**PROGRAMACION ASINCRONA**

**¿Qué es la programación asíncrona?**

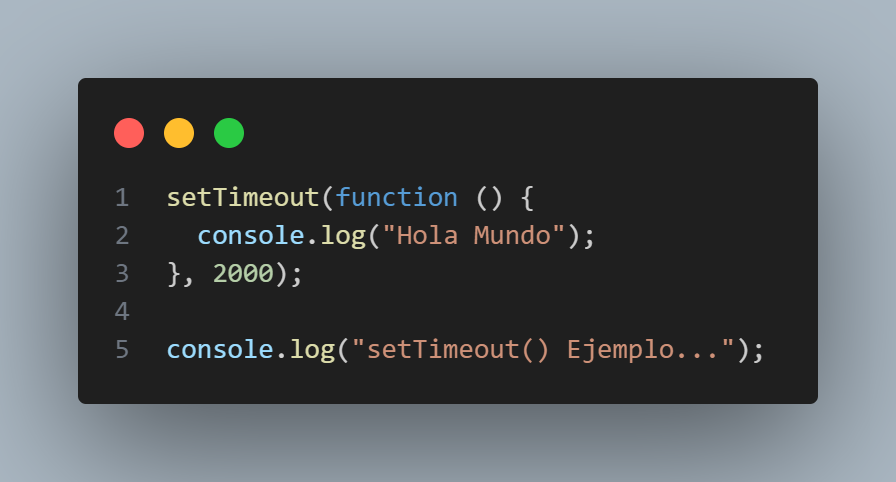
La programación asíncrona permite al programa trabajar en distintas áreas simultáneamente sin tener que esperar a completar una tarea para pasar a la siguiente. Esto permite que un programa sea capaz de hacer mas cosas en menos tiempo.

A diferencia de la programación sincrónica que necesita que se complete una tarea para continuar con la siguiente.

**temporizadores:**

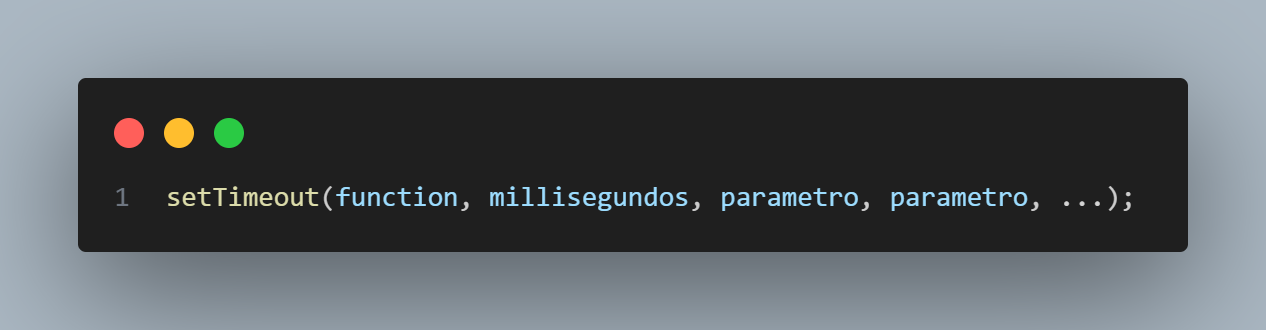
**SET TIME OUT**

El método setTimeout() permite ejecutar un fragmento de código, una vez transcurrido un determinado tiempo. Es una forma de establecer un temporizador para ejecutar código de JavaScript en un momento determinado



