FRONT-END MODULES

What? More?

ES6 MODULES

ES6 MODULES

- When Node.js was introduced (2009), it adopted a module system using require and module.exports, like we know now (influenced by CommonJS spec)

- ECMAScript 6 (2015) approved a native module system for JavaScript using a different set of keywords:
  - import <thing> from '<file>'
  - export <thing>
  - export default <thing>

RATIONALE

- The Node.js style modules:
  - only support synchronous module loading
  - are dynamic

- Import/export were designed:
  - to support both sync and async module loading
  - are not dynamic, which allows static analysis (i.e. tools can examine your code without running it)

DEFAULT EXPORT

a.js

```
const foo = () => {/* etc */}

module.exports = foo
```

b.js

```
const foo = require('./a')
```

DEFAULT EXPORT

a.js	b.js

`const foo = () => {/* etc */}`
`module.exports = foo`
	`const foo = require('./a')`

`const foo = () => {/* etc */}`
`export default foo`
	`import foo from './a'`

DEFAULT EXPORT

a.js

```
const foo = () => {/* etc */}

module.exports = foo
```

b.js

```
const foo = require('./a')
```

```
const foo = () => {/* etc */}

export default foo
```

```
import foo from './a'
```

There can only be one default export!

NAMED EXPORTS

a.js

```
const foo = () => {/* etc */}
const bar = 'bar'

module.exports = {
  foo: foo,
  bar: bar
}
```

b.js

```
const {foo, bar} = require('./a')
```

NAMED EXPORTS

a.js

```
const foo = () => {/* etc */}
const bar = 'bar'

module.exports = {
  foo: foo,
  bar: bar
}
```

b.js

```
const {foo, bar} = require('./a')
```

```
export const foo = () => {/* etc */}

export const bar = 'bar'
```

```
import {foo, bar} from './a'
```

NAMED EXPORTS

a.js

```
const foo = () => {/* etc */}
const bar = 'bar'

module.exports = {
  foo: foo,
  bar: bar
}
```

b.js

```
const {foo, bar} = require('./a')
```

```
export const foo = () => {/* etc */}

export const bar = 'bar'
```

```
import {foo, bar} from './a'
```

There can be *multiple* named exports!

COMBINING NAMED & DEFAULT EXPORTS

a.js

```
const foo = () => {/* etc */}
const bar = ‘bar'
const baz = 42

export foo
export bar
export default baz
```

b.js

```
import {foo, bar}, baz from './a'
```

USING MODULES ON THE FRONT-END

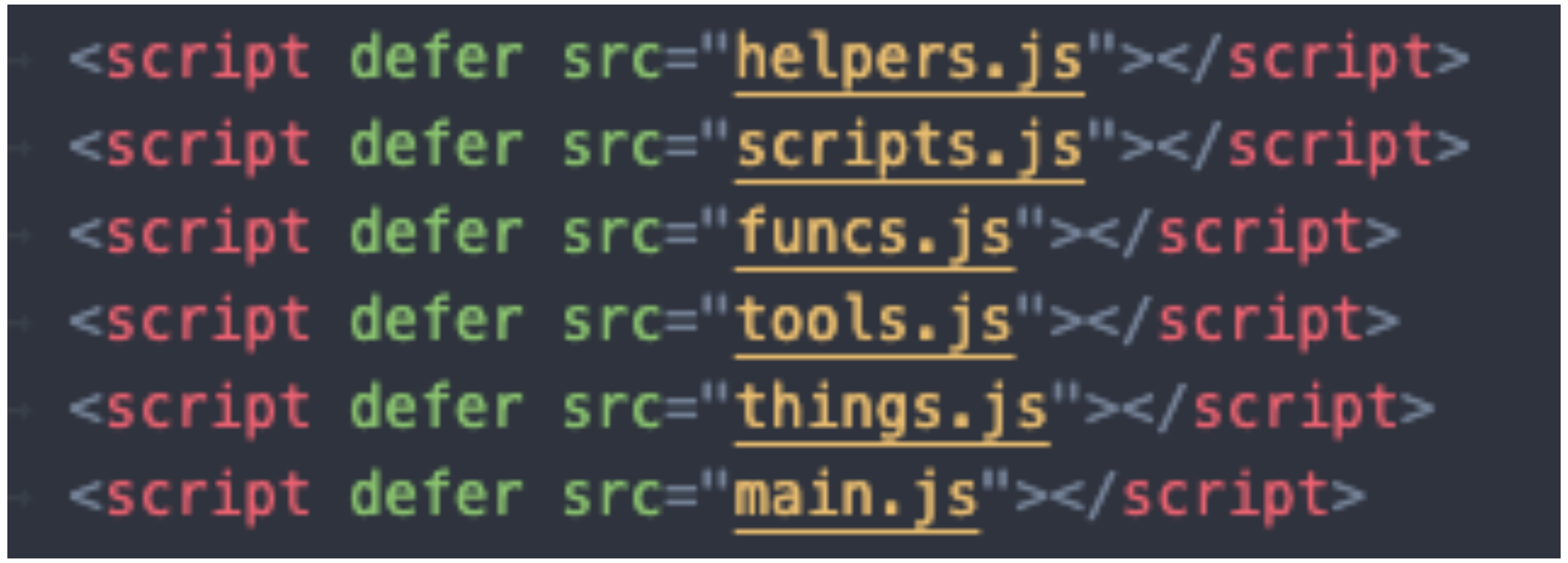
WHERE CAN I USE IMPORT/EXPORT?

- Import/Export has support in the latest versions of some browsers
  - Not quite safe yet to depend on it without a build tool like Webpack

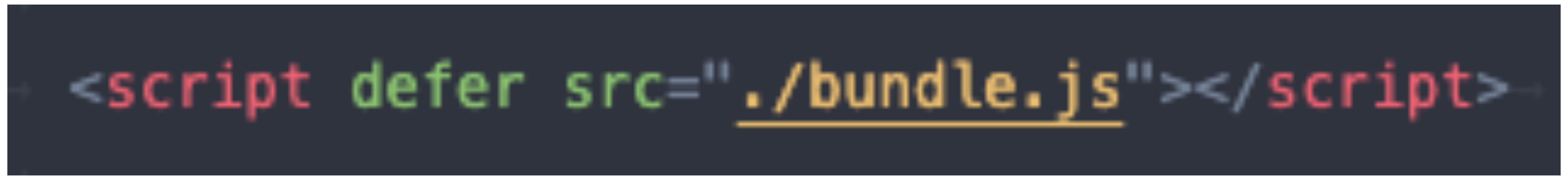
- Node.js does not support import/export natively yet**
  - **It is still “experimental” in Node 14

WEBPACK

- Webpack is a “JavaScript Module Bundler”
  - Takes in modules with dependencies, and generates static assets representing those modules
  - Compiles your code into something the browser understands



```html
<script defer src="helpers.js"></script>
<script defer src="scripts.js"></script>
<script defer src="funcs.js"></script>
<script defer src="tools.js"></script>
<script defer src="things.js"></script>
<script defer src="main.js"></script>
```



```html
<script defer src="./bundle.js"></script>
```

WEBPACK CONFIG

- Webpack will take a special config file where you can specify:
  - Entry: what’s the “source” module file? The file that imports all the others? The “starting point” of your code?
  - Output: after Webpack bundles it all up, where should the result go?
  - …among other things

WHAT SHOULD I USE?

- Browser-side JavaScript: use import/export and use webpack to compile your code
- Node: continue to use require and module.exports
- ^ This is the convention we will be using at Fullstack too :)