**Siguiente consulta JS:**

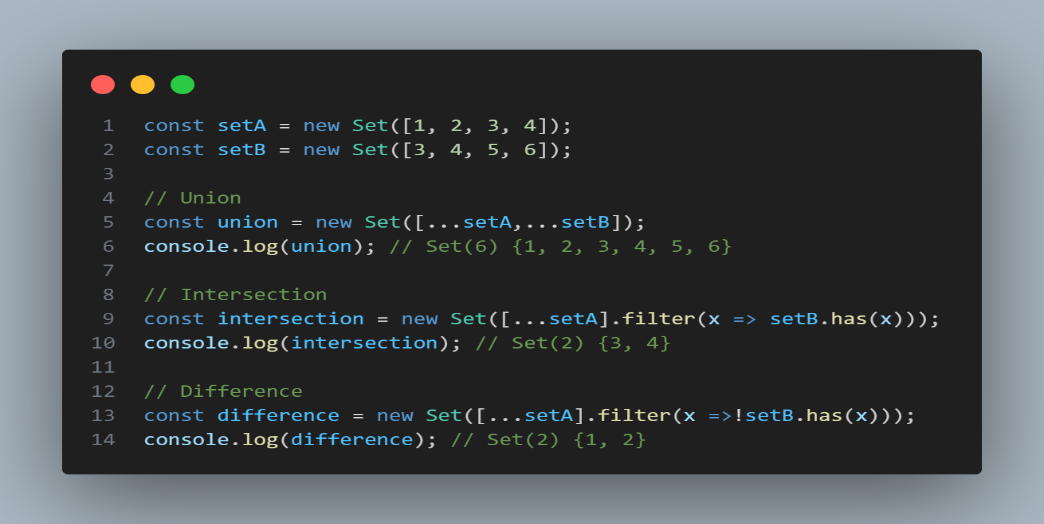
**Symbols**

Los símbolos JavaScript son un nuevo tipo de datos primitivos introducidos en la versión ES6 del lenguaje. Se utilizan para representar valores únicos que pueden usarse como identificadores o claves en objetos. También se utilizan para crear propiedades y métodos privados en clases.

**.**

**Sets**

Los conjuntos en JavaScript son un tipo de estructura de datos que le permite almacenar valores únicos. A continuación se muestra un ejemplo de cómo crear un conjunto y realizar algunas operaciones en él

