Modules

tl;dr

- Polluting the global namespace has always been a problem in JavaScript
- Legacy solutions required terribly ugly patterns like IIFEs.
- RequireJS/AMD is the solution used in Node
- ES2015 imports are used on the browser

```
<head>
<script src="s1.js"></script>
<script src="s2.js"></script>
<script src="s3.js"></script>
</head>
<body>
...
<script src="s4.js"></script>
</body>
```

In the browser all scripts are loaded into the same memory space

Say you have this code ...

```
var c = new Person("James", "Gordon", "Commissioner");
var runTime = new Date();
function showInfo(person) {
  return `${person.alias} created at ${runTime}`;
}
alert(showInfo(c));
```

But we are using a library that does this

```
var runTime = programEnd - programStart;
```

- What happens to runTime?

We've polluted the global namespace

Immediately Invoked Function Expression

Quiz: What do these do?

`var x = foo;`

- Sets x equal to foo – even if foo is a function

`foo();`

- Runs the foo function
- So if you take a function and put parens after it, you're telling JavaScript to run that function

Quiz: What is this?

```
function () {
   // Do stuff here
}
```

- An anonymous function, of course
- How about this?

```
(function () {
   // Do stuff here
})
```

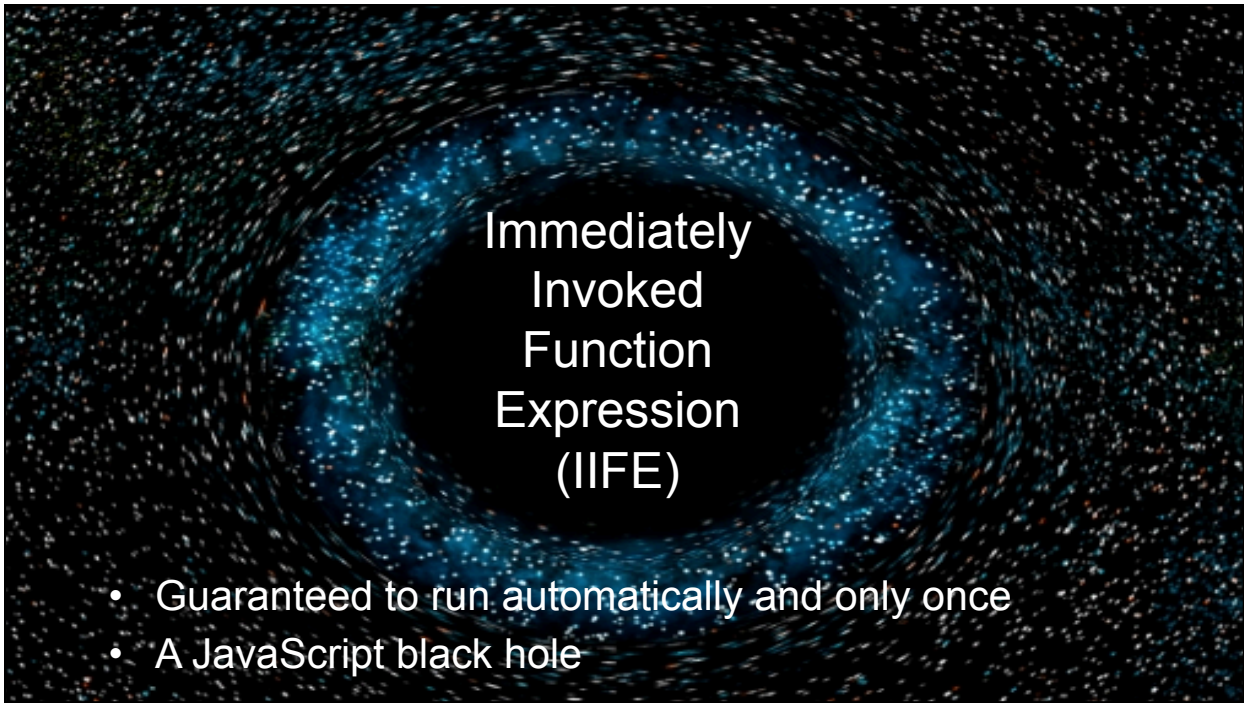
- Same thing

The payoff! The iife

- If you combine the anonymous function with parentheses, you have your iife:

```
(function () {
   // do stuff here
})()
```

- This says to define this function and then run it.



You can encapsulate some parts and expose others

```
(function () {
  Car = function (make) {
    this.make=make;
    this.go = function () {
      // Do stuff to make it 'go'
    }
  };
})();
let c = new Car('chevy');
c.go();
let p = new Car('porsche');
```

- A library called CommonJS solved the problem by creating the idea of modules.
- Each JS file would be encapsulated and then loaded by the library.
- Problem was you needed the library to load everything else.

- Along comes node.
- Ryan Dahl thinks, "We're starting from scratch here. I have the opportunity to make this better"
- He used a form of modules that split from CommonJS. It was called AMD.
- A library called RequireJS supports AMD.

RequireJS exporting

- To export something you put it on an object called "exports".

```
function foo() { ... }
exports.foo = foo;
exports.bar = function () {
  // do stuff here
};
```

- Note: in Node, the exports object is part of the module object, so it is actually *module*.exports.

RequireJS importing

- To import something you *require* it.

```
const allExportedThings = require('./path');
const oneThing = require('./path').foo;
```

- If this format had been adopted natively in the browser, we'd have solved a lot of problems!
- We only have to learn one thing
- Same syntax in node and in the browsers
- But TC39 settled on a different syntax for ES2015

Two ways to export

- A module must export itself before another module can import it.
- Default export

```
function foo() { ... }
function bar() { ... }
export default foo;
```

  - Only one thing can be the default export
- Named export

```
export function foo() { ... }
export function bar() { ... }
```

  - You can have any number of named exports

To import

- Default import:

```
import foo from './other.js';
import bar from './other.js';
```

- Named import:

```
import { foo } from './foo.js';
import { bar } from './bar.js';
```

Note: the "js" is optional. Best practice is to leave it off.

Example

Car.js

```
export class Car
{
  ...
};
```

Main.js

```
import {Car} from './Car.js';
const c = new Car();
```

Your code will always be imported as a relative path. Libraries will always be the name of the library

Your JavaScript code	A JavaScript library
`import foo from './bar';` • Always starts with "." or ".." • Relative to the current file	`import 'React' from 'react';` • Looks for it under node_modules

tl;dr

- Polluting the global namespace has always been a problem in JavaScript
- Legacy solutions required terribly ugly patterns like IIFEs.
- RequireJS/AMD is the solution used in Node
- ES2015 imports are used on the browser