Placeholder page 1 of 2

Placeholder page 2 of 2

- **Presentation is running**

ES5 → ES6

[HoneyBook Tech Talk: Jan 4 2017]

I'm John Haugeland

In this talk

In this talk

- **What is ES6**

In this talk

- What is ES6
- **Why do we care about ES6**

In this talk

- What is ES6
- Why do we care
- **Code examples**

In this talk

- What is ES6
- Why do we care
- **Code examples**
  - **We'll cover `destructuring`, `arrows`, `const`, `let`, `template strings`, `collapsed properties`, `spread`, `rest`, `default`, `modules`, and `tail calls`**

In this talk

- What is ES6
- Why do we care
- **Code examples**
  - **We'll cover `destructuring`, `arrows`, `const`, `let`, `template strings`, `collapsed properties`, `spread`, `rest`, `default`, `modules`, and `tail calls`**
  - **No time for `symbol`, `proxy`, `classes`, `generators`, `iterators`, `promises`, or `reflect`**

In this talk

- What is ES6
- Why do we care
- Code examples
  - We'll cover `destructuring`, `arrows`, `const`, `let`, `template strings`, `collapsed properties`, `spread`, `rest`, `default`, `modules`, and `tail calls`
  - No time for `symbol`, `proxy`, `classes`, `generators`, `iterators`, `promises`, or `reflect`
- **How does ES6 help us**

In this talk

- What is ES6
- Why do we care
- Code examples
  - We'll cover `destructuring`, `arrows`, `const`, `let`, `template strings`, `collapsed properties`, `spread`, `rest`, `default`, `modules`, and `tail calls`
  - No time for `symbol`, `proxy`, `classes`, `generators`, `iterators`, `promises`, or `reflect`
- **How does ES6 help us**
  - **do work faster**

In this talk

- What is ES6
- Why do we care
- Code examples
  - We'll cover `destructuring`, `arrows`, `const`, `let`, `template strings`, `collapsed properties`, `spread`, `rest`, `default`, `modules`, and `tail calls`
  - No time for `symbol`, `proxy`, `classes`, `generators`, `iterators`, `promises`, or `reflect`
- **How does ES6 help us**
  - **do work faster**
  - **find bugs faster**

In this talk

- What is ES6
- Why do we care
- Code examples
  - We'll cover `destructuring`, `arrows`, `const`, `let`, `template strings`, `collapsed properties`, `spread`, `rest`, `default`, `modules`, and `tail calls`
  - No time for `symbol`, `proxy`, `classes`, `generators`, `iterators`, `promises`, or `reflect`
- **How does ES6 help us**
  - **do work faster**
  - **find bugs faster**
  - **better re-use code**

In this talk

- What is ES6
- Why do we care
- Code examples
  - We'll cover `destructuring`, `arrows`, `const`, `let`, `template strings`, `collapsed properties`, `spread`, `rest`, `default`, `modules`, and `tail calls`
  - No time for `symbol`, `proxy`, `classes`, `generators`, `iterators`, `promises`, or `reflect`
- **How does ES6 help us**
  - **do work faster**
  - **find bugs faster**
  - **better re-use code**
  - **do better deploys**

In this talk

- What is ES6
- Why do we care
- Code examples
  - We'll cover `destructuring`, `arrows`, `const`, `let`, `template strings`, `collapsed properties`, `spread`, `rest`, `default`, `modules`, and `tail calls`
  - No time for `symbol`, `proxy`, `classes`, `generators`, `iterators`, `promises`, or `reflect`
- **How does ES6 help us**
  - **do work faster**
  - **find bugs faster**
  - **better re-use code**
  - **do better deploys**
  - **ditch existing libs**

In this talk

- What is ES6
- Why do we care
- Code examples
  - We'll cover `destructuring`, `arrows`, `const`, `let`, `template strings`, `collapsed properties`, `spread`, `rest`, `default`, `modules`, and `tail calls`
  - No time for `symbol`, `proxy`, `classes`, `generators`, `iterators`, `promises`, or `reflect`
- How does ES6 help us
  - do work faster
  - find bugs faster
  - better re-use code
  - do better deploys
  - ditch existing libs
- **What are some downsides?**

In this talk

- What is ES6
- Why do we care
- Code examples
  - We'll cover `destructuring`, `arrows`, `const`, `let`, `template strings`, `collapsed properties`, `spread`, `rest`, `default`, `modules`, and `tail calls`
  - No time for `symbol`, `proxy`, `classes`, `generators`, `iterators`, `promises`, or `reflect`
- How does ES6 help us
  - do work faster
  - find bugs faster
  - better re-use code
  - do better deploys
  - ditch existing libs
- **What are some downsides?**

In this talk

- What is ES6
- Why do we care
- Code examples
  - We'll cover `destructuring`, `arrows`, `const`, `let`, `template strings`, `collapsed properties`, `spread`, `rest`, `default`, `modules`, and `tail calls`
  - No time for `symbol`, `proxy`, `classes`, `generators`, `iterators`, `promises`, or `reflect`
- How does ES6 help us
  - do work faster
  - find bugs faster
  - better re-use code
  - do better deploys
  - ditch existing libs
- What are some downsides?
- **How do we get started**

In this talk

- What is ES6
- Why do we care
- Code examples
  - We'll cover `destructuring`, `arrows`, `const`, `let`, `template strings`, `collapsed properties`, `spread`, `rest`, `default`, `modules`, and `tail calls`
  - No time for `symbol`, `proxy`, `classes`, `generators`, `iterators`, `promises`, or `reflect`
- How does ES6 help us
  - do work faster
  - find bugs faster
  - better re-use code
  - do better deploys
  - ditch existing libs
- What are some downsides?
- How do we get started
- **In summary**

Propaganda site

What is ES6

ECMA-262 Version 6.0, June 2015

What is ES6

- **Short version: it's new, useful, easier tools**

In this talk

What is ES6

What is ES6

- **Short version: it's new, useful, easier tools**

ES5 approach:

```
var container = [1,2,3,4,5],
    result;
```

```
function double(x) {
  return x*2;
}
```

```
for (var i=0, iC<container.length; i<iC; ++i) {
  result.push(double(container[i]));
}
```

In this talk

What is ES6

What is ES6

- **Short version: it's new, useful, easier tools**

ES6 approach:

```
var container = [1,2,3,4,5],
    result    = container.map(x => x*2);
```

In this talk

What is ES6

What is ES6

- Short version: it's new, useful, easier tools
- **Javascript comes in versions**

What is ES6

- Short version: it's new, useful, easier tools
- Javascript comes in versions
- **Modern browsers all do (almost) all of EcmaScript-262 Version 5**

In this talk

What is ES6

What is ES6

- Short version: it's new, useful, easier tools
- Javascript comes in versions
- **Modern browsers all do (almost) all of ~~EcmaScript-262 Version 5~~ ES5**
  - **Because that's horrible to write, we say ES5**

What is ES6

- Short version: it's new, useful, easier tools
- Javascript comes in versions
- **Modern browsers all do (almost) all of ES5**
  - **ES5 was originally ES3.1; renamed in July 2008**

In this talk

What is ES6

What is ES6

- Short version: it's new, useful, easier tools
- Javascript comes in versions
- **Modern browsers all do (almost) all of ES5**
  - **ES5 was originally ES3.1; renamed in July 2008**
  - **ES5 resulted from the political failure of ES4**

What is ES6

- Short version: it's new, useful, easier tools
- Javascript comes in versions
- **Modern browsers all do (almost) all of ES5**
  - **ES5 was originally ES3.1; renamed in July 2008**
  - **ES5 resulted from the political failure of ES4**
  - **ES5.1 came out in 2011; tiny changes**

