Tooling

le wagon

A History of JavaScript

JavaScript

THE **CLIENT-SIDE** PROGRAMMING LANGUAGE

- Created in **10 days** in 1995 by **Brandon Eich**
- Came as a **built-in** language in **Netscape** in 1996
- Standardised by the **ECMA International** from 1998 on

JavaScript

ASYNCHRONICITY

- In **2005**, release of a white paper describing **AJAX**
- Set of **technologies** built over **JS**
- **Web apps** where data can be **loaded in the background**
- **No** more need for **full page reloads**

JavaScript

ECMAScript

- In **2009**, ECMAScript 3.1 renamed ECMAScript 5 (**ES5**)
- **Harmony** project: new cycle of **evolution / innovation**
- **Nodejs:** JS as **server-side** language
- **2015**: release of ECMAScript 2015 (**ES6**)

JavaScript

Frameworks

JavaScript

Tools

BABEL

npm 😱 yarn

JavaScript

How to run some JavaScript?

In your terminal

Run a **file** or open a **console**

```
node hello_world.js
```

```
node
>
```

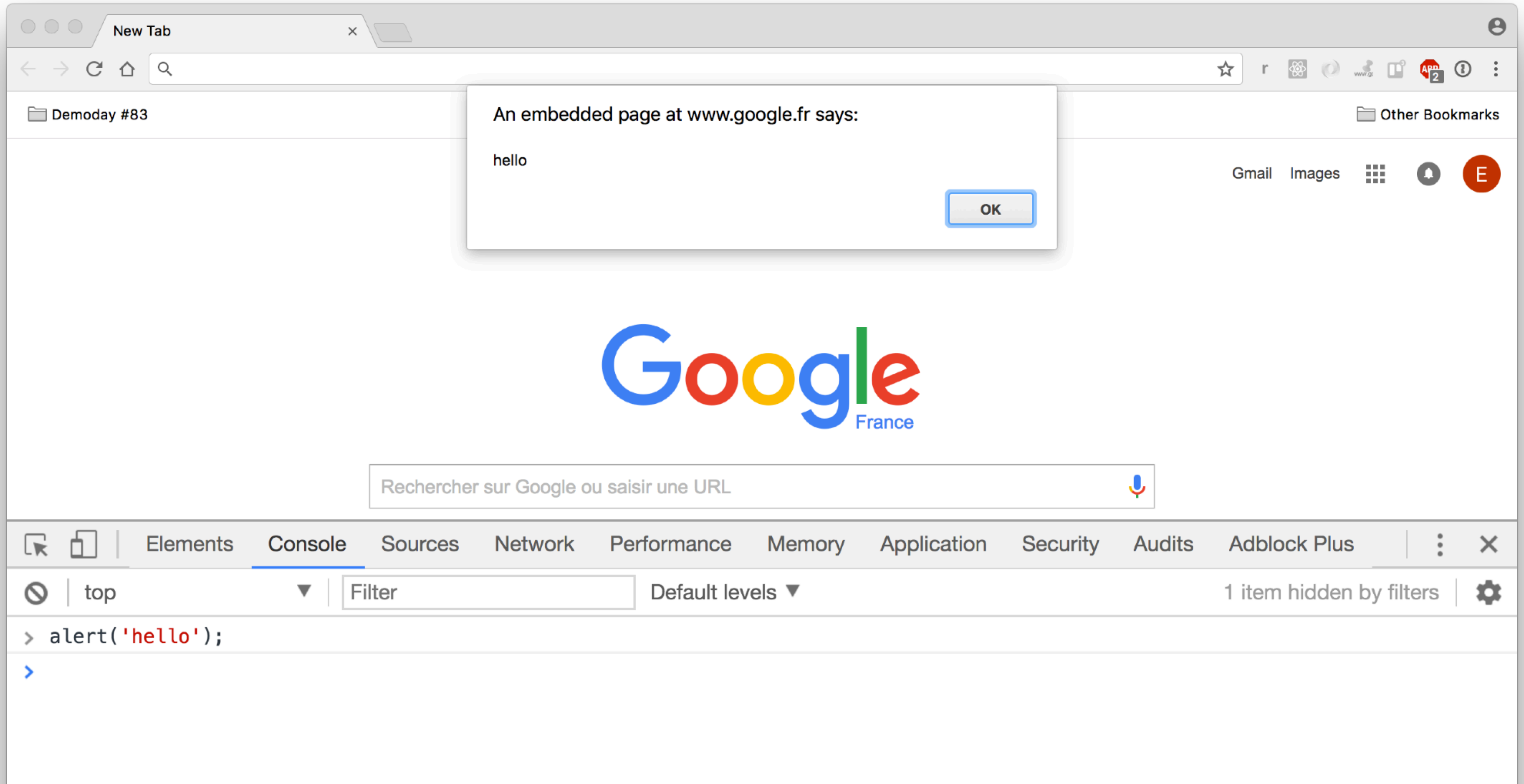
Setup

```
# OSX
brew update
brew install node
```

```
# Ubuntu
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
sudo apt-get install -y nodejs
```

