ES6 aims to create a syntax using the best from both worlds, without caring about the actual implementation. The Ecma TC39 committee (which is responsible for evolving ES6 and authoring the specification of the language) wanted to have a nice and easy syntax (that's arguably CommonJS's strong suit), but to support asynchronous loading (like AMD), and a few goodies like the possibility to statically analyze the code by tools and support cyclic dependencies nicely. The new syntax handles how you export and import things to and from modules.

This module thing is really important in Angular, as pretty much everything is defined in modules, that you have to import when you want to use them. Let's say I want to expose a function to bet on a specific pony in a race and a function to start the race.

In races.service.js:

```
export function bet(race, pony) {
  // ...
}
export function start(race) {
  // ...
}
```

As you can see, this is fairly easy: the new keyword `export` does a straightforward job and exports the two functions.

Now, let's say one of our application components needs to call these functions.

In another file:

```
import { bet, start } from './races.service';
```

```
// later
bet(race, pony1);
start(race);
```

That's what is called a *named export*. Here we are importing the two functions, and we have to specify the filename containing these functions - here 'races.service'. Of course, you can import only one method if you need, you can even give it an alias:

```
import { start as startRace } from './races.service';
```

```
// later
startRace(race);
```

And if you need to import all the methods from the module, you can use a wildcard '*'.

As you would do with other languages, use the wildcard with care, only when you really want all