{ JavaScript }

Callbacks and Promises

In a simplified way

Let's Revisit

First, what is a function?

A function in JavaScript is a set of statements that performs a task.

This set of statements can exist without a function, but having them in a function helps us reuse the task in multiple places.

Callback

Callbacks

A callback function is **a function that is passed as an argument to another function, to be “called back” at a <u>later time</u>.**

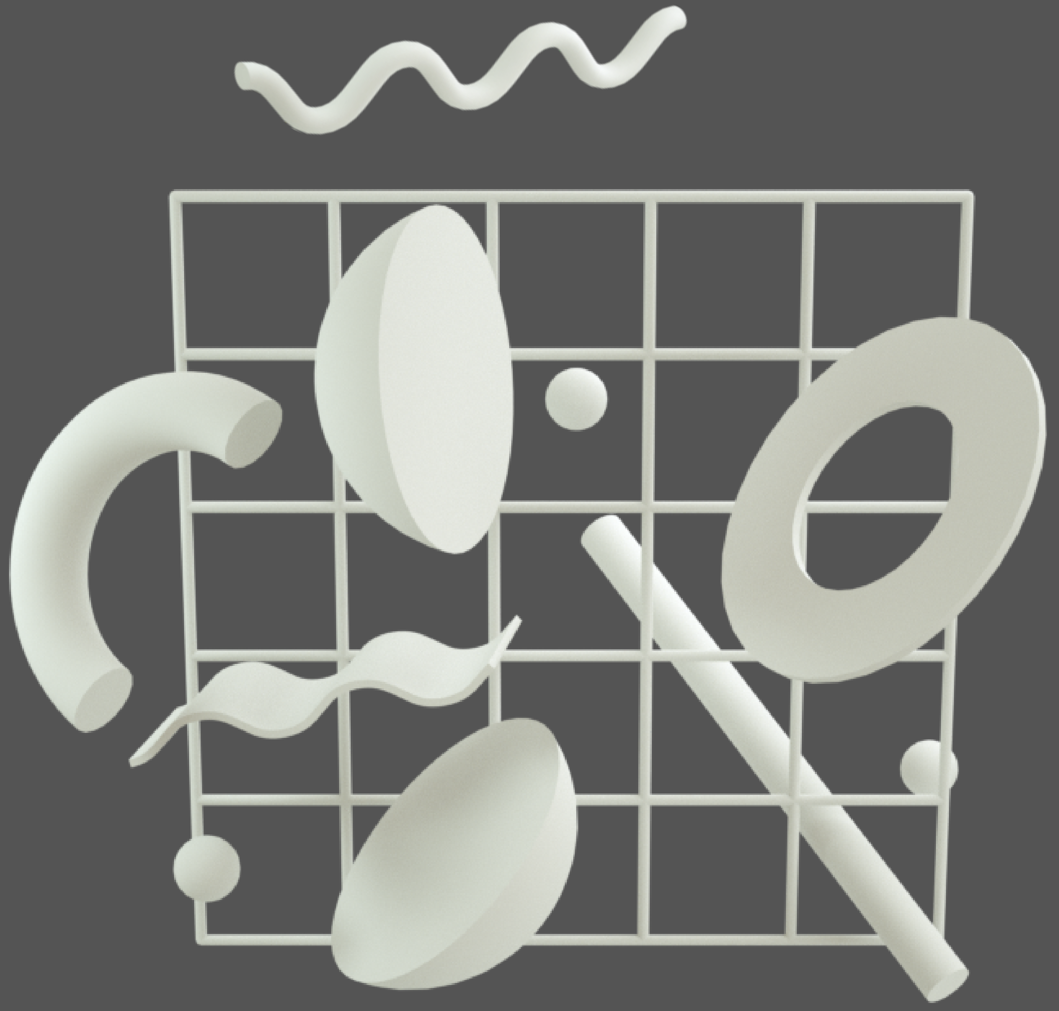
Why use Callbacks??

- Callbacks make sure that a function is not going to run before a task is completed but will run right after the task has completed

- It helps us develop asynchronous JavaScript code and keeps us safe from problems and errors.

Important Points:

- Callbacks are higher order functions so its provides the flexibility to pass another functions as an arguments.
- Nesting too many callback functions may not be a great idea and may create `Callback Hell`.
- Another drwaback is Inversion Of Control due to callbacks.