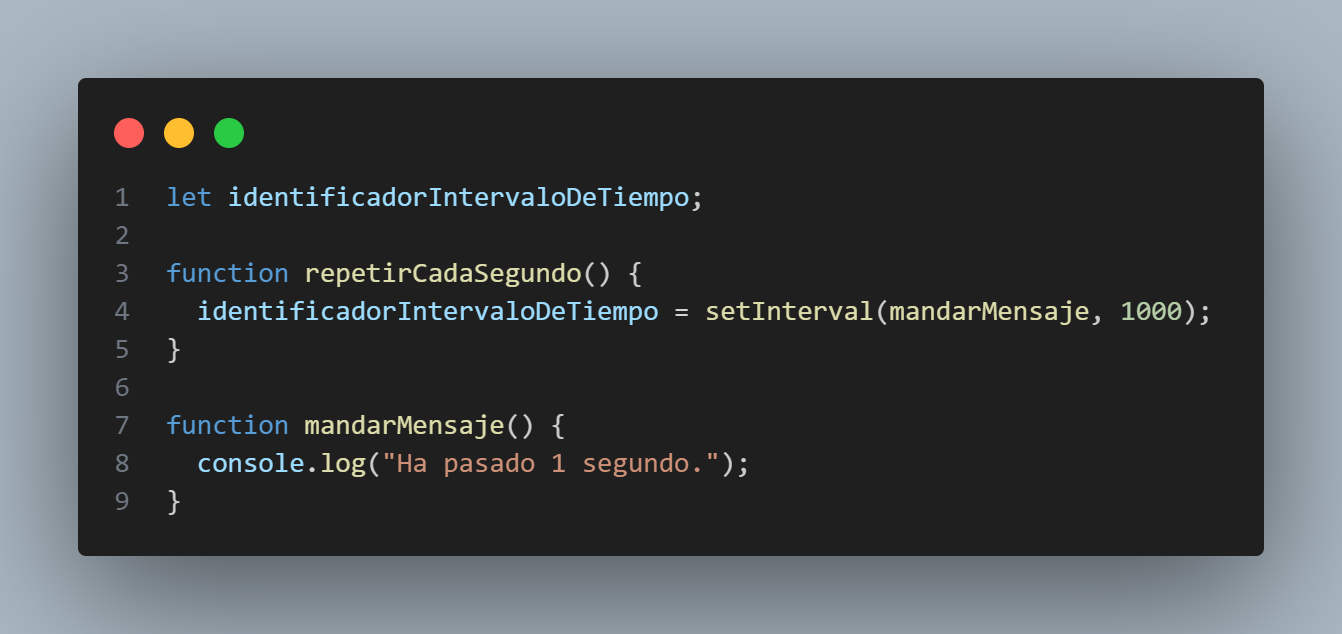
el código imprimirá "Hola Mundo" en la consola de JavaScript después de que hayan pasado 2 segundos:

**SINTAXIS**



**SET INTERVAL**

setInterval() se usa para hacer que una función se repita con un tiempo de retraso entre cada ejecución, se le pasan dos parametros: la función que se quiere llamar y el tiempo en milisegundos a retrasar cada ejecución de la función.



setInterval() continuará ejecutándose hasta que sea borrada.

**CLEARTIMEOUT Y CLEARINTERVAL**

Son funciones para detener los eventos de tiempo clearTimeout() y clearInterval().

Cada función de temporizador se guarda cuando alguna de estas funciones se ejecuta se le asigna un numero identificador el cual es guardado en dicha variable.

Para detener un temporizador se debe llamar a la correspondiente función para borrarlo y pasarle la variable identificadora que corresponde al temporizador que se quiere detener.



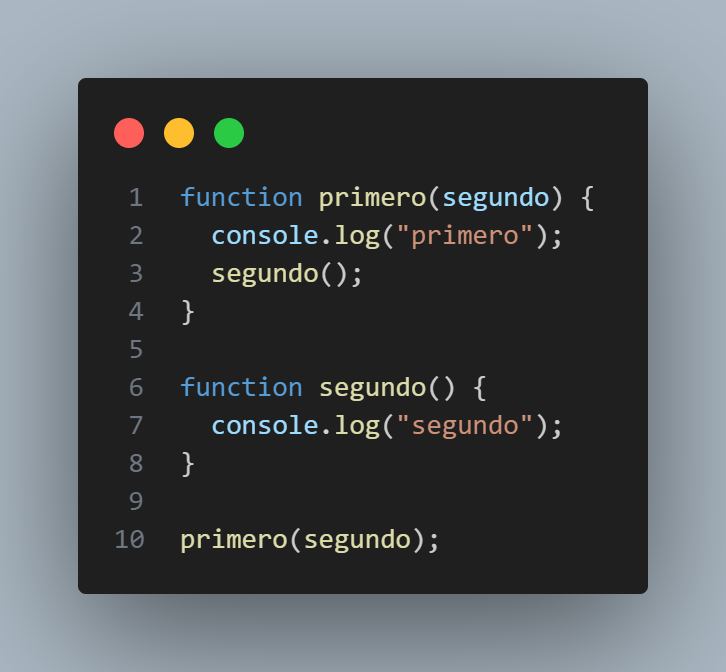
(para Interval funciona igual)

