# Async and await

## async keyword

`async` keyword is placed before a function like so:

```
async function f() {
return 1;
}
```

It basically make your function always returns a promise. If you didn't return a promise explicitly in the function they will be wrapped in a resolved promise automatically. The above async function is equivalent to the one below:

```
function f() {
return Promise.resolve(1);
}
```

Async function are `thenable` because they return a promise object:

```
async function f() {
return 1;
}

f().then(console.log); // prints out 1
```

> It just ensures that the returned value is a promise 100% and enables `await` in the function body, that is all. Function is still executed as a normal function!

## await keyword

You can only use `await` inside an `async` function, using it outside of an `async` function will result in a syntax error

The keyword `await` will make JavaScript wait until that promise settles and return the result of the resolved promise object as the value:

```
async function f() {
let promise = new Promise((res, rej) => {
        setTimeout(() => res("done"), 1000)
    });

    let result = await promise;
    console.log(result); // prints out done!
}

f();
```

`await` basically makes the function wait at the promise that you used it on and wait until the promise settles before moving on.

**It will wait for another promise to resolve before moving on with its own execution.**

The waiting doesn't cost any extra CPU time because since the function is marked asynchronous it can move onto executing other part of the code before resuming the execution after the promise you are awaiting is resolved.

> Using `await` is just a more elegant syntax of executing `.then`, you wait until the promise is resolved before moving on. It is easier to read and write.

> But you can only use `await` in a `async` function!

## Error handling

If the promise resolves normally, `await promise` returns the result. But in the case of rejection, it will throw the error:

```
async function f() {
    await Promise.reject(new Error("oops"));
}

// Is equivalent to
async function f(){
    throw new Error("oops");
}
```

You can handle that error using `try...catch`

```
async function f() {
  try {
    let response = await fetch('http://no-such-url');
  } catch(err) {
    console.log(err); // TypeError: failed to fetch
  }
}


f();
```

If you don't handle it using `try...catch` inside the body of the function, then the `async` function itself will become rejected, and you can handle it by adding `.catch` to handle it.

If you forget `.catch` then you will get an unhandled promise error.

### Why they are needed

If you are going to use `async/await` then you will rarely need to write `promise.then/catch` explicitly. `async/await` are based on promises.

Keep in mind that `await` doesn't work outside of non-async functions.

## Example of call async from non-async

```
async function wait() {
  await new Promise(resolve => setTimeout(resolve, 1000));


  return 10;
}


function f() {
  // How do we call wait() and print out the result 10?
  // simple
  wait().then(val => console.log(val));
}
```

You would just have to treat `wait` as a promise object, remember `async` functions always return a promise object, and then just have to `.then` it to use the result after it is resolved.

Revision #3
Created 29 December 2022 21:08:48 by Tamarine
Updated 29 December 2022 23:04:32 by Tamarine