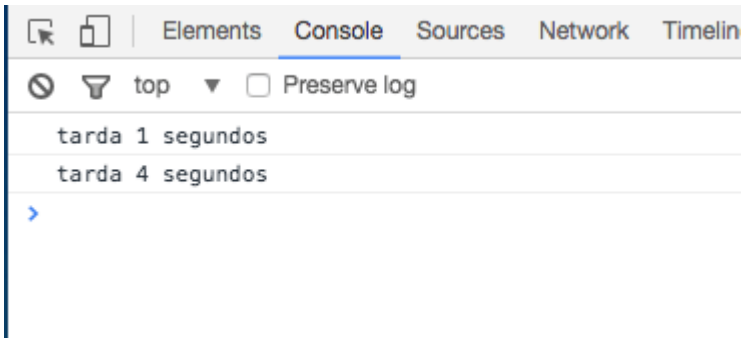
¿Cuál es el concepto de JavaScript Promise?. Las promesas son unos objetos que nos facilitan la gestión de la programación asíncrona dentro de JavaScript. Por ejemplo imaginemos que tenemos un programa de JavaScript que usa dos bloques de código que incluyen un setTimeout.

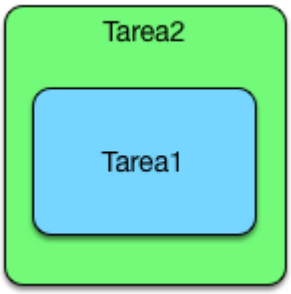
```
setTimeout(function() {
   console.log("tarda 1 segundos")
},1000);
setTimeout(function() {
   console.log("tarda 4 segundos")
},5000);
```

El resultado la veremos impreso por la consola , al pasar 1 segundo se imprimirá “tarda 1 segundo” y a los 4 segundos se imprimirá ” tarda 4 segundos”.



El problema le tenemos si queremos que el código que tarda 4 segundos se ejecute primero que el que tarda 1 segundo. En estos momentos los dos bloques se ejecutan de forma asíncrona e independiente.

Una solución es introducir dentro del código de los 4 segundos una llamada a una función que invoque el de 1 segundo , pero esto acoplaría dos funciones que en principio no están relacionadas, es una mala práctica.



La otra solución es usar JavaScript Promises y encadenar una promesa con otra.

Un ejemplo de JavaScript Promise

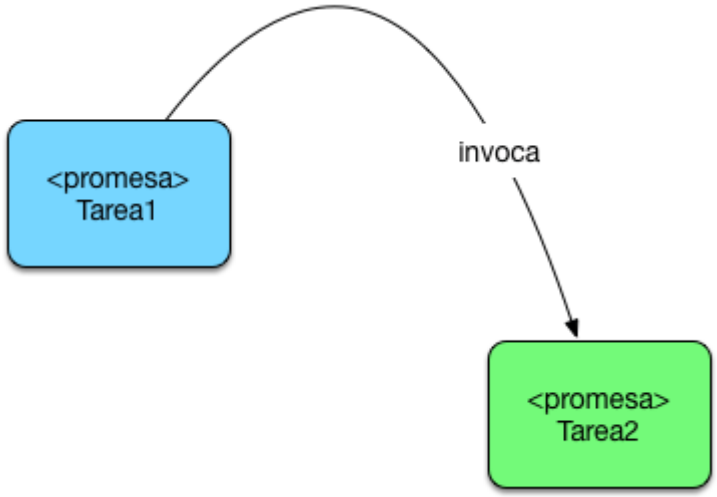
Para ello vamos usar JavaScript ES6 y construirnos dos objetos promesas , el primero tardará 4 segundos en ejecutarse y el segundo únicamente 1. Las promesas disponen del

método then() que permite encadenar unas con otras .

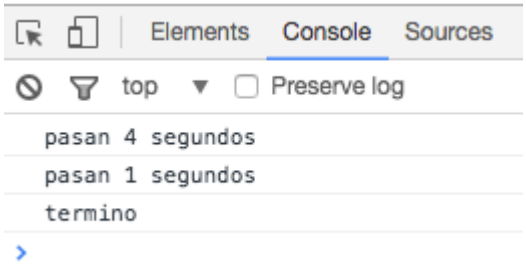
```
var promesa1 = function() {
            return new Promise(function(resolver, cancelar) {

                setTimeout(function() {
                    console.log("pasan 4 segundos");
                    resolver();

                }, 4000);
            });
        }


        var promesa2 = function() {
            return new Promise(function(resolver, cancelar) {

                setTimeout(function() {

                    console.log("pasan 1 segundos");
                    resolver();
                }, 1000);

            });
        }

        promesa1().then(promesa2).then(function() {

            console.log("termino");
```

```
});
```

En este caso la promesa dispone de dos parámetros resolver o cancelar . Cuando se termina el timeout invocamos a resolver para dar la promesa por finalizada. Esta a través de su método then invocará a la siguiente y así sucesivamente.



De esta forma conseguiremos que el código asíncrono se ejecute de la forma que deseamos y primero veamos pasar por pantalla el 4 , luego el 1 y la finalización.



Entender como funcionan las promesas es importante en JavaScript para poder generar un código flexible.

Otros artículos relacionados : JavaScript Reactive Programming , Introducción Rx.js