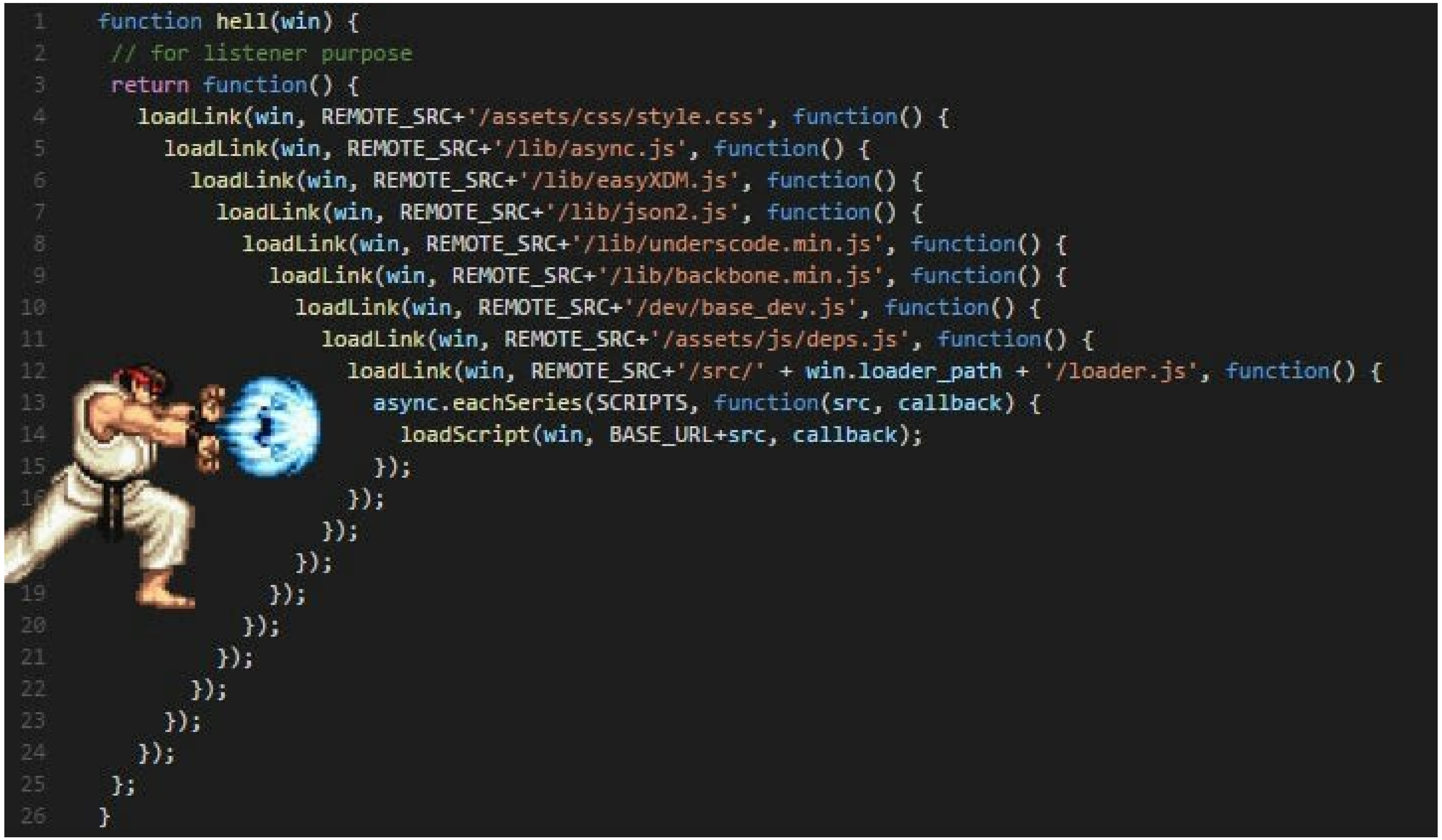
Callback Hell

also called Pyramid Of Doom

Callback hell is just every function call is having a callback function and this nesting of functions reduces code redability

For a big application it creates more nesting.

Callback hell !!



```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

Untitled - TextEdit

File Edit View Help

Let's look at **Inversion of Control** due to Callbacks