In this talk

What is ES6

What is ES6

- Short version: it's new, useful, easier tools
- Javascript comes in versions
- **Modern browsers all do (almost) all of ES5**
  - **ES5 was originally ES3.1; renamed in July 2008**
  - **ES5 resulted from the political failure of ES4**
  - **ES5.1 came out in 2011; tiny changes**
  - **JavaScript stagnated until 2015**

What is ES6

- Short version: it's new, useful, easier tools
- Javascript comes in versions
- **Modern browsers all do (almost) all of ES5**
  - **ES5 was originally ES3.1; renamed in July 2008**
  - **ES5 resulted from the political failure of ES4**
  - **ES5.1 came out in 2011; tiny changes**
  - **JavaScript stagnated until 2015**
  - **`6to5` let community move on without ECMA**

In this talk

What is ES6

What is ES6

- Short version: it's new, useful, easier tools
- Javascript comes in versions
- **Modern browsers all do (almost) all of ES5**
  - **ES5 was originally ES3.1; renamed in July 2008**
  - **ES5 resulted from the political failure of ES4**
  - **ES5.1 came out in 2011; tiny changes**
  - **JavaScript stagnated until 2015**
  - **`6to5` let community move on without ECMA**
  - **ES6 emerged from community, not standards**

What is ES6

- Short version: it's new, useful, easier tools
- Javascript comes in versions
- **Modern browsers all do (almost) all of ES5**
  - **ES5 was originally ES3.1; renamed in July 2008**
  - **ES5 resulted from the political failure of ES4**
  - **ES5.1 came out in 2011; tiny changes**
  - **JavaScript stagnated until 2015**
  - **`6to5` let community move on without ECMA**
  - **ES6 emerged from community, not standards**
  - **22 major changes**

In this talk

What is ES6

What is ES6

- Short version: it's new, useful, easier tools
- Javascript comes in versions
- Modern browsers all do (almost) all of ES5
    - ES3.1; 4; 5.1; 2015; 6to5; community; 22
- **Modern browsers all do most of ES6**

What is ES6

- Short version: it's new, useful, easier tools
- Javascript comes in versions
- Modern browsers all do (almost) all of ES5
  - ES3.1; 4; 5.1; 2015; 6to5; community; 22
- **Modern browsers all do most of ES6**
  - `Edge 14` **93%**
  - `FF50` **92%**
  - `Chrome 55`: **97%**
  - `Safari 10`: **100%**
  - `iOS 10 Safari`: **100%**
  - `Node 7`: **97%**
  - **Reference: Kangax**

In this talk

What is ES6

What is ES6

- Short version: it's new, useful, easier tools
- Javascript comes in versions
- Modern browsers all do (almost) all of ES5
  - ES3.1; 4; 5.1; 2015; 6to5; community; 22
- Modern browsers all do most of ES6
  - 92-100% coverage
- **Babel shims raise all of those to** `>= 98.8%`

Why do we care about ES6

Because we want to be better; faster; lazier

In this talk

What is ES6

Why care

Why do we care about ES6

- **This will help us move faster in new work**

In this talk

What is ES6

Why care

Why do we care about ES6

- This will help us move faster in new work
- **This will help us find bugs faster**

In this talk

What is ES6

Why care

Why do we care about ES6

- This will help us move faster in new work
- This will help us find bugs faster
- **This will help us do a better job re-using existing code**

In this talk

What is ES6

Why care

Why do we care about ES6

- This will help us move faster in new work
- This will help us find bugs faster
- This will help us do a better job re-using existing code
- **This will help us make better deploys**

In this talk

What is ES6

Why care

Why do we care about ES6

- This will help us move faster in new work
- This will help us find bugs faster
- This will help us do a better job re-using existing code
- This will help us make better deploys
- **This will help us ditch some libraries we use**

In this talk

What is ES6

Why care

Why do we care about ES6

- This will help us move faster in new work
- This will help us find bugs faster
- This will help us do a better job re-using existing code
- This will help us make better deploys
- This will help us ditch some libraries we use
- **This is where the world is going**

In this talk

What is ES6

Why care

Why do we care about ES6

- This will help us move faster in new work
- This will help us find bugs faster
- This will help us do a better job re-using existing code
- This will help us make better deploys
- This will help us ditch some libraries we use
- **This is where the world is going**
  - **We'll need this for other tools sooner or later**

In this talk

What is ES6

Why care

Why do we care about ES6

- This will help us move faster in new work
- This will help us find bugs faster
- This will help us do a better job re-using existing code
- This will help us make better deploys
- This will help us ditch some libraries we use
- **This is where the world is going**
  - **We'll need this for other tools sooner or later**
  - **We already do, for `react`**

In this talk

What is ES6

Why care

Why do we care about ES6

- This will help us move faster in new work
- This will help us find bugs faster
- This will help us do a better job re-using existing code
- This will help us make better deploys
- This will help us ditch some libraries we use
- **This is where the world is going**
  - **We'll need this for other tools sooner or later**
  - **We already do, for `react`**
  - **Many other tools, like `koa`, `jspm` / `systemjs`, `rollup`, `es7-membrane`, `angular 2`, `flowtype`, `typescript` also need it**

In this talk

What is ES6

Why care

Why do we care about ES6

- This will help us move faster in new work
- This will help us find bugs faster
- This will help us do a better job re-using existing code
- This will help us make better deploys
- This will help us ditch some libraries we use
- **This is where the world is going**
  - **We'll need this for other tools sooner or later**
  - **We already do, for `react`**
  - **Many other tools also need it**
  - **Hiring is easier in modern tooling**

In this talk

What is ES6

Why care

Why do we care about ES6

- This will help us move faster in new work
- This will help us find bugs faster
- This will help us do a better job re-using existing code
- This will help us make better deploys
- This will help us ditch some libraries we use
- **This is where the world is going**
  - **We'll need this for other tools sooner or later**
  - **We already do, for `react`**
  - **Many other tools also need it**
  - **Hiring is easier in modern tooling**

I'll justify these after the examples. But, first, *let's see some of the actual code*.

Code examples

Finally, some actual things to learn!

In this talk

What is ES6

Why care

Examples!

Code examples

One by one, let's compare the `ES5` and the `ES6` way.

In this talk

What is ES6

Why care

Examples!
  Destructuring

Destructuring

(aka pattern matching)

In this talk

What is ES6

Why care

Examples!

Destructuring

Destructuring

In this talk

What is ES6

Why care

Examples!
Destructuring

Destructuring

Destructuring assignment (called `pattern matching` in some languages) is a much easier way to get values out of structures, with less boilerplate.

In this talk

What is ES6

Why care

Examples!

Destructuring

Destructuring

Destructuring assignment (called `pattern matching` in some languages) is a much easier way to get values out of structures, with less boilerplate.

```
function makeRectangle() {
  return { width: 5, height: 10, color: 'blue' };
}
```

```
const { width, height } = makeRectangle();
```

In this talk

What is ES6

Why care

Examples!

Destructuring

Destructuring

You may also destructure deep datastructures.

```
const person = {
  name: {
    first  : 'Bob',
    last   : 'Dobbs'
  },
  job: {
    title  : 'Subgenius',
    salary : 80000,
    site   : 'Paris'
  }
};
```

```
function GetTitleLine(personData) {
  { name: { first, last }, job: { title } } = personData;
  return first + ' ' + last + "(" + title + ")";
}
```

In this talk

What is ES6

Why care

Examples!