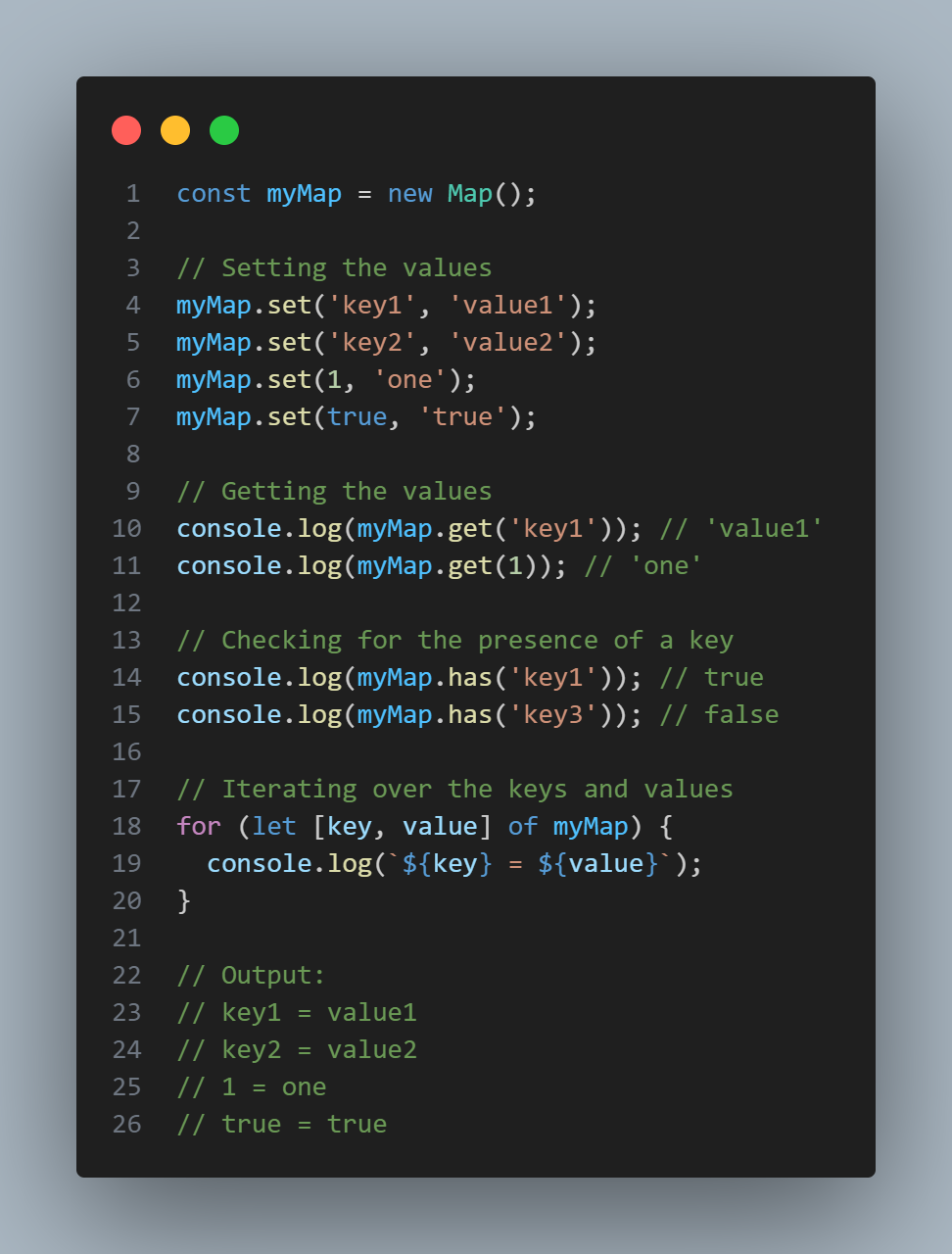




**Maps**

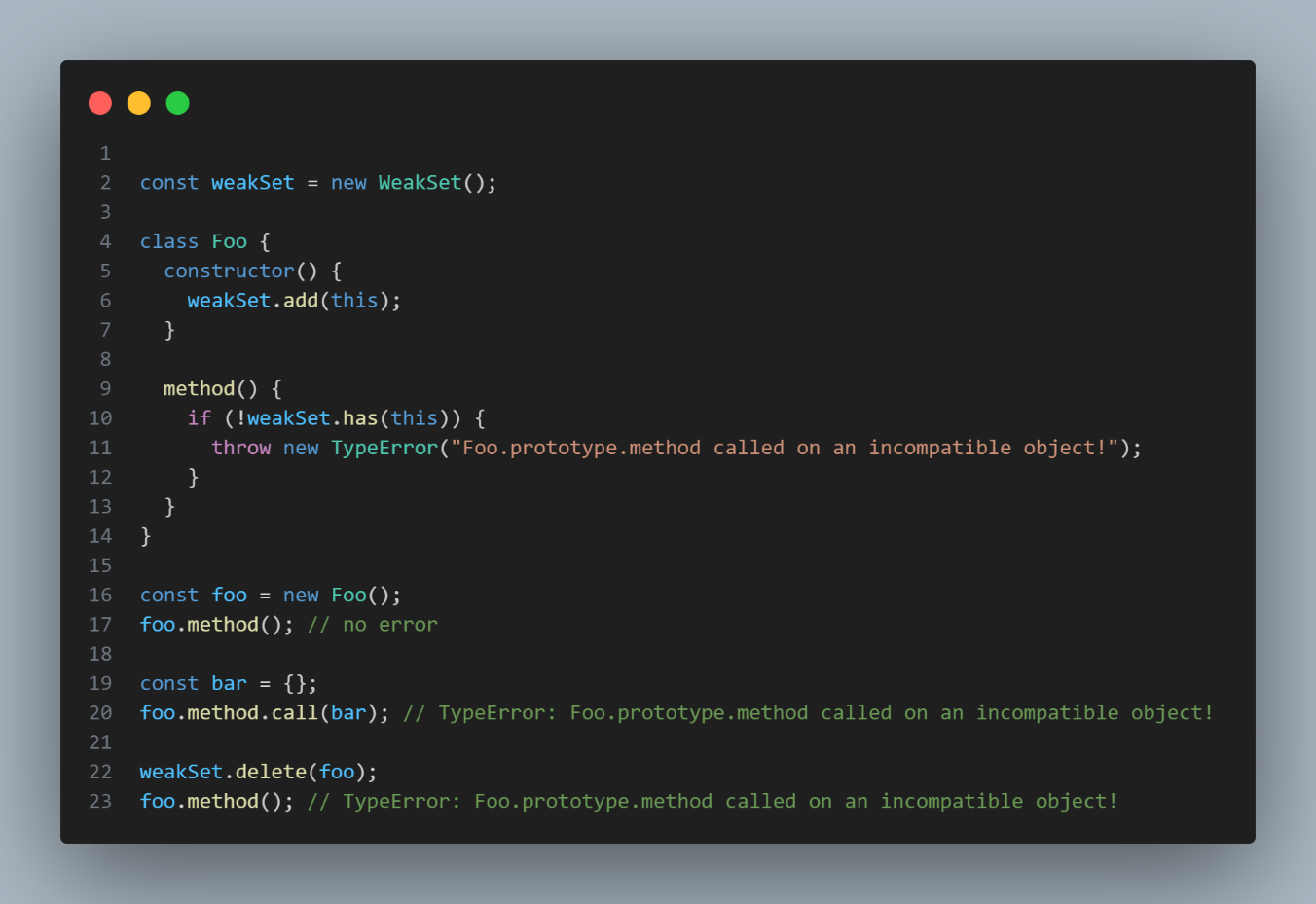
En JavaScript, un mapa es una colección de elementos de datos con clave, al igual que un objeto. Pero la principal diferencia es que Map permite claves de cualquier tipo, mientras que las claves de Objeto son siempre cadenas o símbolos.