**ASINCRONÍA Y EVENT LOOP**

**(falta)**

**CALL BACK**

Una función de callback es una función que se pasa a otra función como un argumento, que luego se invoca dentro de la función externa para completar algún tipo de rutina o acción.



**PROMESAS**

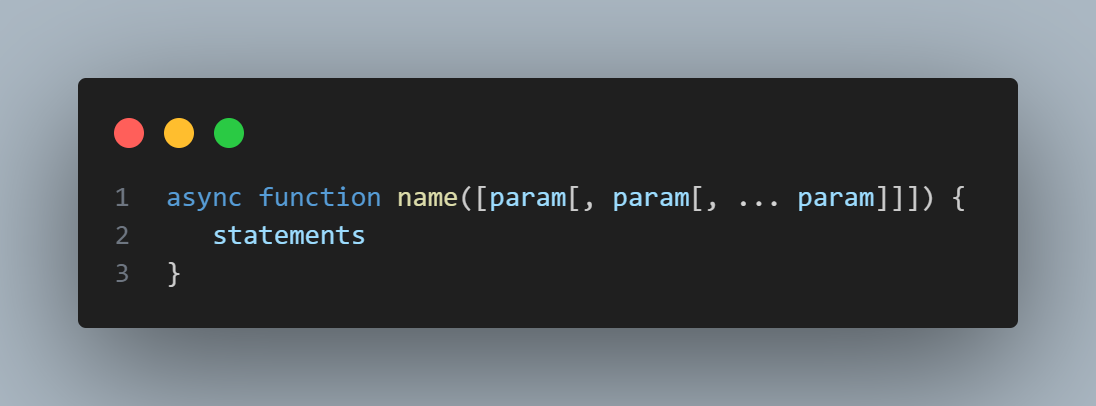
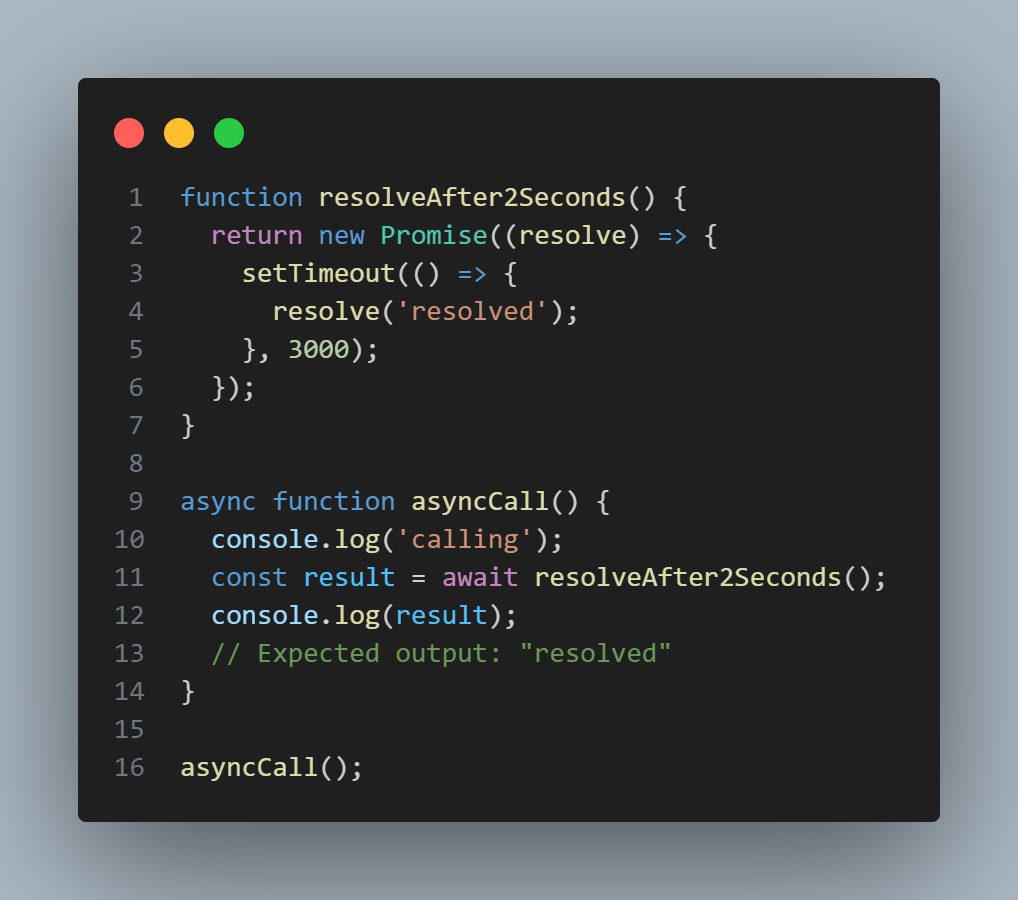
Una promesa es un objeto que representa la terminación o el fracaso de una operación asincrónica, se pueden consumir promesas ya creadas o crear nuevas.

