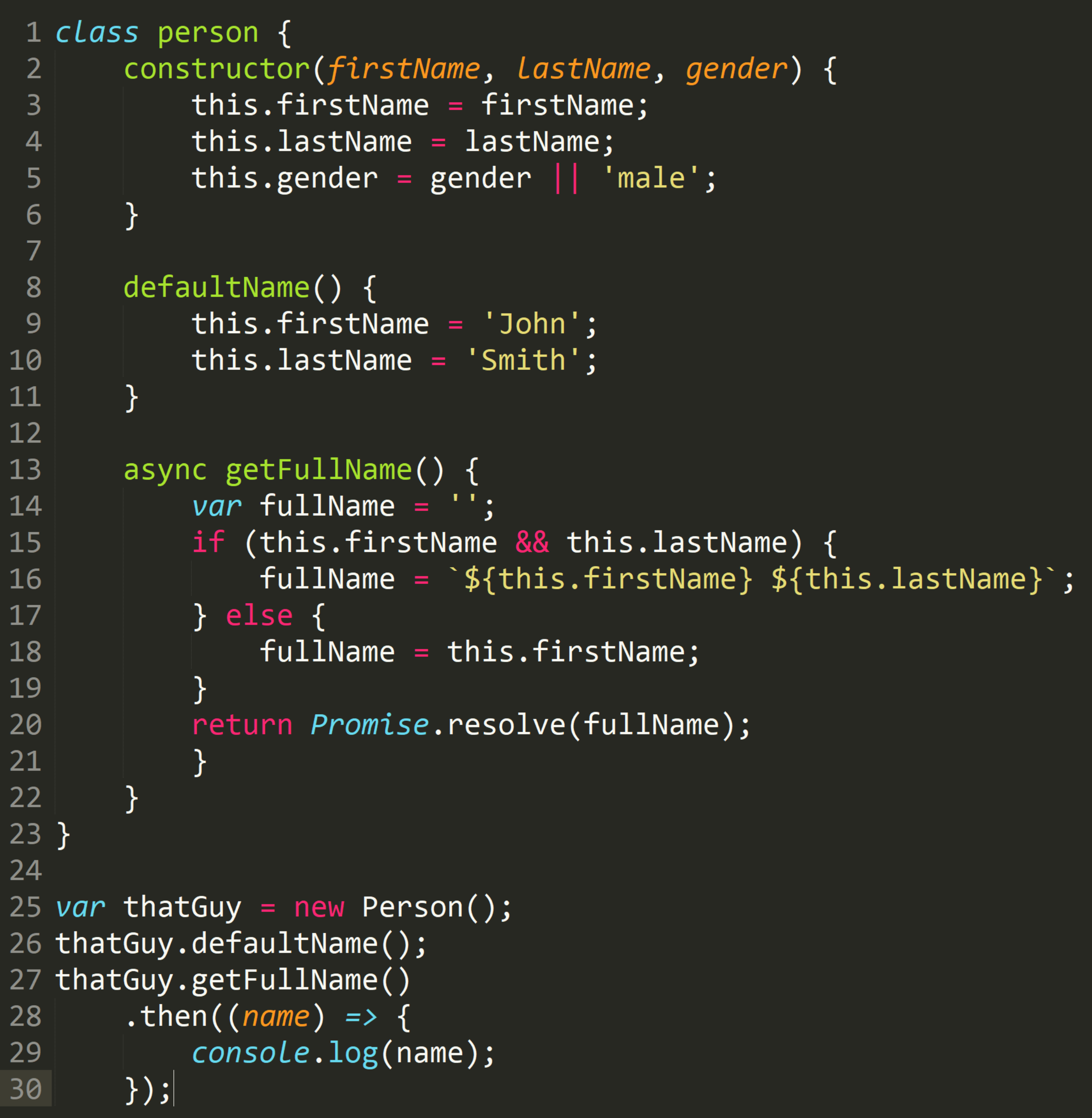
```javascript
MongoClient.connect(url)
    .then((db) => {
        const collection = db.collection('something');
        collection.findAndModify(
            {username: 'alpha'},
            [['_id' 'asc']],
            {$set: {password: '1245'}}
        ).then((results) => {
            if (results.value !== null) {
                users.createUser(results.value)
                    .then((userProfile) => {
                        response.user = userProfile;
                        response.render('users', response);
                    });
            } else {
                ACU.createAccount(username, password)
                    .then((result) => {
                        collection.insertDocument(result)
                            .then(() => {
                                response.user = {result.username, result.password};
                                response.render('users', response);
                            });
                    });
            }
            db.close();
        })
    })
```

Older Methods

- ES6 brought native support for Promises
- Older modules
  - Q
  - RSVP
  - WinSync
- Promises supported with Transpiling

Adding Async to Your Code

- Used in Class Object Design
- Diagram/Outline Code and Design First
- Use Only When Necessary
- Keep Code Modular For Ease of Reading
- General Coding Standards Really Help



```javascript
class person {
    constructor(firstName, lastName, gender) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.gender = gender || 'male';
    }

    defaultName() {
        this.firstName = 'John';
        this.lastName = 'Smith';
    }

    async getFullName() {
        var fullName = '';
        if (this.firstName && this.lastName) {
            fullName = `${this.firstName} ${this.lastName}`;
        } else {
            fullName = this.firstName;
        }
        return Promise.resolve(fullName);
        }
    }
}

var thatGuy = new Person();
thatGuy.defaultName();
thatGuy.getFullName()
    .then((name) => {
        console.log(name);
    });
```

Current State and Future

- Numerous APIs and Frameworks Deliver Promises Now
- Async/Await Syntax Cleans up JavaScript Considerably
  - Simplifies execution path
  - Easy Storage of Promises
- More Frameworks Embracing Async Code
- Seeing Async in other Languages

Summary

- Basics of NodeJS/JavaScript
- Early Concepts of Asynchronous Coding
- Callbacks and Modular Code
- Promises
- Promise Chaining and Combining with Async Modules
- Advantages/Disadvantages
- Common Pitfalls
- Future State

Questions?

Presenter: Alex Schwartz | Madison, WI

Twitter: @TheRedGamer

LinkedIn: https://www.linkedin.com/in/alex-schwartz-83164b3a (Search Alex Schwartz Madison, WI)

Email: alex.schwartz1550@gmail.com

Questions Beyond This Presentation? Please Find me After!

Topics To Talk To Me About

- Video Games (Overwatch, LoZ: Breath of the Wild)
- Competitive Chess
- Aerospace Engineering
- Madison, WI
- Comic Book Characters
- Why I Suck at Fighting Games
- Why I Suck at FPS Games
- Anything At All Really