ASYNCHRONOUS PROGRAMMING IN JAVASCRIPT

‹ Prev
Next ›

In this section, we will learn about synchronous and asynchronous programming in detail.

Synchronous programming is the programming in which the program executes in the order it is written and only one code block is executed at a time.

In another word, synchronous programs are **blocking programs** which block execution of code blocks below it until the current code block finishes its execution. There is no parallel execution in synchronous programming, code executes one-by-one.

```
let today = new Date();
console.log(today);
for (let i = 0; i < 2; i++) {
  console.log(i);
}
console.log("Hello World!");
```

Run Here ›

In the above example, you can see the :

- first, the date object is stored in the variable
- Then in the next line date is logged out
- Then the loop finished its execution
- And finally program logs "Hello World!".

Every code block of the program runs in the same order it is written, hence it is the synchronous program.

What is asynchronous programming?

Asynchronous programming is the programming that executes in the order it is written until some asynchronous operation arrives.

When an asynchronous operation arrives then some block of code split from the program's normal execution and starts executing parallel to other blocks.

This means that instead of executing code blocks one by one it would be executing more than one block of code at a time.

```
let i = 0;
setTimeout(() => {
  console.log("Time out!");
}, 2000);

setInterval(() => {
  console.log(i++);
}, 1000);

console.log("Hello World!");
```

Run Here ›

In the above example you can see:

- Program first output is "Hello World!" which is the last line of the code
- Then it outputs the value of 'i' at an interval of 1 second
- And finally, after 2 seconds of the program outputs "Time out!".

You must have observed program is not executed in the sequence it is defined, it is **non-blocking** hence it is an **asynchronous program**.

javascript is synchronous or asynchronous

We have seen both synchronous and asynchronous programs in JavaScript. **So what is javascript synchronous or asynchronous?**

In short, **JavaScript is synchronous**. It is a blocking and single-threaded programming language. Means `JavaScript can execute only one code block at a time`.

But this is part of the story.

Synchronous codes make it really difficult for developers to work on the things around, suppose you have to download 100 images and store them somewhere in the device then you would not want to sit around it while it downloads all those to jump on the next task.

So JavaScript community developed some ways by which you can manipulate JavaScript to behave in an asynchronous way.

There are mainly 2 types of asynchronous code style in JavaScript:

1. `Callback functions`
2. `JavaScript Promises`

---

1. Callback functions

`Callback functions` are the functions that are passed as an argument to another function.

These callback functions can be served both in a synchronous and asynchronous way. It depends on the type of task, if there is an asynchronous operation then callback can be used in an asynchronous way.

For example, you need to fetch a file from the server then in this case you can use the callback function in an asynchronous way to do something once the file is fetched.

You will learn about the callback function in detail in the next section.

```
function printAfter1Sec(callback) {
  setTimeout(callback, 1000);
}
function learn() {
  console.log("Learning Asynchronous JavaScript");
}
printAfter1Sec(learn);
```

Try It Run Here ›

2. JavaScript Promises

The Promises in JavaScript is a modern way to do asynchronous operations. It is an object which represents the completion or failure of an asynchronous operation.

Just like callback function promise is used to resolve asynchronous operation but in a clean and better way.

Suppose you have to get a file from the server and then print the text content on the screen, then a possible code for fetting file and printing it could be as following `getFile()` function:

```
function printFile(data) {
  document.write(data)
}
function someError(error) {
  console.error(error);
}
```

```
getFile(url, printFile, someError);
```

But this can be rewritten to return a promise and you would attach your callbacks to it instead.

```
getFile(url).then(printFile, someError);
```

Promises guarantees to run callback function after completion of the current run.

You will learn about JavaScript promises in detail in the coming section.

‹ Prev

Next ›

Practice Problems

Tools

HTML Editor

Advance HTML Editor

JavaScript Compiler

Follow Us