In your browser

Modern **browsers** have a **console**



New Tab

Demoday #83

Other Bookmarks

An embedded page at www.google.fr says:

hello

OK

```
> alert('hello');
>
```

ES6 Refresher

Variables - let

Works like the good old **var**...

```
let name = 'Boris';
console.log(name); // Boris
name = 'Romain';
console.log(name); // Romain
```

...but **block-scoped** instead of **function-scoped**

```
if (name === 'Boris') {
  let sentence = `Hello ${name}`;
  console.log(sentence); // hello Boris
}

console.log(sentence);    // ReferenceError
```

Variables - const

Works like **let** but **cannot be reassigned** a value...

```
const name = 'Boris';
name = 'Romain'; // TypeError: Assignment to constant variable.
```

⚠️ the object stored is **not immutable!**

```
const user = {
  name: 'Boris',
  role: 'CEO'
};
user.name = 'Boris Paillard'; // allowed 👌
```

Template literals

Think of **interpolation**...

```
const age = 18;
console.log(`I'm ${age} years old`);
// I'm 18 years old
```

...they support **multi-lines** and you can **nest** them!

```
const people = [
  { name: 'Alice', age: 24 },
  { name: 'Bob', age: 32 },
  { name: 'Charles', age: 45 }
];

const markup = `
  <ul class="people">
    ${people.map(person => {
      `<li>${person.name} is ${person.age} years old</li>`
    }).join('')}
  </ul>
`;
```

Arrow functions

Anonymous **functions** in **ES5**...

```
function() {
  // Some js
}
```

...can be written in **ES6:**

```
() => {
  // Some js
}
```

Arrow functions

Concision

```
const numbers = [ 1, 2, 3 ];
const squares = numbers.map(function(number) {
  return number * number;
});
```

...becomes in **ES6:**

```
const numbers = [ 1, 2, 3 ];
const squares = numbers.map((number) => {
  return number * number;
});
```

Arrow functions

Implicit return

```
const numbers = [ 1, 2, 3 ];
const squares = numbers.map(n => n * n);
```

Arrow functions

This binding

```
const refreshButton = document.querySelector('#refresh');
dropdown.addEventListener('click', function() {
  this.innerHTML = 'Hold still…';
  var that = this;
  setTimeout(function() {
    that.innerHTML = 'Refresh';
  }, 1000);
})
```

…becomes:

```
const refreshButton = document.querySelector('#refresh');
dropdown.addEventListener('click', function() {
  this.innerHTML = 'Hold still…';
  setTimeout(() => { // binds `this` to the function
    this.innerHTML = 'Refresh';
  }, 1000);
})
```

Arrow functions

Even though they are **anonymous**...

```
() => {
  console.log('Hello');
}
```

...you can **store** them in a **variable**

```
const greet = () => {
  console.log('Hello');
}
```