```
const cart =  ["shoes", "jeans" , "bag"]

api.createOrder( cart, function ( ) {
        api.proceedToPayment( );
 } );
```

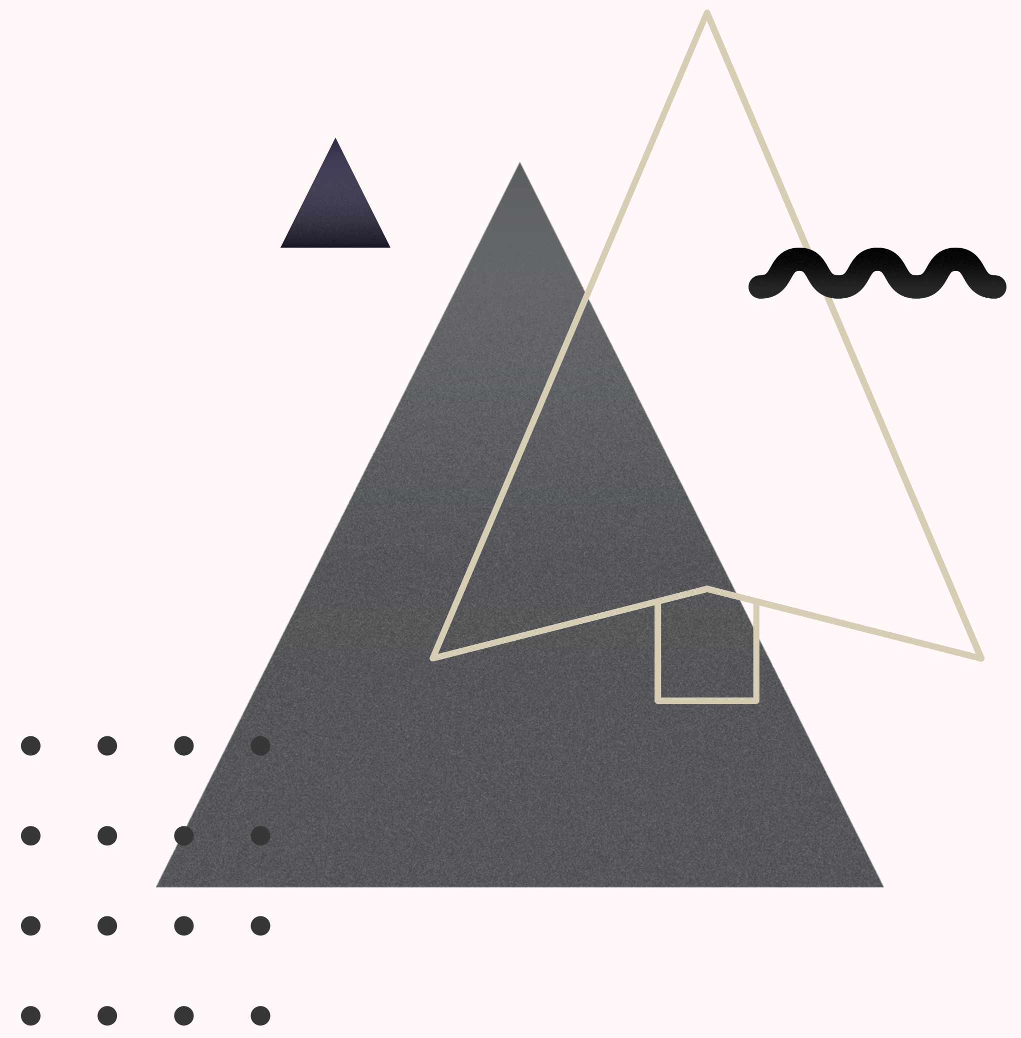
So suppose there is an API for taking orders.

First we need to create an order with the **createOrder( )** funtion
And just after our order is created the control will be shifted to **proceedToPayment( )** function so **to execute the