Los mapas son útiles cuando desea asociar valores con claves que no son cadenas, como objetos o funciones. Los mapas también le permiten iterar fácilmente sobre las claves y los valores en el orden en que se agregaron.

****

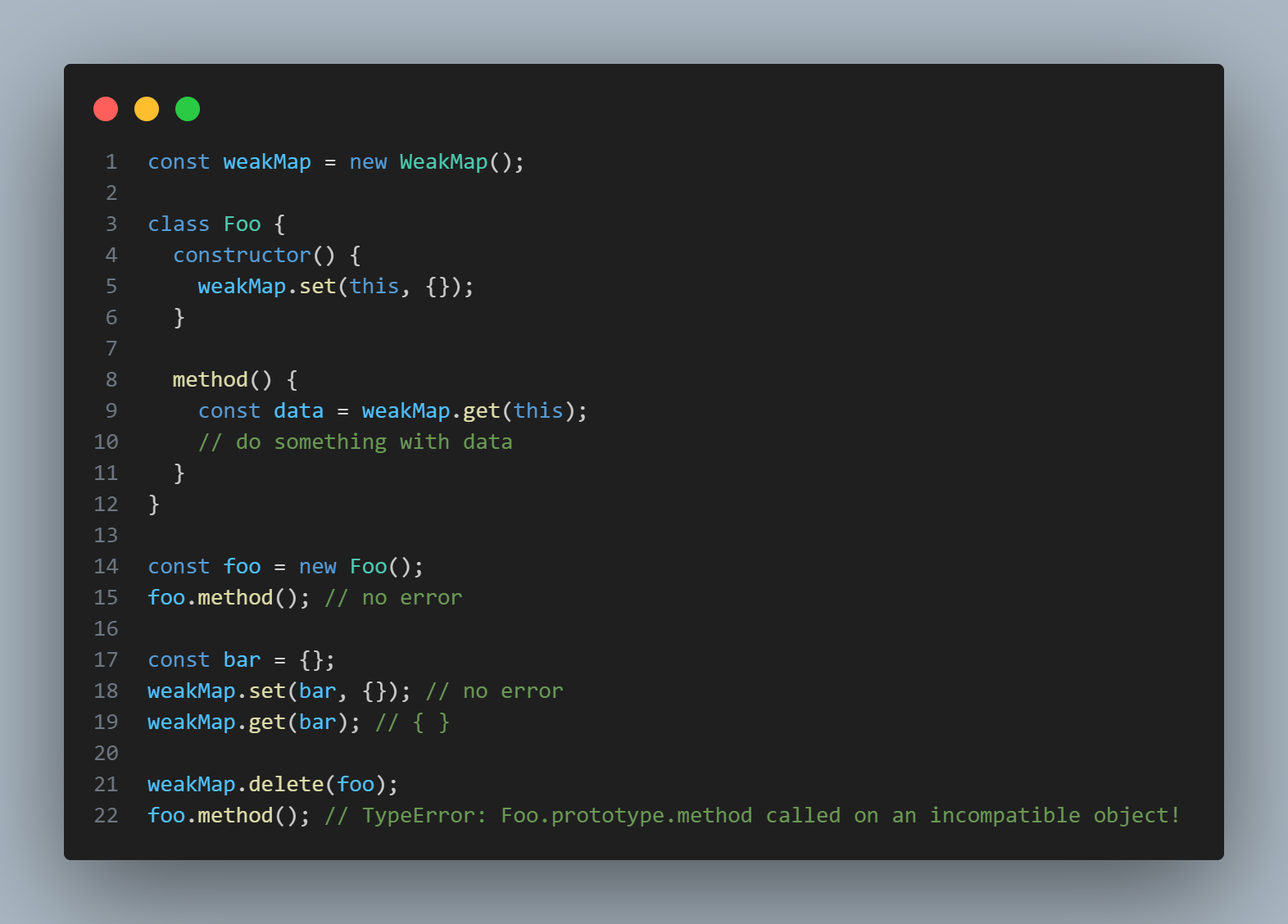
**WeakSets**

Los WeakSets son un tipo de colección en JavaScript que le permite almacenar objetos de forma débil, lo que significa que si no hay otras referencias a un objeto almacenado en un WeakSet, puede ser recolectado como basura. Los WeakSets son similares a los Conjuntos, pero solo pueden almacenar objetos, no valores primitivos.



