As you can see, the keyword `super` allows calling the method of the base class, with `super.speed()` for example.

The `super` keyword can also be used in constructors, to call the base class constructor:

```
class Animal {
  constructor(speed) {
    this.speed = speed;
  }
}
class Pony extends Animal {
  constructor(speed, color) {
    super(speed);
    this.color = color;
  }
}
const pony = new Pony(20, 'blue');
console.log(pony.speed); // 20
```

3.9. Promises

Promises are not so new, and you might know them or use them already, as they were a big part of AngularJS 1.x. But since you will use them a lot in Angular, and even if you're just using JS, I think it's important to make a stop.

Promises aim to simplify asynchronous programming. Our JS code is full of async stuff, like AJAX requests, and usually we use callbacks to handle the result and the error. But it can get messy, with callbacks inside callbacks, and it makes the code hard to read and to maintain. Promises are much nicer than callbacks, as they flatten the code, and thus make it easier to understand. Let's consider a simple use case, where we need to fetch a user, then their rights, then update a menu when we have these.

With callbacks:

```
getUser(login, function (user) {
  getRights(user, function (rights) {
    updateMenu(rights);
  });
});
```

Now, let's compare it with promises: