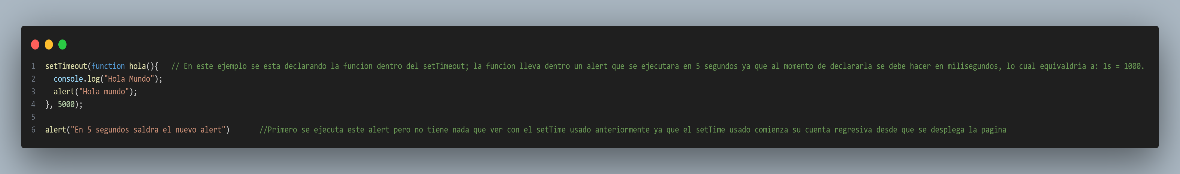
Investigación Programación Asíncrona:

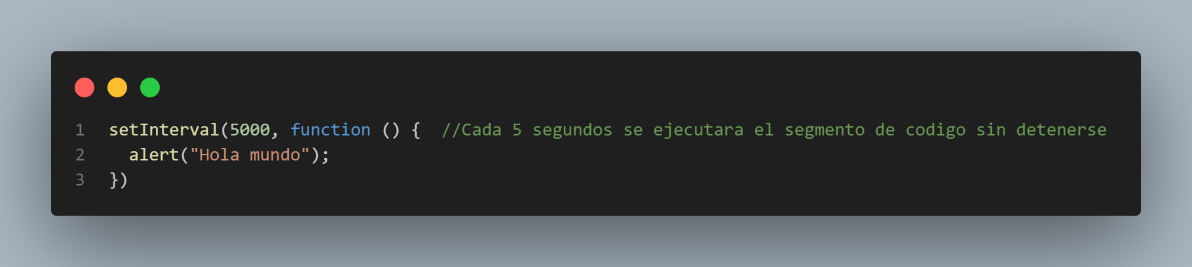
1. Que son los temporizadores setimeout y setinterval (de un ejemplo)

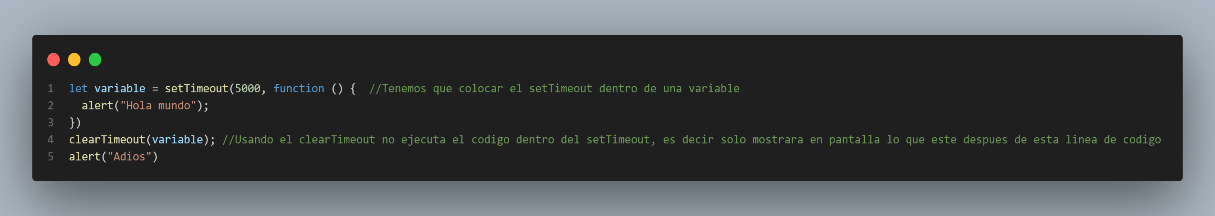
Temporizadores: (Timers) funciones que permiten realizar alguna tarea después de un determinado tiempo; timeout ejecuta cuando pasa cierto tiempo desde que se carga la página. Interval se ejecuta constantemente cada cierto tiempo.

**Nota: Los temporizadores permiten declarar el tiempo de demora antes de llamar la función o después de haberla llamado.**

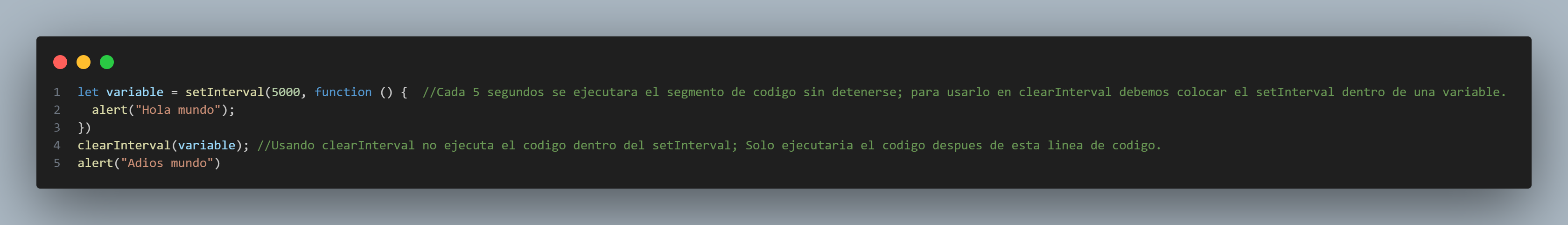
**Nota: 1 segundo equivale a 1000 en los temporizadores.**

**SetTimeout:** Permite ejecutar un fragmento de código luego de un cierto tiempo transcurrido.

**SetInterval:** Permite ejecutar un fragmento de código cada cierto tiempo.

**clearTimeout:** Permite que no se ejecute el código del setTimeout con una condición las cual es colocar el setTimeout dentro de una variable.

