# **Async/await**

## Async

We use the async keyword with a function to represent that the function is an asynchronous function. The async function returns a [promise](https://www.programiz.com/javascript/promise).

The syntax of async function is:

| async function name(parameter1, parameter2, ...paramaterN) {  *// statements* } |
| --- |

| *// async function example*  async function f() {  console.log('Async function.');  return Promise.resolve(1); }  f(); |
| --- |

In the above program, the async keyword is used before the function to represent that the function is asynchronous.

Since this function returns a promise, you can use the chaining method then() like this:

| async function f() {  console.log('Async function.');  return Promise.resolve(1); }  f().then(function(result) {  console.log(result) }); |
| --- |

## AWAIT

The await keyword is used inside the async function to wait for the asynchronous operation.

The syntax to use await is:

| let result = await promise; |
| --- |

| *// a promise* let promise = new Promise(function (resolve, reject) {  setTimeout(function () {  resolve('Promise resolved')}, 4000);  });  *// async function* async function asyncFunc() {   *// wait until the promise resolves*   let result = await promise;    console.log(result);  console.log('hello'); }  *// calling the async function* asyncFunc(); |
| --- |

Output

| Promise resolved hello |
| --- |

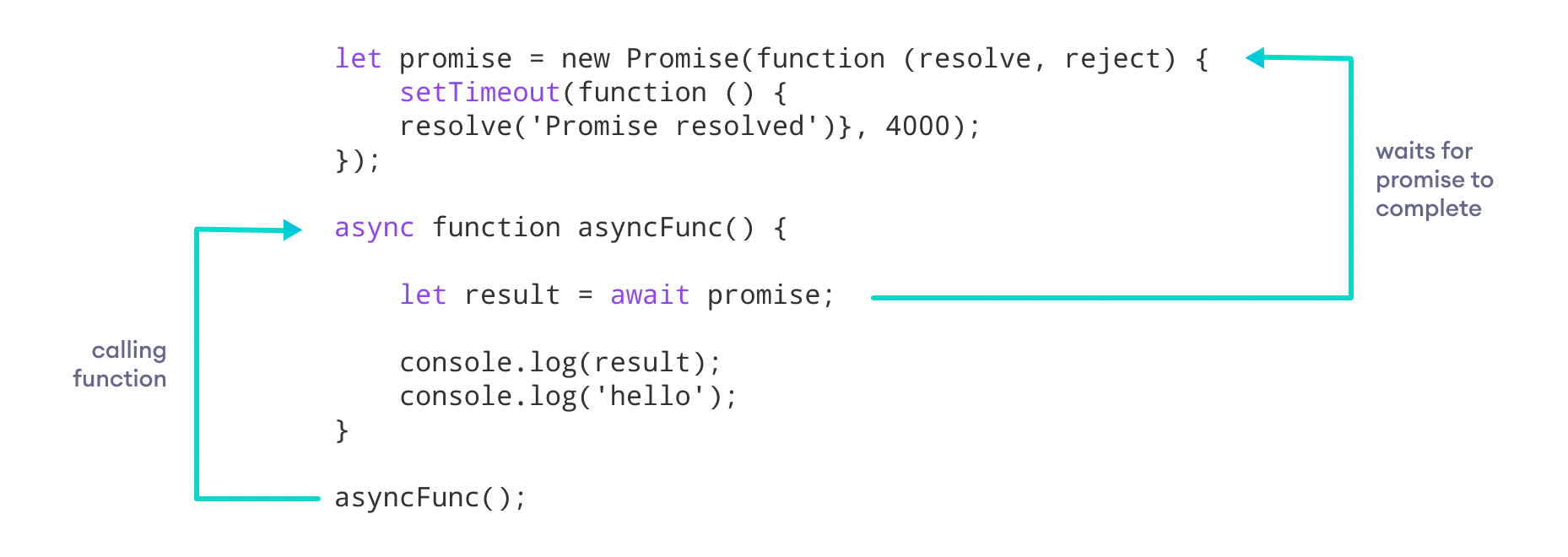
In the above program, a Promise object is created and it gets resolved after 4000 milliseconds. Here, the asyncFunc() function is written using the async function.

The await keyword waits for the promise to be complete (resolve or reject).

| let result = await promise; |
| --- |

Hence, hello is displayed only after promise value is available to the result variable.

In the above program, if await is not used, hello is displayed before Promise resolved.



The async function allows the asynchronous method to be executed in a seemingly synchronous way. Though the operation is asynchronous, it seems that the operation is executed in synchronous manner.

This can be useful if there are multiple promises in the program. For example,

| let promise1; let promise2; let promise3;  async function asyncFunc() {  let result1 = await promise1;  let result2 = await promise2;  let result3 = await promise3;   console.log(result1);  console.log(result1);  console.log(result1); } |
| --- |

In the above program, await waits for each promise to be complete.

## **Error Handling**

While using the async function, you write the code in a synchronous manner. And you can also use the catch() method to catch the error.

For example.

| asyncFunc().catch(  *// catch error and do something* ) |
| --- |

The other way you can handle an error is by using try/catch block. For example.

| *// a promise* let promise = new Promise(function (resolve, reject) {  setTimeout(function () {  resolve('Promise resolved')}, 4000);  });  *// async function* async function asyncFunc() {  try {  *// wait until the promise resolves*   let result = await promise;    console.log(result);  }   catch(error) {  console.log(error);  } }  *// calling the async function* asyncFunc(); *// Promise resolved* |
| --- |

In the above program, we have used try/catch block to handle the errors. If the program runs successfully, it will go to the try block. And if the program throws an error, it will go to the catch block.

To learn more about try/catch in detail, visit JavaScript [JavaScript try/catch](https://www.programiz.com/javascript/try-catch-finally).