...and **call** them

```
greet();
// Hello
```

Array

New methods

```
const cities = [ 'Paris', 'London', 'Berlin' ];

cities.find(city => city.startsWith('P'));      // Paris
cities.findIndex(city => city.startsWith('P')); // 0
cities.some(city => city.startsWith('P'));      // true
cities.every(city => city.startsWith('P'));     // false
```

Classes

Very similar to **Ruby** classes

```
class User {
  constructor(name, email) {
    this.name = name;
    this.email = email;
  }

  greet() {
    return `Hello ${this.name}!`;
  }
}
```

```
const boris = new User('Boris', 'boris@lewagon.org');

console.log(boris.greet());
// Hello Boris!
```

And many more...

- Set, Map
- Parameter **default values** (similar to Ruby)
- **Spread** operator (...)
- **import / export** statements for modules
- All kinds of **destructuring**

THAT WE'LL COVER LATER THIS WEEK

Tooling

Problem

Programmers work in **many** separate **files**

```
<!-- index.html -->
<!DOCTYPE html>
<html>
  <head>
    <!-- [...] -->
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <link rel="stylesheet" href="css/navbar.css">
    <link rel="stylesheet" href="css/banner.css">
    <link rel="stylesheet" href="css/footer.css">
    <link rel="stylesheet" href="css/blog.css">
    <link rel="stylesheet" href="css/product.css">
    <!-- etc. –>
  </head>
  <body>
    <!-- [...] -->
    <script src="//code.jquery.com/jquery-3.2.1.min.js"></script>
    <script src="js/jquery-ui.js"></script>
    <script src="js/isotope.pkgd.js"></script>
    <script src="js/jquery.countTo.js"></script>
    <script src="js/jquery.flexslider.js"></script>
    <script src="js/ads.js"></script>
    <script src="js/scripts.js"></script>
    <script src="js/owl.carousel.js"></script>
    <!-- etc. -->
  </body>
</html>
```

Solution

Bundle, compile & **compress server**-side

👍

Solution

Load only the **bundle** you **need**

```html
<!-- index.html -->
<!DOCTYPE html>
<html>
  <head>
    <!-- [...] -->
  </head>
  <body>
    <!-- Some HTML in here -->
    <script src="dist/bundle.js"></script>
  </body>
</html>
```

Package Repository

"The equivalent of RubyGems for JS libraries"

Package Manager

To download packages (libraries) from **npm** in your project

Setup

```
# OSX
brew install yarn
```

```
# Ubuntu
curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee /etc/
apt/sources.list.d/yarn.list
sudo apt-get update && sudo apt-get install yarn
```

`yarn init`

Starting a front-end project? Run **`yarn init`**

```
cd ~/code/<your_github_nickname>
mkdir yarn_project && cd $_
yarn init
# use UNLICENSED when asked if private code
```