Destructuring

Destructuring

You can also destructure arrays.

```
const someData     = [4,1,7,2,6,3,5],
      [pivot, ...] = whatever,         /* pivot is now 4 */
      [a,b,c, ...] = whatever;         /* b is 1; c is 7 */
```

In this talk

What is ES6

Why care

Examples!

Destructuring

Destructuring

You can also destructure arrays.

```
const someData     = [4,1,7,2,6,3,5],
      [pivot, ...] = whatever,         /* pivot is now 4 */
      [a,b,c, ...] = whatever;         /* b is 1; c is 7 */
```

We'll get back to this example when we do the `spread` operator. This dovetails nicely with other stuff.

In this talk

What is ES6

Why care

Examples!

Destructuring

Destructuring

You can *also* destructure in the arguments of a function.

```
const person = {
  name: {
    first  : 'Bob',
    last   : 'Dobbs'
  },
  job: {
    title  : 'Subgenius',
    salary : 80000,
    site   : 'Paris'
  }
};
```

```
function Title({ name: { first, last }, job: { title } }) {
  return first + ' ' + last + "(" + title + ")";
}
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows

Arrow functions

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows

Arrow functions

Arrow functions are anonymous functions that lexically bind `this` automatically. They also have an interesting notation which allows you to throw away most of the code.

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows

Arrow functions

Arrow functions are anonymous functions that lexically bind `this` automatically. They also have an interesting notation which allows you to throw away most of the code.

ES5:

```
const foo = function(x) { return x*2; }
```

ES6:

```
const foo = (x) => { return x*2; }
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows

Arrow functions

If an arrow has exactly one statement, you may write it as an expression, instead. This means no `{ }` block, and no need for a `return`.

ES5:

```
const foo = function(x) { return x*2; }
```

ES6:

```
const foo = (x) => x*2;
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows

Arrow functions

If an arrow takes exactly one argument, you may also discard the parentheses around the argument. (Not with multiple or zero, though.)

ES5:

```
const foo = function(x) { return x*2; }
```

ES6:

```
const foo = x => x*2;
```

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows

Arrow functions

Arrows are ***super useful*** with destructuring.

```
const area = ({width, height}) => width * height;
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows

Arrow functions

Arrows are ***super useful*** with destructuring.

```
const area = ({width, height}) => width * height;
```

```
[
  { width: 4, height: 2, color: blue },
  { width: 3, height: 3, border: false },
  { width: 5, height: 1, opacity: 0.5 },
].map(
  rekt => console.log(area(rekt))
);
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows

Arrow functions

As a result, you can write code like

```
const top5_salary_sum_daily =
  people
    .filter( person  => person.alive        )
    .filter( person  => person.employed     )
    .sort  ( (a,b)   => a.salary - b.salary )  /* reverse */
    .slice(5)
    .map   ( person  => salary / 365        )
    .reduce( (acc,n) => acc + n             ); /* sum */
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows

Member `Bob Dobbs`?

```
function Title({ name: { first, last }, job: { title } }) {
  return first + ' ' + last + "(" + title + ")";
}
```

becomes

```
const Title = ({ name: { first, last }, job: { title } }) =>
  first + ' ' + last + "(" + title + ")";
```

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows

Arrow functions

Lexical `this` means you don't need to `.bind`.

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows

Arrow functions

Lexical `this` means you don't need to `.bind`.

Given

```
const someObj = { val: 25 };
const caller  = { val: 50, makeCall: () => someObj.doCall() }
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows

Arrow functions

Lexical `this` means you don't need to `.bind`.

Given

```
const someObj = { val: 25 };
const caller  = { val: 50, makeCall: () => someObj.doCall() };
```

This will return 50, not 25:

```
/* `this` refers to the caller, not someObj */
someObj.doCall = function() { return this.val; }
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows

Arrow functions

Lexical `this` means you don't need to `.bind`.

Given

```
const someObj = { val: 25 };
const caller  = { val: 50, makeCall: () => someObj.doCall() }
```

This will return 50, not 25:

```
/* `this` refers to the caller, not someObj */
someObj.doCall = function() { return this.val; }
```

This gives 25 like most people expect:

```
someObj.doCall = function() { return this.val; }.bind(someObj
```

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows

Arrow functions

Lexical `this` means you don't need to `.bind`.

Given

```
const someObj = { val: 25 };
const caller  = { val: 50, makeCall: () => someObj.doCall() }
```

This will return 50, not 25:

```
/* `this` refers to the caller, not someObj */
someObj.doCall = function() { return this.val; }
```

This gives 25 like most people expect:

```
someObj.doCall = function() { return this.val; }.bind(someObj
```

It's just handled with arrows, though.

```
someObj.doCall = () => this.val;
```

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows

Arrow function gotcha

This is Javascript, which is why we can't have nice things.

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows

Arrow function gotcha

This is Javascript, which is why we can't have nice things.

If your arrow returns an `object`, which also uses `{}` to delimit, the arrow is going to think you're writing a block.

```
/ error: this is a block, not an object /
const dagnabbit = () => { foo: 'bar' };
```

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows

Arrow function gotcha

This is Javascript, which is why we can't have nice things.

If your arrow returns an `object`, which also uses `{}` to delimit, the arrow is going to think you're writing a block.

```
// error: this is a block, not an object //
const dagnabbit = () => { foo: 'bar' };
```

You'll need to write

```
const dagnabbit = () => { return { foo: 'bar' }; }
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const

Const

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const

Const

Stop letting things change that shouldn't change!

```
const size = 5;
...
size = 10;
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const

Const

Stop letting things change that shouldn't change!

```
const size = 5;
...
size = 10;
```

In this talk

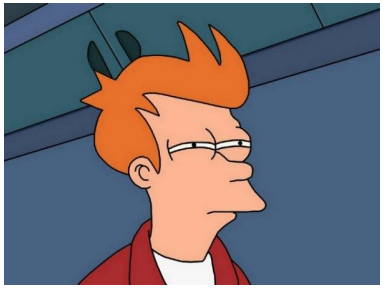
What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const

Const

At Tilt, we thought this wasn't going to be a big deal. We had a low-bug frontend, and we were smart programmers who didn't do silly things with name re-use.



In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const

Const

At Tilt, we thought this wasn't going to be a big deal. We had a low-bug frontend, and we were smart programmers who didn't do silly things with name re-use.

Moving from `var` to `const` ended up turning up almost 40 bugs basically overnight.

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const

Const

At Tilt, we thought this wasn't going to be a big deal. We had a low-bug frontend, and we were smart programmers who didn't do silly things with name re-use.

Moving from `var` to `const` ended up turning up almost 40 bugs basically overnight.

We can get tooling support from `eslint`, to help us find these. This doesn't have to be an archaeological dig.

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let

Let

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let

Let

`let` is just `var` with less-stupid binding rules.

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let

Let

Let's see who's feeling froggy.

```
for (var i=0; i<5; ++i) {
  document.getElementById('button' + i.toString()).onClick =
    () => window.alert('Button ' + i + ' pressed!');
}
document.getElementById('button3').click();
```

What message gets alerted?

In this talk

What is ES6

Why care

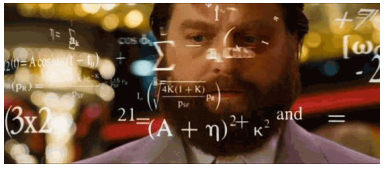
Examples!
- Destructuring
- Arrows
- Const
- Let

Let

```
for (var i=0; i<5; ++i) {
  document.getElementById('button' + i.toString()).onClick =
    () => window.alert('Button ' + i + ' pressed!');
}
document.getElementById('button3').click();
```

The message will be "Button 5 pressed!"