Esencialmente, una promesa es un objeto devuelto al cual se adjuntan funciones callback, en lugar de pasar callbacks a una función.

funciones asíncronas

**ASYNC AWAIT**

la declaración de función **async** define una función asíncrona la cual devuelve un objeto



Operaciones de entrada y salida

**OPERACIONES BLOQUEANTES Y NO BLOQUEANTES**

**Bloqueantes:**  las operaciones bloqueantes son aquellas que detienen la ejecución del código hasta que se complete la operación, durante este tiempo el hilo principal de ejecución queda inactivo.

**No bloqueantes:** las operaciones no bloqueantes son aquellas que permiten continuar con la ejecución del código mientras se lleva a cabo la operación en lugar de esperar a que la operación se complete, esto evita que el hilo principal de ejecución se bloquee y permite que la aplicación sea mas receptiva.

**HILO UNICO (SINGLE THREAD)**

En un entorno de hilo único, el programa solo ejecuta una secuencia de instrucciones a la vez.

Esto significa que las tareas se ejecutan secuencialmente, una después de la otra.

Los programas de un solo hilo son simples de entender y escribir, pero pueden ser menos eficientes para manejar tareas concurrentes o realizar múltiples operaciones al mismo tiempo.

**MULTI HILO (MULTI THREAD)**

En un entorno multihilos, el programa puede ejecutar múltiples secuencias de instrucciones simultáneamente.

Cada secuencia de instrucciones se llama "hilo".

Los programas multihilos pueden realizar múltiples tareas simultáneamente, lo que puede mejorar significativamente la eficiencia y el rendimiento del programa, especialmente en sistemas con múltiples núcleos de CPU.

Sin embargo, la programación multihilo puede ser más compleja y propensa a errores debido a problemas como las condiciones de carrera y la sincronización de datos entre hilos.

**QUE ES LIFO EN PROGRAMACIÓN**

**LIFO: Last-In, first-Out**

LIFO significa "Last In, First Out", que en español se traduce como "último en entrar, primero en salir". Es un principio utilizado en estructuras de datos, como por ejemplo en pilas (stacks). En una estructura LIFO, el último elemento añadido a la estructura es el primero en ser retirado. Esto se asemeja a apilar objetos uno encima del otro, donde el último objeto colocado en la pila es el primero en ser retirado cuando desapilamos. Es una forma común de organizar datos, especialmente en algoritmos de procesamiento de información.

