we'll talk about two new JavaScript keywords: async and await. Keep in mind that these keywords are not supported in every JavaScript environment. When you want to know if a JavaScript feature is supported by a browser, use [http://caniuse.com](http://caniuse.com/#feat=async-functions). As you can see, the async and await keywords are supported by a majority of up-to-date browsers. But of course, Internet Explorer does not support them. Since async and await are keywords, we can't really use a polyfill to fix that. But if you want to target an old browser and you want to code with the async and await keywords, you should use the [Babel compiler](https://babeljs.io/).

function job() {

return new Promise(function(resolve, reject) {

setTimeout(resolve, 500, 'Hello world 1');

});

}

async function test() {

let message = await job();

console.log(message);

return 'Hello world 2';

}

test().then(function(message) {

console.log(message);

});

Let's examine what is going on here:

The async keyword at line 7 means that the function test will always return a promise. Even if the function just executes a simple return 'Hello world'. It will be converted to a promise no matter what. That's why we can use a then at line 14 to print the message.

The await keyword at line 8 means that the code execution will stop until the promise returned by job() is resolved. And the return value from test() is stored in the message variable.

And that's all. It's very simple to understand async and await when you already know how promises work. If you have a hard time understanding this, we can rewrite the test function in plain promise style:

function test() {

return job().then(function(message) {

console.log(message);

return 'Hello world 2';

});

}