In fact, it'll say that for all of them.



In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let

Let

```
for (var i=0; i<5; ++i) {
  document.getElementById('button' + i.toString()).onClick =
    () => window.alert('Button ' + i + ' pressed!');
}
document.getElementById('button3').click();
```

The problem is that what's being bound in that function is the actual variable, rather than the value. So, as the for loop updates the value, the functions just check the variable, and get the new value.

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let

Let

```
for (let i=0; i<5; ++i) {
  document.getElementById('button' + i.toString()).onClick =
    () => window.alert('Button ' + i + ' pressed!');
}
document.getElementById('button3').click();
// because of let i, now gets 3 like you'd expect
```

The problem is that what's being bound in that function is the actual variable, rather than the value. So, as the for loop updates the value, the functions just check the variable, and get the new value.

If you change the `var` to `let`, then the declaration will be remade for each pass of the loop (like most people expect,) and each handler will get its own, no longer updated copy.

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let

Let

`let` also disallows redefinition, which (augh) `var` does not.

```
var a = 3;
var a = 5; // legal; source of bugs when far apart
```

```
let b = 3;
let b = 5; // this is an error, thank ${religion.prophet}
```

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let

Let

`let` also disallows redefinition, which (augh) `var` does not.

```
var a = 3;
var a = 5; // legal; source of bugs when far apart
```

```
let b = 3;
let b = 5; // this is an error, thank ${religion.prophet}
```

The number of bugs this turned up at Womply was startling and gross.

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let

Let

More strictly, `let` is local scoped, whereas `var` is block scoped. That is, `var` is scoped to the block it's in, which is generally the top level function.

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let

Let

More strictly, `let` is local scoped, whereas `var` is block scoped. That is, `var` is scoped to the block it's in, which is generally the top level function.

Block scoping leaks.

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let

Let

More strictly, `let` is local scoped, whereas `var` is block scoped. That is, `var` is scoped to the block it's in, which is generally the top level function.

Block scoping leaks.

```
function foo() {
  for (var i=0; i<10; ++i) { whatever(); }
  console.log(i);  // 10
}
```

```
function bar() {
  for (let i=0; i<10; ++i) { whatever(); }
  console.log(i);  // undefined
}
```

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let

Let

More strictly, `let` is local scoped, whereas `var` is block scoped. That is, `var` is scoped to the block it's in, which is generally the top level function.

Block scoping leaks.

```
function foo() {
  for (var i=0; i<10; ++i) { whatever(); }
  console.log(i);  // 10
}
```

```
function bar() {
  for (let i=0; i<10; ++i) { whatever(); }
  console.log(i);  // undefined
}
```

This can cause weird bugs, especially around shadowing (this was in a PR here the other day, and ***does happen semi-frequently*** in the real world.)

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let

Let

There is almost no good reason to use `var` anymore.

If it's constant (it usually is,) use `const`.

Otherwise, please use `let`. It prevents problems.

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let
- Templates

Template String Literals

We now have a notation for inlining. Backticks make a "template string." Within a template string, expressions like `${foo}` are evaluated ***at assignment time***.

```
var whom = "George",
    what = `I had lunch with ${whom}.`;
```

These interpolations need not be other variables, or literals; they may be any expression.

```
const seq  = ['bob', 'linda', 'bill', 'mayhem'],
      text = `We invited: ${ seq.join(', ') }!`;
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates

Template String Literals

Template string literals may also be multi-line. To replace:

ES5:

```
function pageframe(title, menu, content) {
  const mitems = menu.join('</li><li>')
  return
'<!doctype>\
<html>\
  <head>\
    <title>' + title + '</title>\
  </head>\
  <body>\
    <ul class="menu"><li>' + mitems + '</li></ul>\
    ' + content + '\
  </body>\
</html>';
}
```

(lol the highlighter can't even render this)

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates

Template String Literals

You could instead write:

ES6:

```
const pageframe = (title, menu, content) =>
`<!doctype>
<html>
  <head>
    <title>${title}</title>
  </head>
  <body>
    <ul class="menu"><li>${menu.join('</li><li>')}</li></ul>
    ${content}
  </body>
</html>
}`;
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates

Member `Bob Dobbs`?

```
const Title = ({ name: { first, last }, job: { title } }) =>
  first + ' ' + last + "(" + title + ")";
}
```

becomes

```
const Title = ({ name: { first, last }, job: { title } }) =>
  `${first} ${last} (${title})`;
```

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let
- Templates
- Export

Import/export

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let
- Templates
- Export

Import/export

The module loader we deserve

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates
- Export

Import/export

The module loader we deserve

Will work in browsers Real Soon Now ™

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let
- Templates
- Export

Import/export

The module loader we deserve

Will work in browsers Real Soon Now ™

About the code, not the implementation. Can easily be converted to current tooling.

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let
- Templates
- Export

Import/export

The module loader we deserve

Will work in browsers Real Soon Now ™

About the code, not the implementation. Can easily be converted to current tooling.

Feels like a nicer version of `require` from `node CommonJS`.

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let
- Templates
- Export

Import/export

The module loader we deserve

Will work in browsers Real Soon Now ™

About the code, not the implementation. Can easily be converted to current tooling.

Feels like a nicer version of `require` from `node CommonJS`.

Already works in our stack.

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let
- Templates
- Export

Import/export

This is super useful for breaking code up and making it modular, without shenanigans.

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let
- Templates
- Export

Import/export

This is super useful for breaking code up and making it modular, without shenanigans.

Given:

```
const Bob = { name: {first: 'Bob', last: 'Dobbs'} };
```

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let
- Templates
- Export

Import/export

This is super useful for breaking code up and making it modular, without shenanigans.

Given:

```
const Bob = { name: {first: 'Bob', last: 'Dobbs'} };
```

Utility module:

```
const getname({ name: {first, last} }) => `${first} ${last}`;
export { getname: getname };
```

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let
- Templates
- Export

Import/export

This is super useful for breaking code up and making it modular, without shenanigans.

Given:

```
const Bob = { name: {first: 'Bob', last: 'Dobbs'} };
```

Utility module:

```
const getname({ name: {first, last} }) => `${first} ${last}`;
export { getname: getname };
```

Dependent code:

```
import { getname } from './js/util.js';
console.log(getname(Bob));
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse

Collapsed properties

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse

Collapsed properties

This is kind of a minor thing, but it actually makes stuff a lot more readable, so, whatever.

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse

Collapsed properties

This is kind of a minor thing, but it actually makes stuff a lot more readable, so, whatever.

You know how in `SGML` you can skip the value of a property when it's the same as the key?

```
<input disabled="disabled"/>
<input disabled/>   <!-- same thing -->
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse

Collapsed properties

This is kind of a minor thing, but it actually makes stuff a lot more readable, so, whatever.

You know how in `SGML` you can skip the value of a property when it's the same as the key?

```
<input disabled="disabled"/>
<input disabled/>   <!-- same thing -->
```

You can do that in JS now too.

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse

Collapsed properties

This is kind of a minor thing, but it actually makes stuff a lot more readable, so, whatever.

You know how in `SGML` you can skip the value of a property when it's the same as the key?

```
<input disabled="disabled"/>
<input disabled/>   <!-- same thing -->
```

You can do that in JS now too.

```
const a=5, b=false, c="wha";
console.log({ a: a, b: b, c: c });
console.log({ a, b, c });
/* these both say { a: 5, b: false, c: 'wha' } */
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse

Collapsed properties

Given:

```
const foo = 'stuff' + someCall(),
      bar = 12,
      baz = ['what','ever'].join('');
```

Old way:

```
function DoTheThing() {
  return { foo: foo, bar: bar, baz: baz };
}
```

New way:

```
function DoTheThing() {
  return { foo, bar, baz };
}
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse

Member util.js?

