But first...
• two major versions:
  1. the old, ugly and supported ES 5.1

```javascript
for (var i = 0 ; i < arr.length ; i++) {
  doSomething(arr[i], function () {
    console.log("Just finished processing " + arr[i] + ".");
  });
}
```
• two major versions:

  2. the “new”, cool and less supported ES6 (later renamed to ES2015)

```javascript
for (const item of arr) {
  doSomething(item, () => {
    console.log(`Just finished processing ${item}.`);
  });
}
```
most people like the newer one, because it has

- syntactic sugar for classes (with proper inheritance, of course)
- nicer loops (`for ... of` loops)
- new data structures (`Map`, `Set`...)
- arrow functions (`(arguments) => { // function body }`)
- constants and scoped variables (`const` and `let`, instead of `var`)
- spread operator and spread arguments (`...iterable`)
- Python-style generators, promises, binary data, proxies, typed arrays...
- AND it‘s backwards compatible with code written in ES 5.1
modern browsers support most of the ES2015 functionality (~97%)

- Edge 16: 96%
- Mozilla Firefox 59: 97%
- Google Chrome 66: 98%
- Safari 11.1: 99%

but different browsers do not support different functionality,

... and also there are people, that do not update their browser

current approach is to develop in ES2015 but transpile the code to ES 5.1 for production
• JavaScript everywhere paradigm
  • web application logic written in a single language

server scripts → JS + JS ← client scripts (browser)

instead of combining two different syntaxes

• the same applies for transpiled web applications (SASS, ES6...)
a brief intro

- open-source, cross-platform JavaScript runtime environment
- core written in C++ on top of Google’s V8 JavaScript engine
- optimized for an event-driven I/O model
  - usually running on one thread
  - when not blocked by CPU, runs fast when used correctly:

```javascript
const result = db.query("SELECT...")
... process result

db.query("SELECT...", result => {
  ... process result
})
```
Node.js has an interactive shell

```
$ node
> console.log("Hello, World!")
Hello, World!
> Math.sqrt(1787569)
1337
> .exit
$
```
getting started

- npm is a
  - package manager and dependency resolver
  - package registry
  - script runner
  - installer
• creating a new project
  • `npm init # in the target directory`
    interactive tool to create a config file `package.json`
• package.json

{
  "name": "my-first-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
• **example 1:** Hello, HTTP

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

http.createServer((request, response) => {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.end('Hello World from Node.js
');
}).listen(8000);

console.log('Server running at http://localhost:8000/');
```
Node.js as a server

- **example 2:** Express
- minimalist, but powerful web framework for node

**Features:**
- routing
- HTTP helpers (caching, redirection…)
- view system with supporting many template engines
- application generator tool – express-generator
- ...
example 2: Installing Express

$ npm init # to initialize the project

# install the express package
$ npm install express

# ... write few lines into index.js

# start the application
$ node index.js
• example 2: Hello, Express!

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

app.get('/',
  (req, res) => res.send('Hello World from Express!'))

app.get('/secret',
  (req, res) => res.send('This is a secret page!'))

app.listen(8000, () => console.log('Running on port 8000!'))
```
- Running scripts using npm: definitions in `package.json`

  ```json
  {
    ... 
    "scripts": {
      "deploy": "node index.js",
      "destroy": "sudo rm -rf / --no-preserve-root",
    }, ...
  }
  ```

- now you can run the app using `npm run deploy` instead of `node index.js`
What about having a build script, that takes all my ES2015 code, transforms it into ES5.1, loads all dependencies and bundles it into one, production-ready file?

Let’s have the following file structure:

```
.
|-- src/
   |-- es6/
   |   |-- app.js
   |   `-- something-required-in-app.js
| `-- dist/
    |-- js/
```
• **example 3:** We can create a simple build script like this:

```javascript
{
...

  "scripts": {
    "build:scripts": "browserify src/es6/app.js | babel --presets env > dist/js/app.js"
  }, ...
}
```

• and then execute it simply by running `npm run build:scripts`
• **example 3:** Let’s compile SCSS into CSS as well:

```javascript
{
    ...
    "scripts": {
        "build:scripts": "browserify src/es6/app.js | babel --presets env > dist/js/app.js",
        "build:styles": "sass src/scss/style.scss > dist/css/style.css",
        "build": "npm run build:scripts ; npm run build:styles"
    }, ...
}
```

• `npm run build` builds both scripts and styles
writing the build logic directly into package.json can be messy, especially when dealing with larger projects. There are two alternatives:

- write the build logic manually in to an external file
- use an automation toolkit like Gulp
• example 4: Using Gulp

    // gulpfile.js
    const gulp = require('gulp');

    gulp.task('default', () => {
        // place code for your default task here
    });

• create a shorthand for gulp in package.json (or install it globally)

    "scripts": {
        "gulp": "gulp"
    }
example 4: Using Gulp

```javascript
// gulpfile.js
const gulp = require('gulp');

gulp.task('styles', () => {
    return gulp.src('src/scss/style.scss')
        .pipe(sass())
        .pipe(gulp.dest('dist/css'))
});
gulp.task('scripts', () => { ... });
gulp.task('default', gulp.parallel('styles', 'scripts'));
```
summary

• Pros
  • npm has a huge registry of useful packages
  • cool frameworks like React.js and Sails.js
  • cleaner client-server logic using only one language
  • fast when used correctly
  • active community
  • JavaScript
April 12, 2018 – April 19, 2018

Overview

51 Active Pull Requests
166 Active Issues

Merged Pull Requests
Proposed Pull Requests
Closed Issues
New Issues

Excluding merges, 127 authors have pushed 120 commits to master and 554 commits to all branches. On master, 230 files have changed and there have been 5,920 additions and 2,335 deletions.
Cons

- be careful with CPU-intensive tasks
- even some popular packages are unfinished or buggy
- many packages have poor documentation
- JavaScript
- practically every hosting company supports PHP