ASYNC AND AWAIT

Async:

An **async** function is a function declared with the **async** keyword, and the **await** keyword is permitted within it. The **async** and **await** keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains.

Syntax:

async function name(param0, param1, /* … ,*/ paramN) {

  statements

}

Return value:

A **Promise** which will be resolved with the value returned by the async function, or rejected with an exception thrown from, or uncaught within, the async function.

Example:

```
function resolveAfter2Seconds() {
  console.log("starting slow promise");
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("slow");
      console.log("slow promise is done");
    }, 2000);
```

```
  });
}

function resolveAfter1Second() {
  console.log("starting fast promise");
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("fast");
      console.log("fast promise is done");
    }, 1000);
  });
}

async function sequentialStart() {
  console.log("==SEQUENTIAL START==");

  // 1. Execution gets here almost instantly
  const slow = await resolveAfter2Seconds();
  console.log(slow); // 2. this runs 2 seconds after 1.

  const fast = await resolveAfter1Second();
```

```
  console.log(fast); // 3. this runs 3 seconds after 1.
}
```

Await:

The **await** operator is used to wait for a Promise and get its fulfillment value. It can only be used inside an **async** function or a JavaScript module.

Syntax:

await expression

Return value:

The fulfillment value of the promise, or the expression itself's value itself if it's not a **Promise**.

Example:

```
function resolveAfter2Seconds(x) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(x);
    }, 2000);
  });
}


async function f1() {
  const x = await resolveAfter2Seconds(10);
```

```
  console.log(x); // 10
}
f1();
```

Snapshot:

Description:

1) Use of **async** and **await** enables the use of ordinary **try** / **catch** blocks around asynchronous code.
2) **Async** functions can contain zero or more await expressions.

3) The **await** keyword is only valid inside **async** functions within regular JavaScript code. If you use it outside of an async function's body, you will get a **SyntaxError**.

   await can be used on its own with JavaScript modules.

4) The purpose of **async/await** is to simplify the syntax necessary to consume promise-based APIs. The behavior of **async/await** is similar to combining generators and promises.
5) **Async** functions always return a promise. If the return value of an **async** function is not explicitly a promise, it will be implicitly wrapped in a promise.
6) The word "**async**" before a function means one simple thing: a function always returns a promise. Other values are wrapped in a resolved promise automatically. So, **async** ensures that the function returns a promise, and wraps non-promises in it.

Promise:

The **Promise** object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

A **Promise** is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.

A **Promise** is in one of these states:

pending: initial state, neither fulfilled nor rejected.

fulfilled: meaning that the operation was completed successfully.

rejected: meaning that the operation failed.

Difference:

As you can see, both of the functions above have the same body in which we try to access a property of an argument that is undefined in both cases. The only difference between the two functions is that asyncFn is declared with the async keyword.

This means that Javascript will make sure that the asnycFn will return with a Promise (either resolved or rejected) even if an error occured in it, in our case calling our .catch() block.

However with the fn function the engine doesn't yet know that the function will return a Promise and thus it will not call our catch() block.

Example:

```
function fn(obj) {
  const someProp = obj.someProp
  return Promise.resolve(someProp)
}


async function asyncFn(obj) {
  const someProp = obj.someProp
  return Promise.resolve(someProp)
}


asyncFn().catch(err => console.error('Catched')) // => 'Catched'
fn().catch(err => console.error('Catched')) // => TypeError: Cannot read property 'someProp' of undefined
```