Was:

```
const getname({ name: {first, last} }) => `${first} ${last}`;
export { getname: getname };
```

Is now:

```
const getname({ name: {first, last} }) => `${first} ${last}`;
export { getname };
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse

Collapsed properties

Given:

```
const foo = 'stuff' + someCall(),
      bar = 12,
      baz = ['what','ever'].join('');
```

Old way:

```
function DoTheThing() {
  return { foo: foo, bar: bar, baz: baz };
}
```

New way:

```
function DoTheThing() {
  return { foo, bar, baz };
}
```

In this talk

What is ES6

Why care

Examples!
- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse

Collapsed properties

Given:

```
const foo = 'stuff' + someCall(),
      bar = 12,
      baz = ['what','ever'].join('');
```

Old way:

```
function DoTheThing() {
  return { foo: foo, bar: bar, baz: baz };
}
```

New way:

```
function DoTheThing() {
  return { foo, bar, baz };
}
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse
- Tail calls

Tail calls

What's the threat here?

```
function doEach(container) {
  if (!container.length) { return; }
  console.log(container[0]);
  doEach(container.slice(1));
}
```

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse
- Tail calls

Tail calls

What's the threat here?

```
function doEach(container) {
  if (!container.length) { return; }
  console.log(container[0]);
  doEach(container.slice(1));
}
```

Each recursion pushes a new stack frame onto the stack. Eventually the stack runs out.

Stack frames contain the "previous function invokation," which the VM goes back to.

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse
- Tail calls

Tail calls

What's the threat here?

```
function doEach(container) {
  if (!container.length) { return; }
  console.log(container[0]);
  doEach(container.slice(1));
}
```

Tail calls mean there's no stack frame to go back to, so you can recurse safely over huge data.

In this talk

What is ES6

Why care

Examples!

- Destructuring
- Arrows
- Const
- Let
- Templates
- Collapse
- Tail calls
- Defaults

Defaults

ES5:

```
function foo(a, b) {
  a = a || 5;
  b = b || 'whatever';
  console.log(`${a}, ${b}`);
}
```

ES6:

```
function foo(a=5, b='whatever') {
  console.log(`${a}, ${b}`);
}
```

Tail calls mean there's no stack frame to go back to, so you can recurse safely over huge data.

How does ES6 help us

Let's actually get down into some specifics

In this talk

What is ES6

Why care

Examples!

How helped

How does ES6 help us

Lots of ways, it turns out.

Let's group them into five sets.

How does ES6 help us do work faster

Simpler traversals; pattern matching; arrows; functional code; better APIs

In this talk

What is ES6

Why care

Examples!

How helped
Work

Help us do work faster

Some of the features in ES6 can have a surprisingly dramatic impact on productivity.

In this talk

What is ES6

Why care

Examples!

How helped
Work

Help us do work faster

Some of the features in ES6 can have a surprisingly dramatic impact on productivity.

- Template string literals
- Arrow functions
- Destructuring assignment
- The spread operator
- Block scoping through `let`
- `import` / `export`
- Real inheritance
- `get`ters and `set`ters
- `promise`s
- **default values**

In this talk

What is ES6

Why care

Examples!

How helped
Work

Help us do work faster

Some of the features in ES6 can have a surprisingly dramatic impact on productivity.

- Template string literals
- Arrow functions
- Destructuring assignment
- The spread operator
- Block scoping through `let`
- `import` / `export`
- Real inheritance
- `get`ters and `set`ters
- `promise`s
- **default values**

Many classes of code that are common online can see 50-75% reductions in the amount of work needed to implement something. Woo less work!

How does ES6 help us find bugs faster

Stronger rules; proper classes; 'use strict'; removing manual code

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug

Help us find bugs faster

Lots of things have been added to `ES6` that make it easier to find problems.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug

Help us find bugs faster

Lots of things have been added to `ES6` that make it easier to find problems.

- `const` - let's stop letting things change which shouldn't
- `let` - makes `const` declarations in loops tolerable
- arrow lexical `this` - bye bye `.bind` (and `that`)
- typed `Array`s - eliminate many mis-fills
- pseudotype `class`es - `__prototype__` begone!
- default values - better than `||` or `!== undefined`
- `get`ters and `set`ters - as log points
- **`WeakSet` and `WeakMap` - fewer memory leaks for sure**

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug

Help us find bugs faster

Lots of things have been added to `ES6` that make it easier to find problems.

- `const` - let's stop letting things change which shouldn't
- `let` - makes `const` declarations in loops tolerable
- arrow lexical `this` - bye bye `.bind` (and `that`)
- typed `Array`s - eliminate many mis-fills
- pseudotype `class`es - `__prototype__` begone!
- default values - better than `||` or `!== undefined`
- `get`ters and `set`ters - as log points
- **`WeakSet` and `WeakMap` - fewer memory leaks for sure**

ES6 is stricter in important ways than ES5, and gives you the tools to be stricter, too

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug

Help us find bugs faster

Lots of things have been added to `ES6` that make it easier to find problems.

- `const` - let's stop letting things change which shouldn't
- `let` - makes `const` declarations in loops tolerable
- arrow lexical `this` - bye bye `.bind` (and `that`)
- typed `Array`s - eliminate many mis-fills
- pseudotype `class`es - `__prototype__` begone!
- default values - better than `||` or `!== undefined`
- `get`ters and `set`ters - as log points
- **`WeakSet` and `WeakMap` - fewer memory leaks for sure**

ES6 is stricter in important ways than ES5, and gives you the tools to be stricter, too

Plus, our path through `babel` means that `'use strict'` is always on.

How does ES6 help us better re-use code

Better interfaces; compositional mechanisms; tooling allows ES7 import/export

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Help us better re-use code

Re-use is a very big deal.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Help us better re-use code

Re-use is a very big deal.

- The whole team's looking to combine HB and TWS
- Sean and Tracy are working on a reusable style guide
- Jonah has been looking at reusable pricing widgets
- Sean is discussing reusable component boundaries
- I'd like to move towards libraries in `NPM Private`
- Shai is looking at reusable `hb_stats` replacements
- Wendy has a re-use problem with mail
- Re-use was a big part of the "why React Native" talk
- Re-use of frames caused a Salesforce pileup for Abhi and Christina
- **It's probably on all of our minds; these are just the ones I've seen recently**

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Help us better re-use code

Re-use is a very big deal.

- (huge stupid list of examples)
- **It's probably on all of our minds**

The earlier you tackle the mechanics of ***how*** re-use works, the easier it is in the long term.

This is important because re-use is hard.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

What makes this hard?

Lots of the problems you see in earlier module systems, such as `tinyloader`s, `angular 1`, `requirejs`, `amd`, `commonjs`, and `npm commonjs` have to do with their only being designed to handle one build target (frontend, backend, custom module, etc.)

The emergence of `bower` and `systemjs` was too late; instead a second layer of packagers, starting with `browserify` and moving on to `webpack`, `rollup`, and so forth, performed a large physical inclusion with a calling convention to hard-rig portable modules.

They all sucked, and were terrible.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

So what do we do?

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

So what do we do?

`import/export` is **The Truth**.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

So what do we do?

`import/export` is **The Truth**.

`import/export` is **The Way**.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

So what do we do?

`import/export` is **The Truth**.

`import/export` is **The Way**.

`import/export` loves you.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

So what do we do?

`import/export` is **The Truth**.

`import/export` is **The Way**.

`import/export` loves you.

`import/export` just wants you to be happy.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

So what do we do?

`import/export` is **The Truth**.

`import/export` is **The Way**.

`import/export` loves you.

`import/export` just wants you to be happy.

There is **none other** than `import/export`.

In this talk
What is ES6
Why care
Examples!
How helped
Work
Debug
Re-use

So what do we do?

`import/export` is **The Truth**.

`import/export` is **The Way**.

`import/export` loves you.

`import/export` just wants you to be happy.

There is **none other** than `import/export`.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

No seriously, what do we do?

You can write a tinyloader in 20 minutes. Angular's module system was two people for four days. RequireJS was one guy and was working in three days. Browserify was one guy and five days. Webpack was two dudes and a weekend. Rollup was one guy's two week project.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

No seriously, what do we do?

You can write a tinyloader in 20 minutes. Angular's module system was two people for four days. RequireJS was one guy and was working in three days. Browserify was one guy and five days. Webpack was two dudes and a weekend. Rollup was one guy's two week project.

Javascript itself was a six day Brendan Eich hack.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

No seriously, what do we do?

You can write a tinyloader in 20 minutes. Angular's module system was two people for four days. RequireJS was one guy and was working in three days. Browserify was one guy and five days. Webpack was two dudes and a weekend. Rollup was one guy's two week project.

Javascript itself was a six day Brendan Eich hack.

A couple hundred neckbeards fought over `import/export` for **two and a half years**.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

No seriously, what do we do?

You can write a tinyloader in 20 minutes. Angular's module system was two people for four days. RequireJS was one guy and was working in three days. Browserify was one guy and five days. Webpack was two dudes and a weekend. Rollup was one guy's two week project.

Javascript itself was a six day Brendan Eich hack.

A couple hundred neckbeards fought over `import/export` for two and a half years.

`import/export` is much more thoughtfully designed than most of Javascript.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

What's the difference?

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

What's the difference?

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

What's the difference?

The other stuff is about how the packaging works. `import/export` doesn't care.

`import/export` is about how the code interacts with other code. And then, it can be compiled down to `commonjs` or whatever, that `webpack` and `browserify` like, or consumed directly from `rollup` or `systemjs`.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

What's the difference?

The other stuff is about how the packaging works. `import/export` doesn't care.

`import/export` is about how the code interacts with other code. And then, it can be compiled down to `commonjs` or whatever, that `webpack` and `browserify` like, or consumed directly from `rollup` or `systemjs`.

The difference is that `import/export` is defined at the level of a language feature, so more modern tools that are legitimately aware of it, like `rollup`, can look at what's being used, and discard the rest (in a process called `tree shaking`,) which dramatically reduces package sizes.

`webpack` and `browserify` have sort of after-the-fact tree shaking, but since `commonjs` is just a set of assumptions, the environment doesn't have as strong guarantees, and can't make nearly as many exclusions.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Who cares?

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Who cares?

Okay, how about this, instead?

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Who cares?

Okay, how about this, instead?

`import/export` is

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Who cares?

Okay, how about this, instead?

`import/export` is

- **way easier to read,**

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Who cares?

Okay, how about this, instead?

`import/export` is

- way easier to read,
- **way easier to write, and**

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Who cares?

Okay, how about this, instead?

`import/export` is

- way easier to read,
- way easier to write, and
- **will be supported in browsers,**

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Who cares?

Okay, how about this, instead?

`import/export` is

- way easier to read,
- way easier to write, and
- will be supported in browsers,
- **letting you do roughly what `require` does in `node`,**

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Who cares?

Okay, how about this, instead?

`import/export` is

- way easier to read,
- way easier to write, and
- will be supported in browsers,
- letting you do roughly what `require` does in `node`,
- **but portably on the frontend and the backend,**

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Who cares?

Okay, how about this, instead?

`import/export` is

- way easier to read,
- way easier to write, and
- will be supported in browsers,
- letting you do roughly what `require` does in `node`,
- but portably on the frontend and the backend,
- **and without any tooling at all?**

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Who cares?

Okay, how about this, instead?

`import/export` is

- way easier to read,
- way easier to write, and
- will be supported in browsers,
- letting you do roughly what `require` does in `node`,
- but portably on the frontend and the backend,
- **and without any tooling at all?**

Because it'll finally be an actual part of the language.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

Who cares?

Okay, how about this, instead?

`import/export` is

- way easier to read,
- way easier to write, and
- will be supported in browsers,
- letting you do roughly what `require` does in `node`,
- but portably on the frontend and the backend,
- **and without any tooling at all?**

Because it'll finally be an actual part of the language.

```
const is_it_nice = true;
export { is_it_nice };
```

```
import { is_it_nice } from './its_nice.js';
```

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

How much work is it?

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

How much work is it?

Zero.

Babel converts it to `commonjs require`, then `webpack` (or whatever) bundles it.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

How much work is it?

Zero.

Babel converts it to `commonjs require`, then `webpack` (or whatever) bundles it.

So just use it and you're done.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use

How much work is it?

Zero.

Babel converts it to `commonjs require`, then `webpack` (or whatever) bundles it.

So just use it and you're done.

There's even a predictable name so that you can use it with `require` from the outside.

How does ES6 help us do better deploys

Better packaging; less repetition; easier externality; tree shaking

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use
- Deploy

Help us do better deploys

All that packaging stuff means that packagers can do a better job of

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use
- Deploy

Help us do better deploys

All that packaging stuff means that packagers can do a better job of

- **excluding duplicates**

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use
- Deploy

Help us do better deploys

All that packaging stuff means that packagers can do a better job of

- excluding duplicates
- **removing unused code**

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use
- Deploy

Help us do better deploys

All that packaging stuff means that packagers can do a better job of

- excluding duplicates
- removing unused code
- **preventing version contrast**

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use
- Deploy

Help us do better deploys

All that packaging stuff means that packagers can do a better job of

- excluding duplicates
- removing unused code
- **preventing version contrast**

This means smaller deploy packages

How does ES6 help us ditch existing libs

You might not need: jQuery, underscore, bluebird, ...

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use
- Deploy
- Ditch

Help us ditch existing libs

We've all seen the trendy websites, of the form *you might not need jQuery*, yes?

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use
- Deploy
- Ditch

Help us ditch existing libs

We've all seen the trendy websites, of the form *you might not need jQuery*, yes?

We all remember when Koçulu rage-quit NPM and broke the internet, yes?

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use
- Deploy
- Ditch

Help us ditch existing libs

We've all seen the trendy websites, of the form *you might not need jQuery*, yes?

We all remember when Koçulu rage-quit NPM and broke the internet, yes?

We all know that programmer who includes `underscore.js` to get object cloning, yes?

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use
- Deploy
- Ditch

Help us ditch existing libs

We've all seen the trendy websites, of the form *you might not need jQuery*, yes?

We all remember when Koçulu rage-quit NPM and broke the internet, yes?

We all know that programmer who includes `underscore.js` to get object cloning, yes?

`Libraries are liabilities`

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use
- Deploy
- Ditch

Help us ditch existing libs

We've all seen the trendy websites, of the form *you might not need jQuery*, yes?

We all remember when Koçulu rage-quit NPM and broke the internet, yes?

We all know that programmer who includes `underscore.js` to get object cloning, yes?

```
Libraries are liabilities
```

Important libraries - difficult work, like `react`, or `d3`, or `processing`, are great.

In this talk

What is ES6

Why care

Examples!

How helped
- Work
- Debug
- Re-use
- Deploy
- Ditch

Help us ditch existing libs

We've all seen the trendy websites, of the form *you might not need jQuery*, yes?