**FIFO: First-In, First-Out**

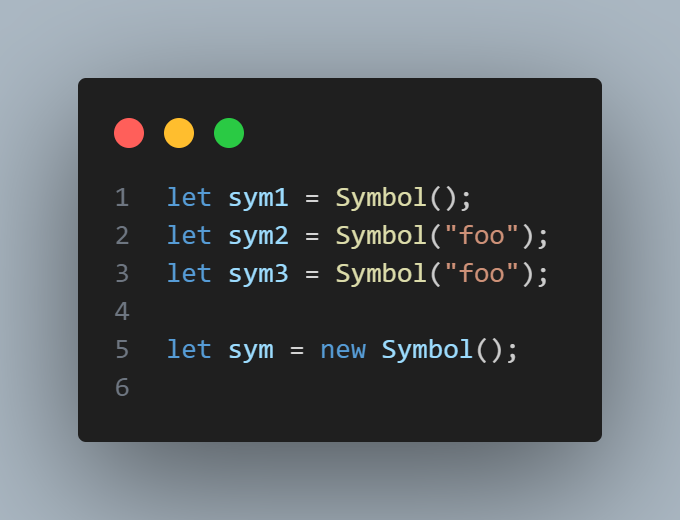
En programación, FIFO significa "First In, First Out", que en español se traduce como "primero en entrar, primero en salir". Al contrario de LIFO, en una estructura FIFO, el primer elemento que se añade a la estructura es el primero en ser retirado. Es similar a formar una fila, donde el primer elemento que se une a la fila es el primero en ser atendido o retirado. FIFO es utilizado en diversas estructuras de datos, como colas (queues), donde los elementos se manejan siguiendo este principio de ordenamiento.

**NUEVOS TIPOS Y CARACTERISTICAS**

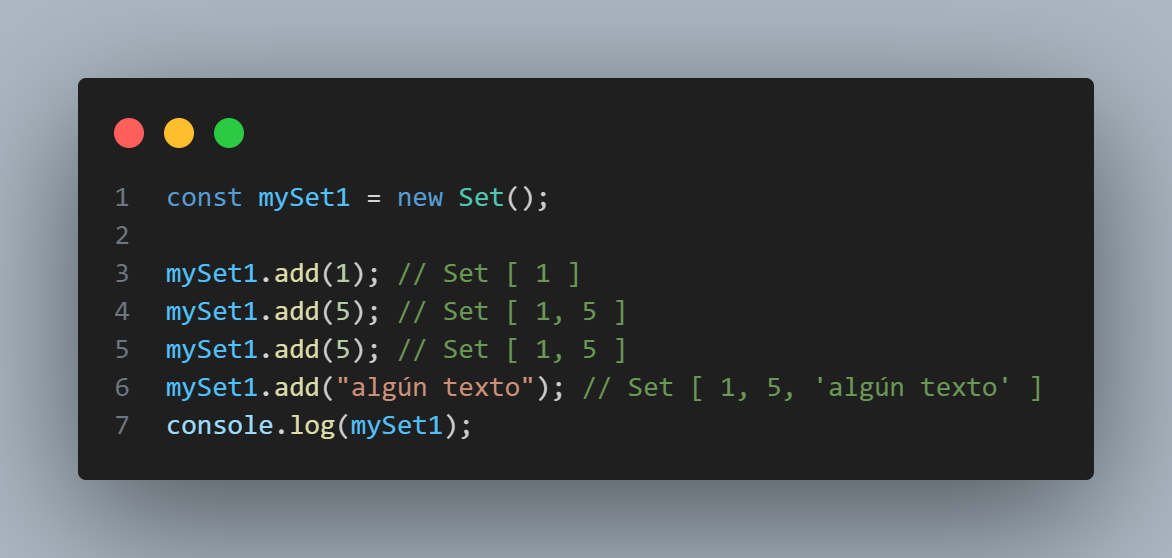
**SYMBOL**

Es un tipo de dato primitivo, así como String, number, boolean, undefined y demás.

