async function

The `async function` declaration defines an *asynchronous function*, which returns an `AsyncFunction` object.

You can also define async functions using an async function expression.

Syntax

```
async function name([param[, param[, ... param]]]) {
   statements
}
```

Parameters

`name`
The function name.

`param`
The name of an argument to be passed to the function.

`statements`
The statements comprising the body of the function.

Return value

An `AsyncFunction` object, representing an asynchronous function which executes the code contained within the function.

Description

When an `async` function is called, it returns a `Promise`. When the `async` function returns a value, the `Promise` will be resolved with the returned value. When the `async` function throws an exception or some value, the `Promise` will be rejected with the thrown value.

An `async` function can contain an `await` expression, that pauses the execution of the async function and waits for the passed `Promise`'s resolution, and then resumes the `async` function's execution and returns the resolved value.

> The purpose of `async`/`await` functions is to simplify the behavior of using promises synchronously and to perform some behavior on a group of `Promises`. Just as `Promises` are similar to structured callbacks, `async`/`await` is similar to combining generators and promises.

Examples

Simple example

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


async function add1(x) {
  const a = await resolveAfter2Seconds(20);
  const b = await resolveAfter2Seconds(30);
  return x + a + b;
}

add1(10).then(v => {
  console.log(v);  // prints 60 after 4 seconds.
});


async function add2(x) {
  const p_a = resolveAfter2Seconds(20);
  const p_b = resolveAfter2Seconds(30);
  return x + await p_a + await p_b;
}

add2(10).then(v => {
  console.log(v);  // prints 60 after 2 seconds.
});
```

Rewriting a promise chain with an `async` function

An API that returns a `Promise` will result in a promise chain, and it splits the function into many parts. Consider the following code:

```
function getProcessedData(url) {
  return downloadData(url) // returns a promise
    .catch(e => {
      return downloadFallbackData(url)  // returns a promise
        .then(v => {
          return processDataInWorker(v); // returns a promise
        });
    })
    .then(v => {
      return processDataInWorker(v); // returns a promise
    });
}
```

it can be rewritten with a single `async` function as follows:

```
async function getProcessedData(url) {
  let v;
  try {
    v = await downloadData(url);
  } catch(e) {
    v = await downloadFallbackData(url);
  }
  return processDataInWorker(v);
}
```

Note that in the above example, there is no `await` statement on the `return` statement, because the return value of an `async function` is implicitly wrapped in `Promise.resolve`.

Specifications

Specification	Status	Comment
ECMAScript Latest Draft (ECMA-262) The definition of 'async function' in that specification.	LS Living Standard	Initial definition in ES2017.

Specification	Status	Comment
ECMAScript 2017 (ECMA-262) The definition of 'async function' in that specification.	ST Standard	

Browser compatibility

Desktop Mobile

Feature	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari
Basic Support	55	(Yes)	52	No	42	10.1

See also

- `async function expression`
- `AsyncFunction` object
- `await`
- "Decorating Async Javascript Functions" on "innolitics.com"