We all remember when Koçulu rage-quit NPM and broke the internet, yes?

We all know that programmer who includes `underscore.js` to get object cloning, yes?

```
Libraries are liabilities
```

Important libraries - difficult work, like `react`, or `d3`, or `processing`, are great.

But including frivolous libraries for convenience functions has a ***high*** cost, and so when ES6 gives you alternatives, ***ditch the library***.

What are some downsides?

Babel; the build organization; new thinking; old code needs vetting; angular 1 modules; code style transition; learning to hate ES5; shim weight

In this talk
What is ES6
Why care
Examples!
How helped
Downsides

What are some downsides?

- **Babel**

In this talk

What is ES6

Why care

Examples!

How helped

Downsides
Babel

What are some downsides?

- **Babel**
  - `Babel` **is honestly pretty heavy**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - **`Babel` is honestly pretty heavy**
    - **104k uncompressed, 50-80k compressed**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - `Babel` **is heavy (50-80k)**
  - `Babel` **'s shims aren't dynamic, and have significant speed overhead**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - `Babel` **is heavy (50-80k)**
  - `Babel` **'s shims aren't dynamic, and have significant speed overhead**
    - **Real world hits of ~30-40% are realistic**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - **`Babel` is heavy (50-80k)**
  - **`Babel`'s shims are 30-40% slower**
  - **`Babel` shims things that don't need to be shimmed in evergreen browsers**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - **`Babel` is heavy (50-80k)**
  - **`Babel`'s shims are 30-40% slower**
  - **`Babel` shims things that don't need to be shimmed in evergreen browsers**
    - **This is removable when it's time, so it's still better than a polyfill**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - `Babel` **is heavy (50-80k)**
  - `Babel` **'s shims are 30-40% slower**
  - `Babel` **over-shims**
  - `Babel` **builds can be slow**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - `Babel` **is heavy (50-80k)**
  - `Babel` **'s shims are 30-40% slower**
  - `Babel` **over-shims**
  - `Babel` **builds can be slow**
    - **Helped by utility builds and** `common.js`

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - `Babel` **is heavy (50-80k)**
  - `Babel` **'s shims are 30-40% slower**
  - `Babel` **over-shims**
  - `Babel` **builds can be slow**
  - `Babel` **has plugins**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - `Babel` **is heavy (50-80k)**
  - `Babel` **'s shims are 30-40% slower**
  - `Babel` **over-shims**
  - `Babel` **builds can be slow**
  - `Babel` **has plugins**
    - **Plugins are the devil, in general**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - `Babel` **is heavy (50-80k)**
  - `Babel` **'s shims are 30-40% slower**
  - `Babel` **over-shims**
  - `Babel` **builds can be slow**
  - `Babel` **has plugins**
    - **Plugins are the devil, in general**
    - **Plugins are how one tool ends up doing 55 jobs**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - `Babel` **is heavy (50-80k)**
  - `Babel` **'s shims are 30-40% slower**
  - `Babel` **over-shims**
  - `Babel` **builds can be slow**
  - `Babel` **has plugins**
    - **Plugins are the devil, in general**
    - **Plugins are how one tool ends up doing 55 jobs**
    - **When one of those jobs needs replacement, You're Screwed ™**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - `Babel` **is heavy (50-80k)**
  - `Babel` **'s shims are 30-40% slower**
  - `Babel` **over-shims**
  - `Babel` **builds can be slow**
  - `Babel` **has plugins**
  - `Babel` **makes coverage, testing slightly gross**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel

What are some downsides?

- **Babel**
  - `Babel` **is heavy (50-80k)**
  - `Babel` **'s shims are 30-40% slower**
  - `Babel` **over-shims**
  - `Babel` **builds can be slow**
  - `Babel` **has plugins**
  - `Babel` **makes coverage, testing slightly gross**
    - **We've already mostly gotten around this**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- **The build organization**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- **The build organization**
  - **When you have ES5 and ES6, you need to distinguish what gets transpiled and what doesn't**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- **The build organization**
  - **Distinguish what gets transpiled**
  - **Day one, this is "what have we made sure is clean ES6"**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- **The build organization**
  - **Distinguish what gets transpiled**
  - **"what have we made sure is clean"**
  - **Day two, this is "what new stuff are we writing in ES6"**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- **The build organization**
  - **Distinguish what gets transpiled**
  - **"what have we made sure is clean"**
  - **"what new stuff are we writing"**
  - **Day three, this is "how do we exclude vendor script"**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- **The build organization**
  - **Distinguish what gets transpiled**
  - **"what have we made sure is clean"**
  - **"what new stuff are we writing"**
  - **"how do we exclude vendor script"**
  - **This is particularly complex in our current layout**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- **The build organization**
  - **Distinguish what gets transpiled**
  - **"what have we made sure is clean"**
  - **"what new stuff are we writing"**
  - **"how do we exclude vendor script"**
  - **Our current layout**
  - **This can also make calling into 6 from 5 complex**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- **The build organization**
  - **Distinguish what gets transpiled**
  - **"what have we made sure is clean"**
  - **"what new stuff are we writing"**
  - **"how do we exclude vendor script"**
  - **Our current layout**
  - **Calling into 6 from 5 can be complex**
  - **This is all handled in our current `gulpfile`.**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- **The build organization**
  - **Distinguish what gets transpiled**
  - **"what have we made sure is clean"**
  - **"what new stuff are we writing"**
  - **"how do we exclude vendor script"**
  - **Our current layout**
  - **Calling into 6 from 5 can be complex**
  - **This is all handled in our current `gulpfile`.**
  - **You don't need to handle any of this.**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- **New thinking**

In this talk

What is ES6

Why care

Examples!

How helped

Downsides
- Babel
- Build
- Thinking

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- **New thinking**
  - **Language changes alter how you think**

In this talk

What is ES6

Why care

Examples!

How helped

Downsides
- Babel
- Build
- Thinking

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- **New thinking**
  - **Language changes alter how you think**
  - **This is honestly kind of the whole point**

In this talk

What is ES6

Why care

Examples!

How helped

Downsides
- Babel
- Build
- Thinking

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- **New thinking**
  - **Language changes alter how you think**
  - **This is honestly kind of the whole point**
  - **But also this costs time and can be frustrating**

In this talk

What is ES6

Why care

Examples!

How helped

Downsides
- Babel
- Build
- Thinking

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- **New thinking**
  - **Language changes alter how you think**
  - **This is honestly kind of the whole point**
  - **But also this costs time and can be frustrating**
  - **Using `.reduce`, etc isn't initially easy**

In this talk

What is ES6

Why care

Examples!

How helped

Downsides
- Babel
- Build
- Thinking
- Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`) are no longer allowed**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **Babel wants `eval()` gone**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **Babel wants `eval()` gone**
    - **Removing `eval()` hurts Angular 1**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **Babel wants `eval()` gone**
    - **Removing `eval()` hurts Angular 1**
    - **Solved by not transpiling Angular 1**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **`eval()` gone (for the best)**
  - **Everything is in `use strict` now**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **`eval()` gone (for the best)**
  - **Everything is in `use strict` now**
    - **Screws with `this`, which is an actual problem**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **`eval()` gone (for the best)**
  - **Everything is in `use strict` now**
    - **Screws with `this`**
    - **Sean already took a couple bullets on `this`**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **`eval()` gone (for the best)**
  - **Everything is in `use strict` now**
  - **Certain extremely weird code is no longer legal**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **`eval()` gone (for the best)**
  - **Everything is in `use strict` now**
  - **Certain extremely weird code is no longer legal**
    - **eg `for (var i = 1 in []) {}`**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **`eval()` gone (for the best)**
  - **Everything is in `use strict` now**
  - **Certain extremely weird code is no longer legal**
    - **eg `for (var i = 1 in []) {}`**
    - **I've never actually heard of this stuff hurting anyone, ever**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **`eval()` gone (for the best)**
  - **Everything is in `use strict` now**
  - **Certain extremely weird code is no longer legal**
  - **Extreme edge cases**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **`eval()` gone (for the best)**
  - **Everything is in `use strict` now**
  - **Certain extremely weird code is no longer legal**
  - **Extreme edge cases**
    - **`Number` accepts octal literals now**

In this talk

What is ES6

Why care

Examples!