functions in a serial manner we are using callback here**. But.......

JS

Promises

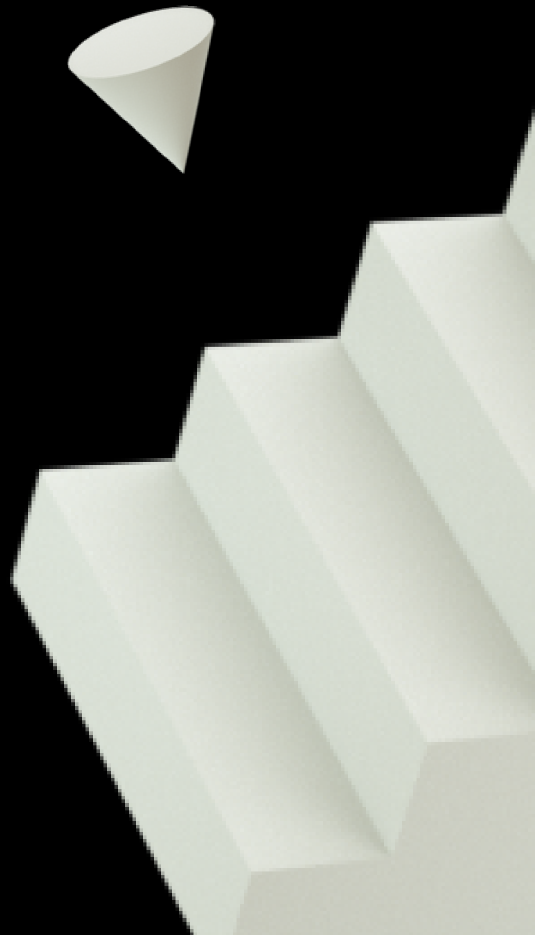
A Promise is an object representing the eventual completion or failure of an asynchronous operation.



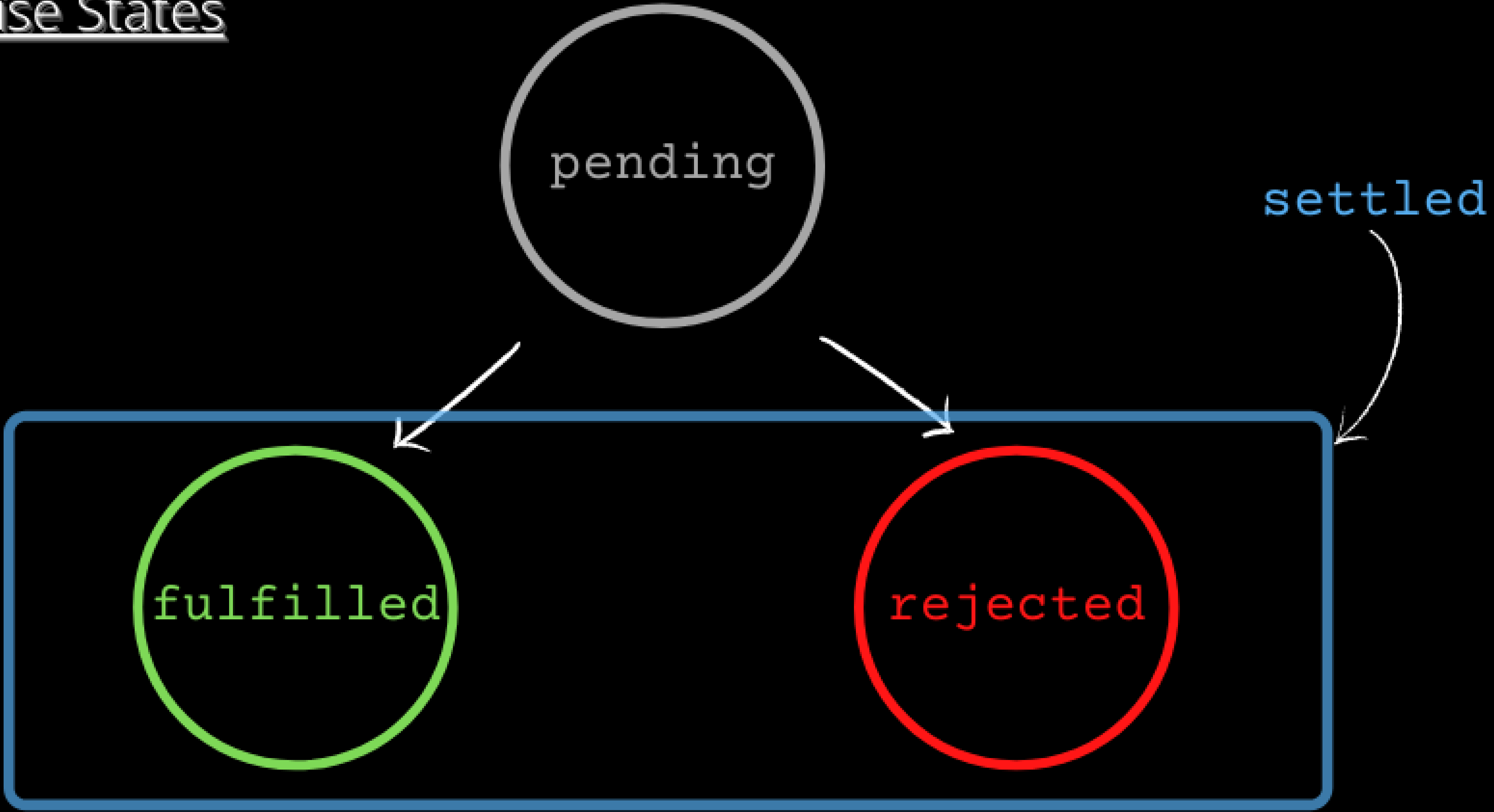
Promise States

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.



Promise States



CONNECT WITH ME

@mafizonly

@pacehutt

@mafiz._