the functions, or most of them. As this will be analyzed by our IDEs, we will see auto-import soon and that will free us from the bother to import the right things.

With a wildcard, you have to use an alias, and I kind of like it, because it makes the rest of the code clearer:

```
import * as racesService from './races.service';
```

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

If your module exposes only one function or value or class, you don't have to use named export, and you can leverage the default keyword. It works great for classes for example:

```
// pony.js
export default class Pony {
}
// races.service.js
import Pony from './pony';
```

Notice the lack of curly braces to import a default. You can import it with the alias you want, but to be consistent, it's better to call the import with the module name (except if you have multiple modules with the same name of course, then you can chose an alias that allows to distinguish them). And of course, you can mix default export with named ones, but obviously with only one default per module.

In Angular, you're going to use a lot of these imports in your app. Each component and service will be a class, generally isolated in their own file and exported, and then imported when needed in other components.

3.14. Conclusion

That ends our gentle introduction to ES6. We skipped some other parts, but if you're comfortable with this chapter, you will have no problem writing your apps in ES6. If you want to have a deeper understanding of this, I highly recommend Exploring JS by Axel Rauschmayer or Understanding ES6 from Nicholas C. Zakas ... Both ebooks can be read online for free, but don't forget to buy it to support their authors, they have done a great work! Actually I've re-read Speaking JS, Axel's previous book, and I again learned a few things, so if you want to refresh your JS skills, I definitely recommend it!