3. setTime\_out

**Set TimeOut**

Vamos a profundizar un poco con las funciones callbacks utilizadas para realizar tareas asíncronas. Probablemente, el caso más fácil de entender es utilizar un temporizador mediante la función setTimeout(callback, time).

Dicha función nos exige dos parámetros:

* La función callback a ejecutar
* El tiempo time que esperará antes de ejecutarla

El ejemplo sería el siguiente:

setTimeout(function() {

console.log("He ejecutado la función");

}, 2000);

Simplemente, le decimos a setTimeout() que ejecute la función callback que le hemos pasado por primer parámetro cuando transcurran 2000 milisegundos (es decir, 2 segundos). Utilizando arrow functions se puede simplificar el callback y hacer mucho más «fancy» y legible:

setTimeout(() => console.log("He ejecutado la función"), 2000);

Si lo prefieres y lo ves más claro (no suele ser habitual en código Javascript, pero cuando se empieza suele resultar más fácil entenderlo) podemos guardar el callback en una constante:

const action = () => console.log("He ejecutado la función");

setTimeout(action, 2000);

En cualquiera de los casos, lo importante es darse cuenta que estamos usando una función callback para pasársela a setTimeout(), que es otra función. En este caso, se trata de «programar» un suceso que ocurrirá en un momento conocido del futuro, pero muchas veces desconoceremos cuando se producirá (o incluso si se llegará a producir).

Si probamos el código que verás a continuación, comprobarás que el segundo console.log() se ejecutará antes que el primero, dentro del setTimeout(), mostrando primero Código síncrono y luego Código asíncrono en la consola del navegador:

setTimeout(() => console.log("Código asíncrono."), 2000);

console.log("Código síncrono.");

El último console.log del código se ejecuta primero (forma parte del flujo principal de ejecución del programa). El setTimeout() que figura en una línea anterior, aunque se ejecuta antes, pone en espera a la función callback, que se ejecutará cuando se cumpla una cierta condición (transcurran 2 segundos desde ese momento).

Esto puede llegar a sorprender a desarrolladores que llegan de otros lenguajes considerados bloqueantes; Javascript sin embargo se considera un lenguaje asíncrono y no bloqueante. ¿Qué significa esto? Al ejecutar la línea del setTimeout(), el programa no se queda bloqueado esperando a que terminen los 2 segundos y se ejecute la función callback, sino que continúa con el flujo general del programa para volver más adelante cuando sea necesario a ejecutar el callback, aprovechando así mejor el tiempo y realizando tareas de forma asíncrona.