This creates a **`package.json`** file

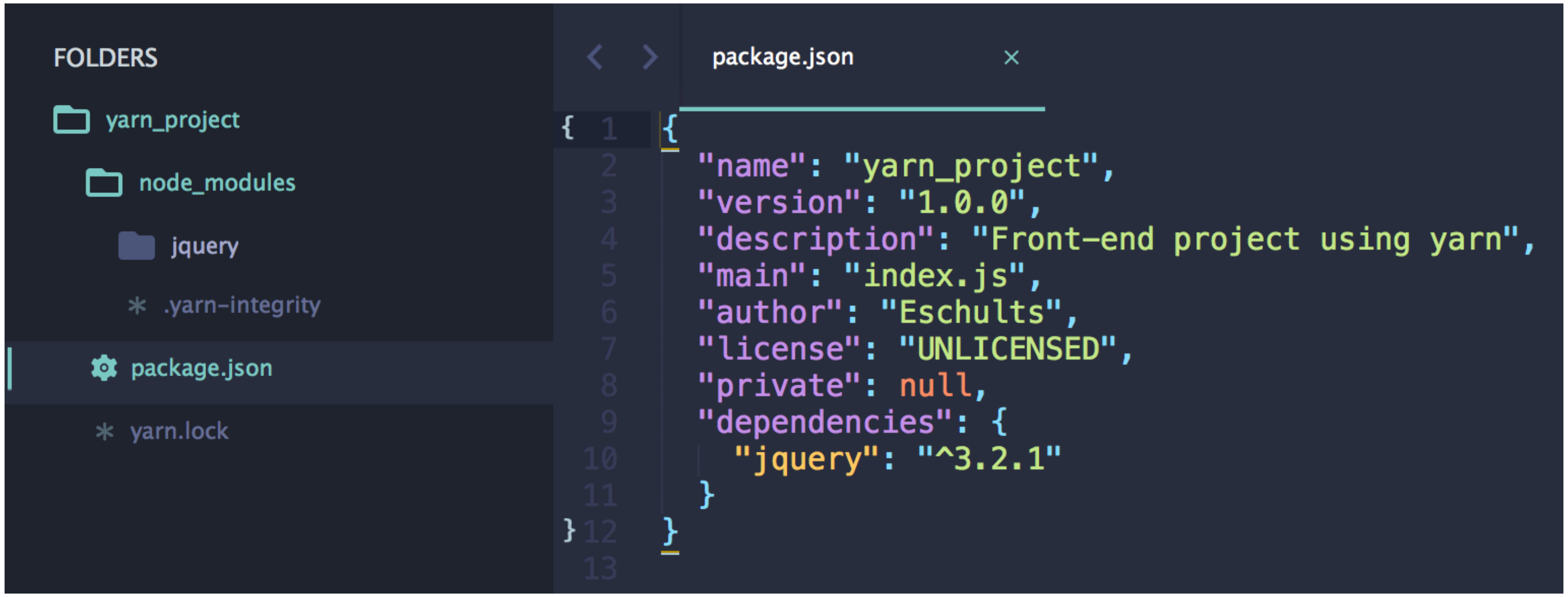
Think of Ruby's **`Gemfile`**

Usage

Add a new package in your project with **`yarn add`**

```
yarn add <package> [--dev]
```

For instance, **`yarn add jquery`** results in:

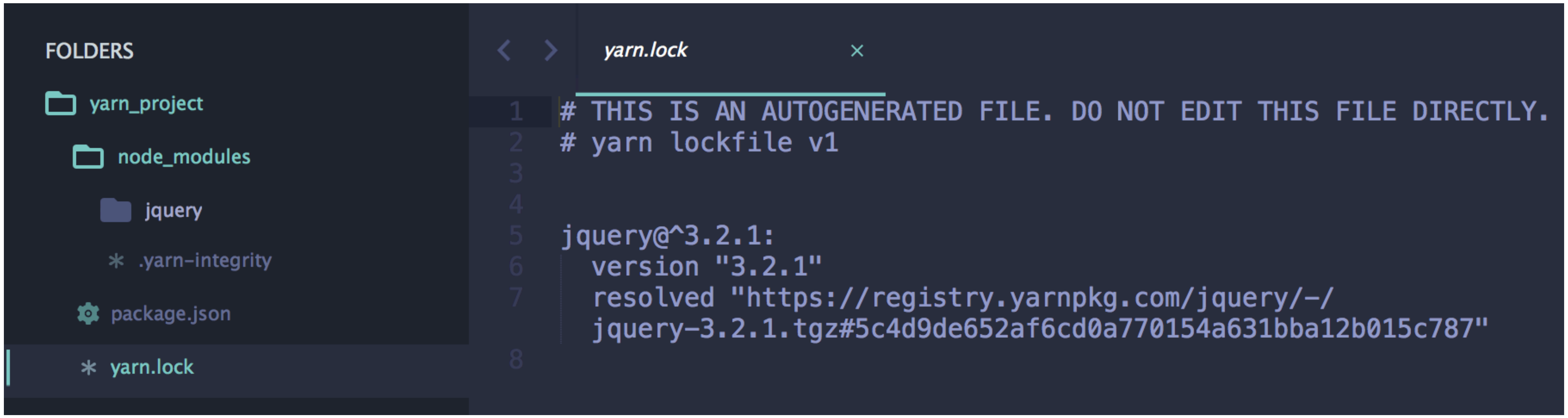


```json
{
  "name": "yarn_project",
  "version": "1.0.0",
  "description": "Front-end project using yarn",
  "main": "index.js",
  "author": "Eschults",
  "license": "UNLICENSED",
  "private": null,
  "dependencies": {
    "jquery": "^3.2.1"
  }
}
```

Generated files

`node_modules` is a folder created by your first **`yarn add`**
it stores all the project’s libraries you downloaded with yarn
⚠️ add it in your `.gitignore`

`yarn.lock` is the file locking the versions of your dependencies
think of Ruby's **`Gemfile.lock`**



```
# THIS IS AN AUTOGENERATED FILE. DO NOT EDIT THIS FILE DIRECTLY.
# yarn lockfile v1


jquery@^3.2.1:
  version "3.2.1"
  resolved "https://registry.yarnpkg.com/jquery/-/jquery-3.2.1.tgz#5c4d9de652af6cd0a770154a631bba12b015c787"
```

Check your style

ESLint is **the** tool to help you write **proper** JS

Add it in **every** project

```
yarn add eslint --dev
eslint --init
#? How would you like to configure ESLint? Use a popular style
guide
#? Which style guide do you want to follow? Airbnb
#? Do you use React? Yes
#? What format do you want your config file to be in? JSON

rm package-lock.json # we just need yarn.lock
```

We recommend the **Airbnb JS style guide**

Update rules

```
# .eslintrc.json
{
  "extends": "airbnb", # this line should already be here.
  "rules": {
    "no-console": "off",
    "comma-dangle": "off",
    "quotes": "off",
    "react/prop-types": 0,
    "arrow-body-style": 0,
    "space-before-function-paren": 0
  },
  "env": {
    "browser": true
  }
}
```

