We also have a class Set:

```
const cedric = { id: 1, name: 'Cedric' };
const users = new Set();
users.add(cedric); // adds a user
console.log(users.has(cedric)); // true
console.log(users.size); // 1
users.delete(cedric); // removes the user
```

You can iterate over a collection, with the new syntax for … of:

```
for (let user of users) {
  console.log(user.name);
}
```

You'll see that the for … of syntax is the one the Angular team chose to iterate over a collection in a template.

3.12. Template literals

Composing strings has always been painful in JavaScript, as we usually have to use concatenation:

```
const fullname = 'Miss ' + firstname + ' ' + lastname;
```

Template literals are a new small feature, where you have to use backticks (`) instead of quotes or simple quotes, and you have a basic templating system, with multiline support:

```
const fullname = `Miss ${firstname} ${lastname}`;
```

The multiline support is especially great when your are writing HTML strings, as we will do for our Angular components:

```
const template = `<div>
  <h1>Hello</h1>
</div>`;
```

3.13. Modules

A standard way to organize functions in namespaces and to dynamically load code in JS has always been lacking. NodeJS has been one of the leaders in this, with a thriving ecosystem of modules using the CommonJS convention. On the browser side, there is also the AMD (Asynchronous Module Definition) API, used by RequireJS. But none of these were a real standard, thus leading to endless discussions on what's best.