**clearInterval:** Permite que no se ejecute el código del setInterval con una condición las cual es colocar el setInterval dentro de una variable.



1. Que es la asincronía y el evento loop (de un ejemplo)

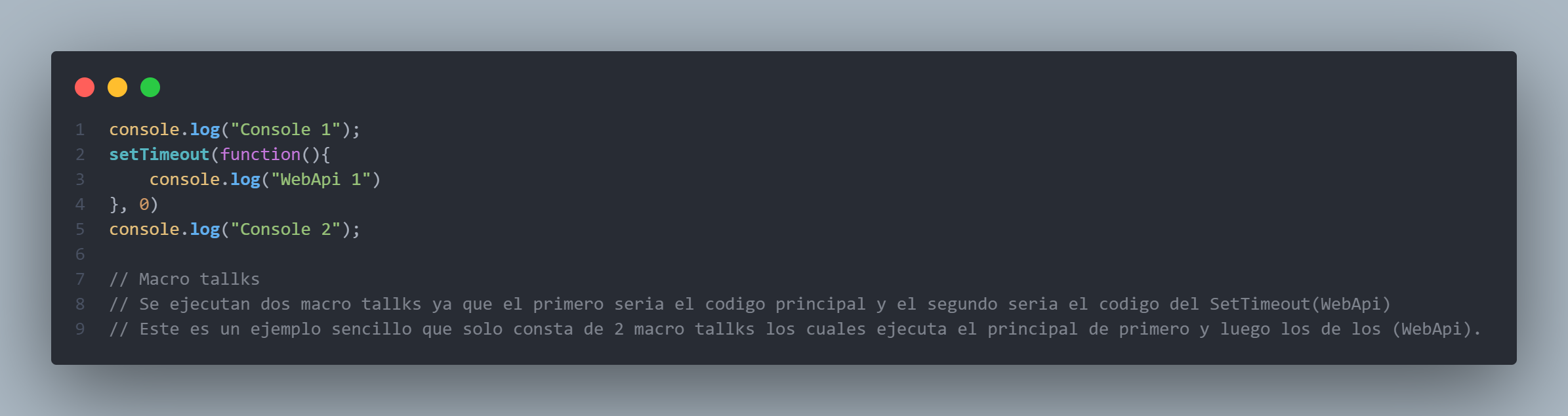
**Asincronía**: Permite ejecutar el código de una manera distinta a la secuencial, es decir podemos ejecutar funciones que codificadas son de manera secuencial, pero si es necesario se modifica y JavaScript lo ejecutara de manera distinta. Para entender esto mejor lo miraremos en el siguiente ejemplo:

****

**EVENT LOOP**: En términos sencillos, el event loop es como un vigilante que está continuamente atento a la aparición de nuevos eventos o tareas para ser procesadas. Estos eventos pueden ser acciones del usuario, respuestas de servidores o temporizadores, entre otros. En lugar de ejecutar una tarea tras otra de forma secuencial, el event loop se encarga de organizar y gestionar estas tareas de manera asíncrona, lo que permite que el programa siga respondiendo a otros eventos mientras espera la finalización de operaciones lentas.

Imagina que es el encargado de recibir todas las peticiones y eventos que ocurren mientras el programa está en ejecución y luego decide cómo y cuándo deben ser atendidas.

