**Asynchronous:**

Functions running in **parallel** with other functions are called **asynchronous**

A good example is JavaScript setTimeout().

With simple function we can use asynchronous methods like setTimeOut but we cant be able to achieve the exact functionality. For this purpose we use callbacks to get desire results.

Alternate to call-backs is promises:---

**Promises object:** a promise is an **object** that encapsulates the result of an **asynchronous operation**.

a promise is an **object** that encapsulates the Producing Code and Consuming Code.

Generally, we resolve expected outputs, and reject errors, both of the handled and unexpected variety. Note that a Promise can handle synchronous computation as well, though it’s just less exciting.

A promise object has a state that can be one of the following:

* Pending
* Fulfilled with a **value**
* Rejected for a **reason**

Promise constructor accept call back knows as **executer** that takes two callbacks **resolve and reject.**

**Important:** Always return results, otherwise callbacks won't catch the result of a previous promise (with arrow functions, () => x is short for () => { return x; }). If the previous handler started a promise but did not return it, there's no way to track its settlement anymore, and the promise is said to be **"FLOATING"**.

**.then**

The .then() method takes up to two arguments; the first argument is a callback function for the fulfilled case of the promise, and the second argument is a callback function for the rejected case. Each .then() returns a newly generated promise object, which can optionally be used for chaining;

Unterminated promise chains lead to uncaught promise rejections in most browsers.

A good rule of thumb is to always either return or terminate promise chains, and as soon as you get a new promise, return it immediately, to flatten things:

[**https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using\_promises**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises)

**Where the value of the producer store for later use mean in .then?**

The Promise stores that value in closure so that when then is called later, it can go right ahead and invoke the callback function passed to then on the value.

If then is called before resolve/reject, then the value handler passed to then is deferred. Once resolve/reject is eventually called, meaning we have the value we were waiting for, that deferred handler from earlier is used.

[**https://medium.com/@ningxia/javascript-promises-under-the-hood-4ce853fcb2c9**](https://medium.com/@ningxia/javascript-promises-under-the-hood-4ce853fcb2c9)

**Detailed explanation with promise methods:**

[**https://www.programiz.com/javascript/promise**](https://www.programiz.com/javascript/promise)

**Quiz**

[**https://www.codingame.com/playgrounds/347/javascript-promises-mastering-the-asynchronous/its-quiz-time**](https://www.codingame.com/playgrounds/347/javascript-promises-mastering-the-asynchronous/its-quiz-time)

[**https://www.codeguage.com/courses/advanced-js/promises-chaining-quiz**](https://www.codeguage.com/courses/advanced-js/promises-chaining-quiz)