Los símbolos son valores únicos e inmutables que se pueden usar como identificadores para propiedades de objetos. Se crean utilizando la función Symbol() sin la palabra clave new.



**SETS**

Los objetos Set son colecciones de valores. Puede iterar a través de los elementos de un conjunto en orden de inserción. Un valor en un Set solo puede ocurrir una vez; es único en la colección del Set.

Constructor

Set()

Crea un nuevo objeto Set.

Propiedades de instancia

Set.prototype.size

Devuelve el número de valores en el objeto Set.

Métodos de instancia

Set.prototype.add(value)

Añade value al objeto Set. Devuelve el objeto Set con el valor añadido.

Set.prototype.clear()

Elimina todos los elementos del objeto Set.

Set.delete(value)

Elimina el elemento asociado a value y devuelve un booleano que afirma si un elemento se eliminó correctamente o no. Set.prototype.has(value) devolverá false después.

Set.prototype.has(value)

Devuelve un booleano que afirma si un elemento está presente con el valor dado en el objeto Set o no.

Métodos de iteración

Set.[@@iterator]()

Devuelve un nuevo objeto iterador que genera los values de cada elemento del objeto Set en el orden de inserción.

Set.values()

Devuelve un nuevo objeto iterador que genera los values de cada elemento del objeto Set en el orden de inserción.

Set.keys()

Un alias para Set.prototype.values().

Set.entries()

Devuelve un nuevo objeto iterador que contiene un arreglo de [value, value] para cada elemento del objeto Set, en orden de inserción.

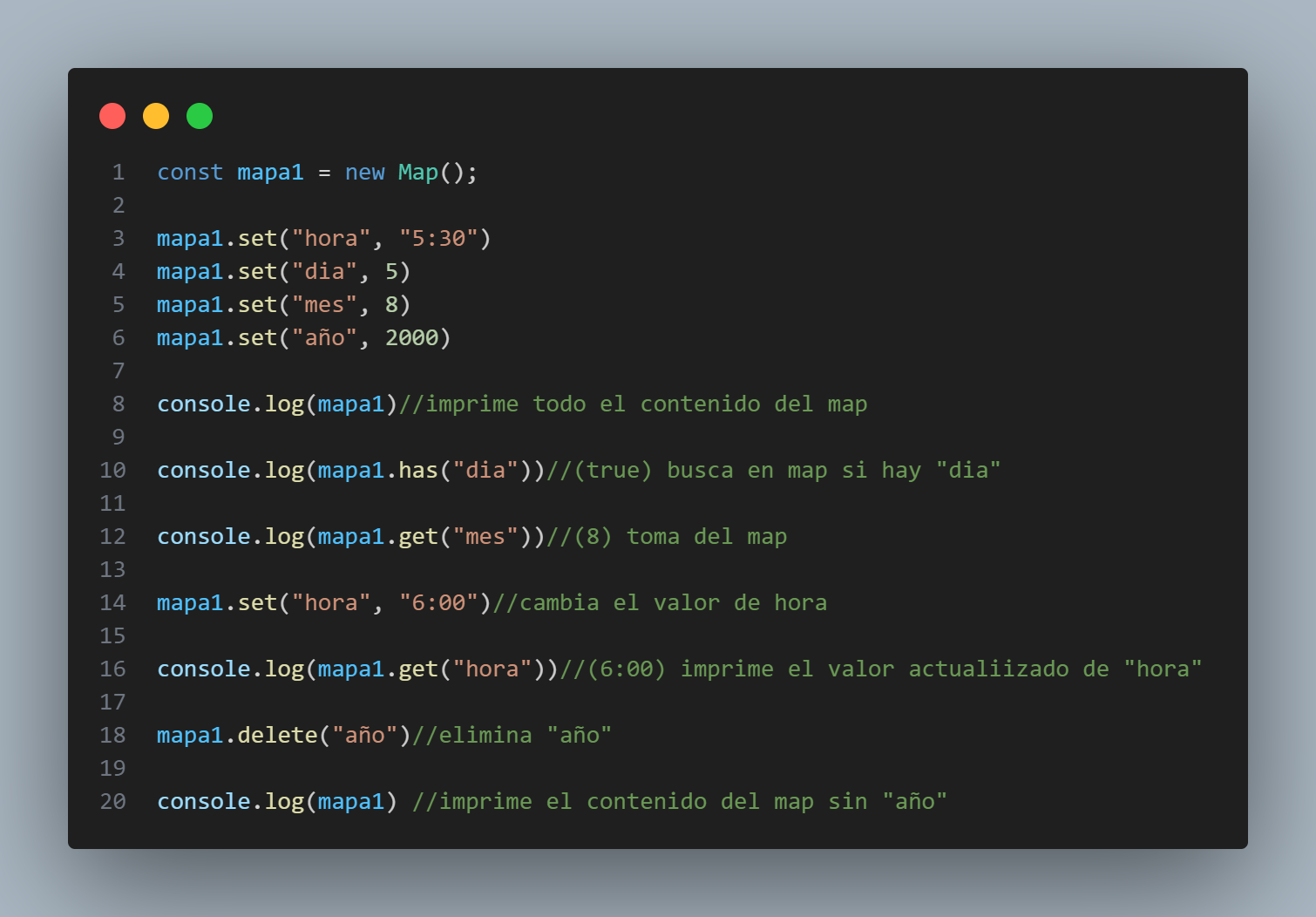
Esto es similar al objeto Map, de modo que la clave de cada entrada es la misma que su valor para un Set.

