ES2017: Async and Await

Async and await are going to be an important part of your coding knowledge, so let's learn how to use them.

WE'LL COVER THE FOLLOWING

ES2017 introduced a new way of working with promises, called async and await.

`Promise` review

Before we dive into this new strategy, let's quickly review how we would usually write a promise:

```
// fetch a user from github
fetch('api.github.com/user/AlbertoMontalesi').then( res => {
  // return the data in json format
  return res.json();
}).then(res => {
  // if everything went well, print the data
  console.log(res);
}).catch( err => {
  // or print the error
  console.log(err);
})
```

This is a very simple promise to fetch a user from GitHub and print it to the console.

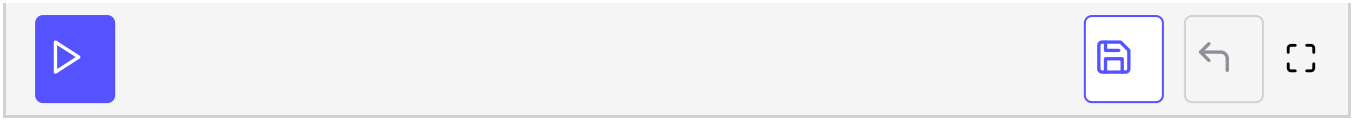
Let's see a different example:

```
function walk(amount) {
  return new Promise((resolve,reject) => {
    if (amount < 500) {

      reject ("the value is too small");
    }
    setTimeout(() => resolve(`you walked for ${amount}ms`),amount);
  });
}

walk(1000).then(res => {
  console.log(res);
  return walk(500);
}).then(res => {
  console.log(res);
  return walk(700);
}).then(res => {
  console.log(res);
  return walk(800);
}).then(res => {
  console.log(res);
  return walk(100);
}).then(res => {
  console.log(res);
  return walk(400);
}).then(res => {
  console.log(res);
  return walk(600);
});

// you walked for 1000ms
// you walked for 500ms
// you walked for 700ms
// you walked for 800ms
// uncaught exception: the value is too small
```



Let's see how we can rewrite this `Promise` with the new async/await syntax.

Async and await

```
function walk(amount) {
  return new Promise((resolve,reject) => {
    if (amount < 500) {
      reject ("the value is too small");
    }
    setTimeout(() => resolve(`you walked for ${amount}ms`),amount);
  });
}

// create an async function
async function go() {
  // use the keyword `await` to wait for the response
  const res = await walk(500);
```

```
  const res = await walk(500);
  console.log(res);
  const res2 = await walk(900);
  console.log(res2);
  const res3 = await walk(600);
  console.log(res3);
  const res4 = await walk(700);
  console.log(res4);
  const res5 = await walk(400);
  console.log(res5);
  console.log("finished");
}

go();

// you walked for 500ms
// you walked for 900ms
// you walked for 600ms
// you walked for 700ms
// uncaught exception: the value is too small
```

Running the code on the widget above will show all the console statements at the same time because of how this platform works. Try running it in the developer tools of your browser to see them logged one at the time.

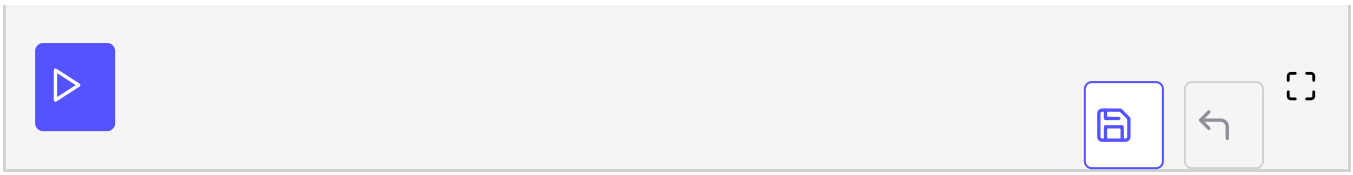
Let's break down what we just did:

- to create an `async` function we need to put the `async` keyword in front of it
- the keyword will tell JavaScript to always return a promise
- if we specify to `return <non-promise>` it will return a value wrapped inside a promise
- the `await` keyword **only** works inside an `async` function
- as the name implies, `await` will tell JavaScript to wait until the promise returns its result

Let's see what happens if we try to use `await` outside an `async` function

```
// use await inside a normal function
function func() {
  let promise = Promise.resolve(1);
  let result = await promise;
}
func();
// SyntaxError: await is only valid in async functions and async generators


// use await in the top-level code
let response = Promise.resolve("hi");
let result = await response;
// SyntaxError: await is only valid in async functions and async generators
```



> **Remember**: You can only use `await` inside an `async` function.

Error handling

In a normal promise we would use `.catch()` to catch eventual errors returned by the promise. Here, it is not much different:

```
async function asyncFunc() {

  try {
    let response = await fetch('http:your-url');
    } catch(err) {
        console.log(err);
      }
}

asyncFunc();
// TypeError: failed to fetch
```

We use `try...catch` to grab the error, but in a case where we do not have them we can still catch the error like this:

```
async function asyncFunc(){
  let response = await fetch('http:your-url');
}
asyncFunc();
// Uncaught (in promise) TypeError: Failed to fetch

asyncFunc().catch(console.log);
// TypeError: Failed to fetch
```

Let's test these concepts with another quiz.