$PATH

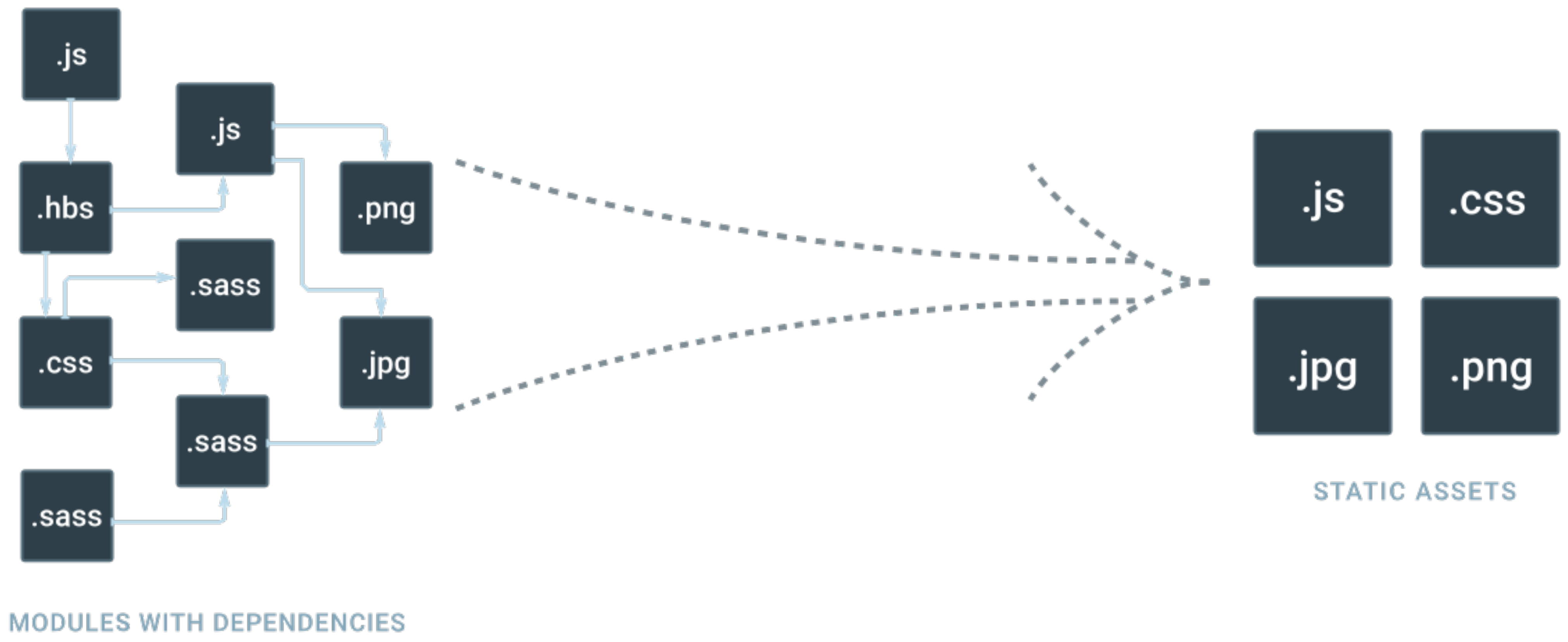
Make sure you have `./node_modules/.bin` in your **$PATH**!

```
# check your current PATH with:
echo $PATH
./bin:./node_modules/.bin:/usr/local/opt/rbenv/shims:/usr/local/
opt/rbenv/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/
local/sbin
```

To change it:

```
# Open zshrc in Sublime Text:
st ~/.zshrc
# If you make a change, restart all your terminals!
```

Webpack



“The JavaScript version of the Rails Asset Pipeline”

“A **module** **bundler** for modern JS apps”

Webpack

Nice **dev-friendly tools** like **HMR** or **sourcemap**

Handles not only JS but **all kinds of assets**

Loaders transform other resources into JS

Built-in feature in **Rails 5.1**

Webpack

Setup

```
yarn add webpack webpack-dev-server --dev
```

```
touch index.html
mkdir src
touch webpack.config.js
touch src/index.js # <– code in here
```

Basic config here

Launch a dev **server** with:

```
webpack-dev-server
```

Open a browser & listen to **http://localhost:8080**

Problem

We saw that programmers work in many separate files.
How do we handle **dependencies** when we develop?

```
// src/greet.js

function greet(firstName) {
  return `Hello ${firstName}`;
}
```

```
// src/index.js

greet('Boris');
// ReferenceError: greet is not defined
```

Here, we need to load **`greet()`** before calling it in **`index.js`**

Solution

Explicitly **import** the desired code.
Needs to be **exported** first.

```
// src/greet.js

function greet(firstName) {
  return `Hello ${firstName}`;
}

export { greet };
```

```
// src/index.js
import { greet } from './greet'; // relative path

greet('Boris');
// Hello Boris
```

export default

You can export a single item as the **default** export

Drop the brackets when importing it

Acts as **fallback** import when the **whole module** is required

```
// src/greet.js

export default function greet(firstName) {
  return `Hello ${firstName}`;
}
```

```
// src/index.js
import greet from './greet';

greet('Boris');
// Hello Boris
```

import (external)

Assuming you already ran **`yarn add jquery`**...

```
// ./src/index.js
import $ from 'jquery'; // name of the module === folder

// you can use $ in here 👌
$(document).ready(function() {
  console.log('jQuery just checked that the DOM is ready!');
});
```

We can now handle **dependencies** thoroughly

Production

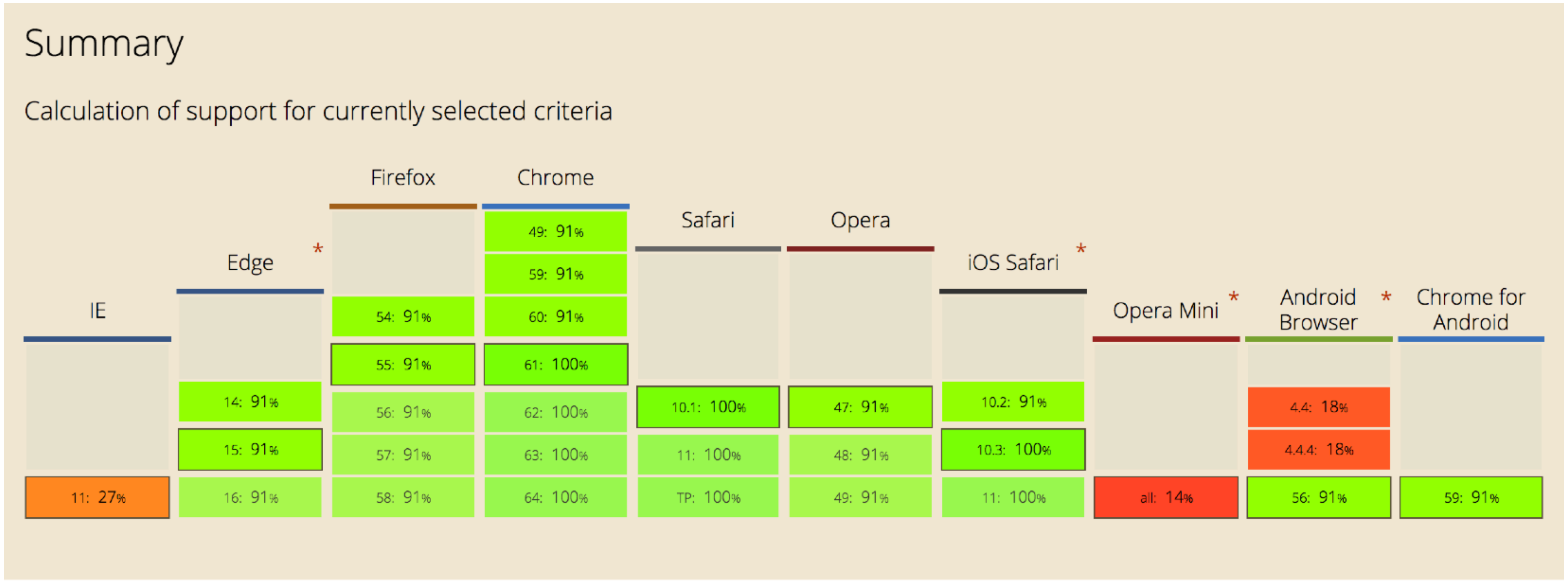
To **bundle** your modules **as if** in production, run:

```
webpack -p
```

This will create a **dist/bundle.js** file usable for production **BUT** if you use arrow functions, webpack needs **Babel**

Can I Use ES6?