How helped

Downsides
- Babel
- Build
- Thinking
- Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **`eval()` gone (for the best)**
  - **Everything is in `use strict` now**
  - **Certain extremely weird code is no longer legal**
  - **Extreme edge cases**
    - **`Number` accepts octal literals now**
    - **Missing time zone interpretation changed**

In this talk

What is ES6

Why care

Examples!

How helped

Downsides
- Babel
- Build
- Thinking
- Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **`eval()` gone (for the best)**
  - **Everything is in `use strict` now**
  - **Certain extremely weird code is no longer legal**
  - **Extreme edge cases**
    - **`Number` accepts octal literals now**
    - **Missing time zone interpretation changed**
    - **`start` and `end` symbols added to unicode strings**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **`eval()` gone (for the best)**
  - **Everything is in `use strict` now**
  - **Certain extremely weird code is no longer legal**
  - **Extreme edge cases**
    - **`Number` accepts octal literals now**
    - **Missing time zone interpretation changed**
    - **`start` and `end` symbols**
    - **Other stuff like that, which we don't care about**

In this talk

What is ES6

Why care

Examples!

How helped

Downsides
- Babel
- Build
- Thinking
- Vetting

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- **Old code needs vetting**
  - **Removed ES3 things (like `with`)**
  - **`eval()` gone (for the best)**
  - **Everything is in `use strict` now**
  - **Certain extremely weird code is no longer legal**
  - **Extreme edge cases**
  - **We got through one of our major branches in a couple hours' work**

In this talk

What is ES6

Why care

Examples!

How helped

Downsides
- Babel
- Build
- Thinking
- Vetting
- Angular 1

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- **Angular 1 modules**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Angular 1

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- **Angular 1 modules**
  - **Angular 1's module system is just string lookup**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Angular 1

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- **Angular 1 modules**
  - **Angular 1's module system is just string lookup**
    - **Angular 1 predates CommonJS / AMD / etc**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Angular 1

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- **Angular 1 modules**
  - **Angular 1's module system is just string lookup**
    - **Angular 1 predates CommonJS / AMD / etc**
    - **Angular 1 had to invent its own module system**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Angular 1

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- **Angular 1 modules**
  - **Angular 1's module system is just string lookup**
    - **Angular 1 predates CommonJS / AMD / etc**
    - **Angular 1 had to invent its own module system**
    - **It's frankly a mess**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Angular 1

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- **Angular 1 modules**
  - **Angular 1's module system is just string lookup**
  - **That string lookup frequently fails in transpiling**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Angular 1

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- **Angular 1 modules**
  - **Angular 1's module system is just string lookup**
  - **That string lookup frequently fails in transpiling**
    - **Transpiling introduces new scopes**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Angular 1

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- **Angular 1 modules**
  - **Angular 1's module system is just string lookup**
  - **That string lookup frequently fails in transpiling**
    - **Transpiling introduces new scopes**
    - **Angular's modules are global; scopes kill it**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Angular 1

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- **Angular 1 modules**
  - **Angular 1's module system is just string lookup**
  - **That string lookup frequently fails in transpiling**
    - **Transpiling introduces new scopes**
    - **Angular's modules are global; scopes kill it**
    - **Weaving this is do-able, but a hassle**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Angular 1

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- **Angular 1 modules**
  - **Angular 1's module system is just string lookup**
  - **That string lookup frequently fails in transpiling**
    - **Transpiling introduces new scopes**
    - **Angular's modules are global; scopes kill it**
    - **Weaving this is do-able, but a hassle**
    - **Angular 2 fixes this; for 1, you hack**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Angular 1

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- **Angular 1 modules**
  - **Angular 1's module system is just string lookup**
  - **That string lookup frequently fails in transpiling**
    - **Transpiling introduces new scopes**
    - **Angular's modules are global; scopes kill it**
    - **Weaving this is do-able, but a hassle**
    - **Angular 2 fixes this; for 1, you hack**
    - **It's a one-time problem in the build system**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Angular 1

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- **Angular 1 modules**
  - **Angular 1's module system is just string lookup**
  - **That string lookup frequently fails in transpiling**
  - **Sean already mostly fixed this in our codebase**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Style

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- Angular 1 modules
- **Code style transition**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Style

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- Angular 1 modules
- **Code style transition**
  - **Changing approaches you're used to is hard**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Style

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- Angular 1 modules
- **Code style transition**
  - **Changing approaches you're used to is hard**
  - **Loops to `.map`, `.filter`, `.reduce`, etc**

In this talk

What is ES6

Why care

Examples!

How helped

Downsides
- Babel
- Build
- Thinking
- Vetting
- Style

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- Angular 1 modules
- **Code style transition**
  - **Changing approaches you're used to is hard**
  - **Loops to `.map`, `.filter`, `.reduce`, etc**
  - **Actual classes vs what we used to call classes**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Style

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- Angular 1 modules
- **Code style transition**
  - **Changing approaches you're used to is hard**
  - **Loops to `.map`, `.filter`, `.reduce`, etc**
  - **Actual classes vs what we used to call classes**
  - **`let` scope vs `var` scope**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Style

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- Angular 1 modules
- **Code style transition**
  - **Changing approaches you're used to is hard**
  - **Loops to `.map`, `.filter`, `.reduce`, etc**
  - **Actual classes vs what we used to call classes**
  - **`let` scope vs `var` scope**
  - **Getting used to `const`**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Style

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- Angular 1 modules
- **Code style transition**
  - **Changing approaches you're used to is hard**
  - **Loops to `.map`, `.filter`, `.reduce`, etc**
  - **Actual classes vs what we used to call classes**
  - **`let` scope vs `var` scope**
  - **Getting used to `const`**
  - **Destructuring assignment**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Style

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- Angular 1 modules
- **Code style transition**
  - **Changing approaches you're used to is hard**
  - **Loops to `.map`, `.filter`, `.reduce`, etc**
  - **Actual classes vs what we used to call classes**
  - **`let` scope vs `var` scope**
  - **Getting used to `const`**
  - **Destructuring assignment**
  - **Things you trust vs things that are new**

In this talk
What is ES6
Why care
Examples!
How helped
Downsides
Babel
Build
Thinking
Vetting
Style

What are some downsides?

- Babel - heavy; slow shims; slow builds; plugins; testing
- The build organization - separate; transform; exclude
- New thinking - learn, adapt
- Old code needs vetting - `eval`, `use strict`, once-over
- Angular 1 modules
- **Code style transition**
  - **Changing approaches you're used to is hard**
  - **Loops to `.map`, `.filter`, `.reduce`, etc**
  - **Actual classes vs what we used to call classes**
  - **`let` scope vs `var` scope**
  - **Getting used to `const`**
  - **Destructuring assignment**
  - **Things you trust vs things that are new**
  - **Learning to hate ES5 is a lot of work**

In summary

Hooray, he's almost in shut-up mode