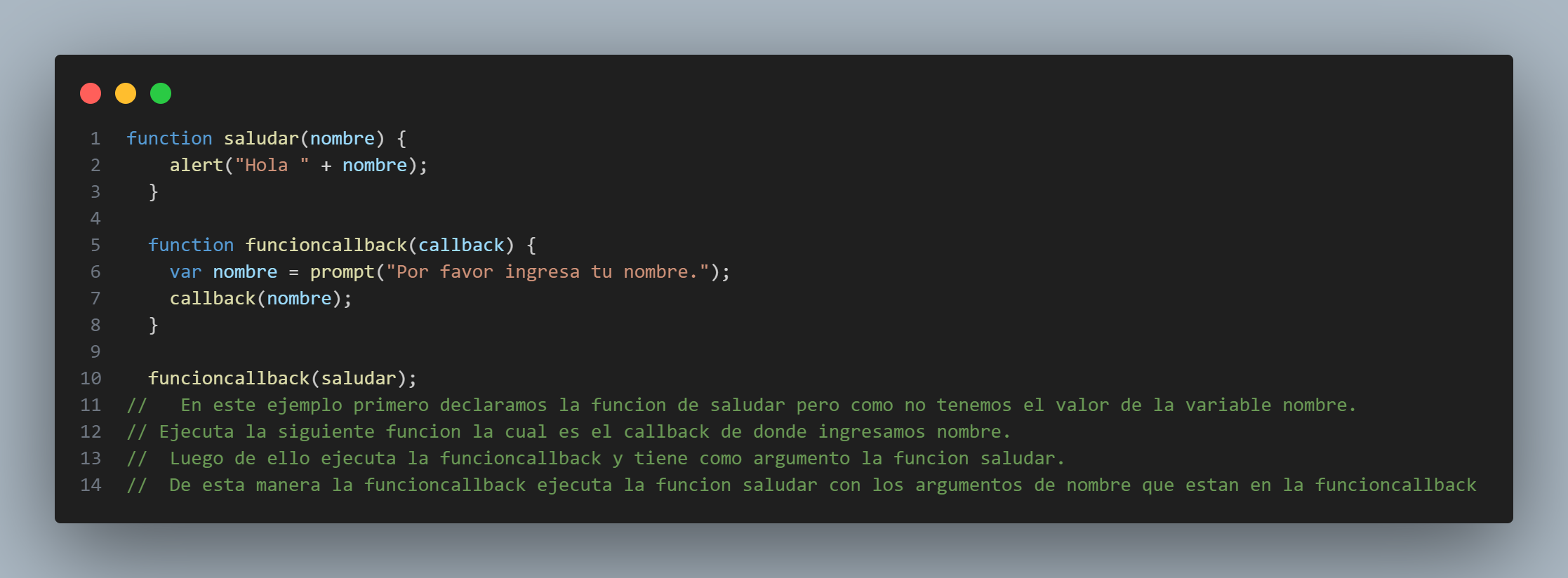
Es el encargado de gestionar y organizar las tareas de manera asíncrona lo que permite que el programa responda a todos los eventos pero tomando por importancia a unos y dándoles un orden.



1. **Que son los callbacks (de un ejemplo)**

Es básicamente usar datos de otras funciones en una función externa y luego ejecutar dicha función. Es enredado la explicación por texto así que ve el código ejemplo.

Cuando se encuentra disponible una función que toma mucho tiempo para completarse, puede bloquear el call stack y ralentizar todo el proceso del event loop. Sin embargo, JavaScript tiene un as bajo la manga: la callback queue o cola de devoluciones de llamada. Esta cola almacena todas las funciones que están listas para ser ejecutadas una vez que el call stack esté vacío. Así, el event loop puede continuar dando vueltas y atendiendo a otros eventos sin problemas.



1. **Que son las promesas (de un ejemplo)**

Las promesas son como bien dice algo que pensamos que se cumplirá a futuro, pero puede que pase algo y bien se cumpla o no se cumpla o quede en espera mientras se cumple. Then da la respuesta que este colocada en reponse y catch muestra el error del mismo. Finally se ejecuta siempre así se cumpla promesa o no se cumpla.

Nota: los finally no funcionan en promesas en espera.



**5. Que son las funciones asíncronas (async / await) (de un ejemplo)**

1. ¿Qué es async/await?

async y await son características introducidas en ECMAScript 2017 (también conocido como ES8) que facilitan la escritura y gestión de código asíncrono en JavaScript. Permiten escribir código asíncrono de manera más limpia y legible, evitando el anidamiento excesivo de callbacks.

2. ¿Cómo funciona async/await?

async: Se utiliza para declarar una función como asíncrona. Esto significa que la función retornará una promesa automáticamente y permitirá el uso de la palabra clave await dentro de ella.

await: Se utiliza dentro de funciones marcadas como async para esperar la resolución de una promesa. Cuando se encuentra una expresión await, la ejecución de la función se detiene hasta que la promesa sea resuelta. Mientras tanto, el control se devuelve al event loop, lo que permite que otras tareas se ejecuten.