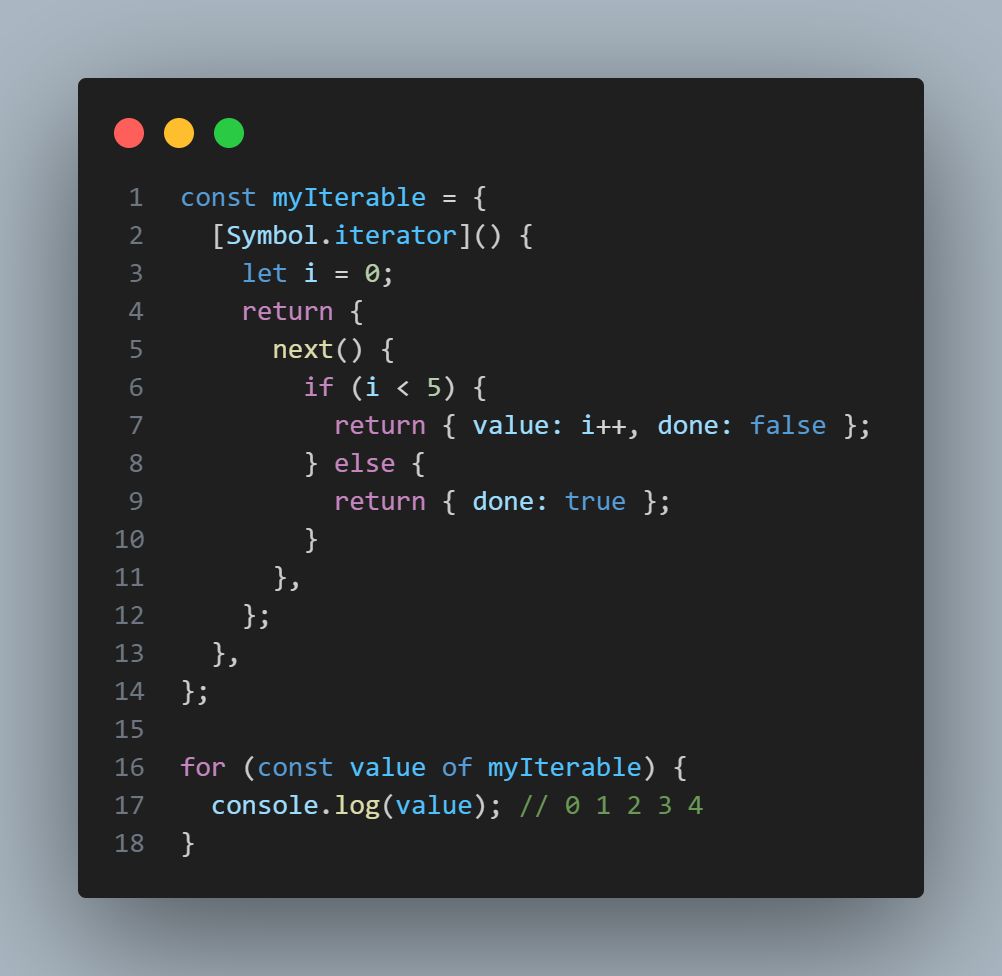
**WeakMaps**

WeakMaps es un tipo de colección en JavaScript que le permite almacenar pares clave-valor de manera débil, lo que significa que si no hay otras referencias a una clave almacenada en un WeakMap, puede ser recolectada como basura. WeakMaps son similares a Maps, pero solo pueden almacenar objetos como claves, no valores primitivos.

****

**Iterables**

En JavaScript, un iterable es un objeto que define su comportamiento de iteración, como cadenas, matrices y conjuntos. Un objeto iterable debe implementar el método @@iterator, que devuelve un objeto iterador. El objeto iterador debe implementar el siguiente método, que devuelve un objeto con valor y propiedades terminadas.



**Generadores**

Los generadores (también conocidos como "generators" en inglés) son una característica de los lenguajes de programación que permiten a una función producir una secuencia de valores de manera lazy, es decir, solo cuando se necesita.

En otras palabras, un generador es una función que puede ser pausada y reanudada en cualquier momento, permitiendo que se produzcan valores de manera incremental, en lugar de tener que producir todos los valores al mismo tiempo.

Los generadores se caracterizan por:

Evaluación perezosa : Los generadores solo producen valores cuando

Producción incremental : Los generadores

Pausa y reanudación : Los generadores pueden ser pausados

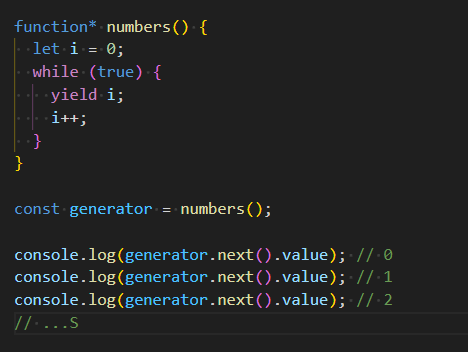
Los generadores se utilizan principalmente en situaciones donde se necesita:

Procesar grandes cantidades de datos :

Realizar operaciones asíncronas : Los generadores permiten realizar operaciones asíncronas de manera más fácil y eficiente.

Crear iteradores personalizados : Los generadores permiten crear iteradores personalizados que se adaptan a las necesidades específicas de un problema.

En JavaScript, los generadores se definen utilizando la sintaxis function\*y se utilizan con el método next()para obtener el siguiente valor en la secuencia.



**Iterators**

En JavaScript, un iterador es un objeto que proporciona una forma de acceder a los elementos de una colección, como una matriz o una cadena, uno a la vez. Un objeto iterador debe implementar el nextmétodo, que devuelve un objeto con propiedades valuey done. La valuepropiedad contiene el siguiente elemento de la colección y la donepropiedad es un valor booleano que indica si hay más elementos sobre los que iterar.

****

**Proxies**

En JavaScript, un Proxy es un objeto que actúa como intermediario entre un objeto de destino y el código que accede a él. Un Proxy puede interceptar y modificar operaciones en el objeto de destino, como acceso a propiedades, invocación de funciones o asignación, antes de que se realicen. Esto le permite agregar comportamiento a objetos existentes, como registro, validación o control de acceso.

****

**Propiedades dinámicas de los objetos**

En JavaScript, puede agregar propiedades dinámicas a los objetos utilizando la notación entre corchetes. Esto le permite establecer no sólo los valores de los miembros, sino también los nombres de los miembros de forma dinámica



También puede utilizar la notación de puntos para acceder a las propiedades de un objeto, pero solo puede aceptar un nombre de propiedad literal, no una variable que apunte a un nombre de propiedad.