Set.forEach(callbackFn[, thisArg])

Llama a callbackFn una vez por cada valor presente en el objeto Set, en orden de inserción. Si se proporciona un parámetro thisArg, se utilizará como valor this para cada invocación de callbackFn.

**MAP**

Los Map en Javascript son estructuras de datos nativas que permiten implementar una estructura de tipo mapa, es decir, unas estructuras donde tiene valores guardados a través de una clave para identificarlos. Comúnmente, esto se denomina pares clave-valor.



**WEAKSETS**

El objeto WeakSet te permite almacenar objetos débiles en una colección.

Los objetos WeakSet son colecciones de objetos. Al igual que Set, cada objeto WeakSet puede estar solo una vez; todos los objetos en una colección WeakSet son únicos.



**WEAKMAPS**

WeakMap es una colección similar a Map que permite solo objetos como propiedades y los elimina junto con el valor asociado una vez que se vuelven inaccesibles por otros medios.



**ITERABLES**

Los objetos iterables son una generalización de arrays. Es un concepto que permite que cualquier objeto pueda ser utilizado en un bucle

Las matrices o arrays son iterables. Pero hay muchos otros objetos integrados que también lo son. Por ejemplo, las cadenas o strings son iterables también. Como veremos, muchos operadores y métodos se basan en la iterabilidad.

**PROXIES**

El objeto Proxy permite crear un intermediario para otro objeto, el cual puede interceptar y redefinir operaciones fundamentales para dicho objeto mediante un handler o manipulador.

Un Proxy se crea con dos parámetros:

target: el objeto original que se quiere envolver.

handler: un objeto que define cuáles operaciones serán interceptadas y cómo redefinir dichas operaciones.

**PROPIEDADES DINAMICAS DE LOS OBJETOS**

Las propiedades dinámicas de los objetos se refieren a la capacidad de los objetos en programación para tener propiedades que pueden ser agregadas, modificadas o eliminadas en tiempo de ejecución. En muchos lenguajes de programación orientados a objetos, como JavaScript, Python y Ruby, los objetos son flexibles y pueden adaptarse dinámicamente a los cambios en el entorno de ejecución. Esto permite una mayor flexibilidad y adaptabilidad en el desarrollo de software.