Modern browsers implement **95%+** of ES6



Summary

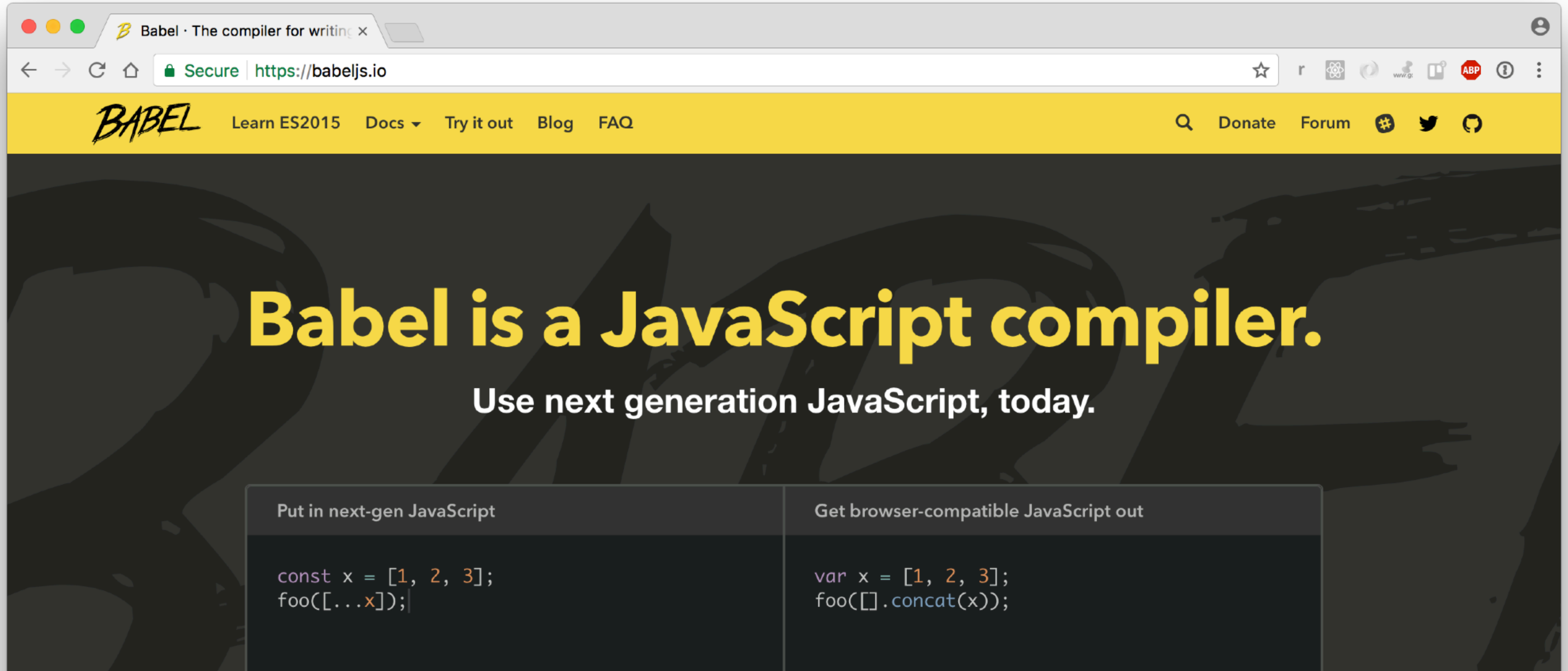
Calculation of support for currently selected criteria

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49: 91%						
			59: 91%						
		54: 91%	60: 91%						
		55: 91%	61: 100%						
	14: 91%	56: 91%	62: 100%	10.1: 100%	47: 91%	10.2: 91%		4.4: 18%	
	15: 91%	57: 91%	63: 100%	11: 100%	48: 91%	10.3: 100%		4.4.4: 18%	
11: 27%	16: 91%	58: 91%	64: 100%	TP: 100%	49: 91%	11: 100%	all: 14%	56: 91%	59: 91%

Babel

To reach **100%**...

Babel compiles **ES6** into **ES5**



Babel is a JavaScript compiler.

Use next generation JavaScript, today.

Put in next-gen JavaScript

```
const x = [1, 2, 3];
foo([...x]);
```

Get browser-compatible JavaScript out

```
var x = [1, 2, 3];
foo([].concat(x));
```

Setup

```
yarn add babel-core babel-preset-es2015 --dev
echo '{ "presets": [ "es2015" ] }' > .babelrc

yarn add babel-loader --dev # For webpack
```

Open **webpack.config.js** and add a "module" key:

```
// [...]
  module: {
    loaders: [{
      test: /\.js$/,
      exclude: /node_modules/,
      loader: 'babel-loader'
    }]
  }
```

Your turn!

/lewagon/react-redux-challenges