Investigación Nuevos tipos y características de JavaScript:

1. Que son los Symbols (de un ejemplo de cada uno)

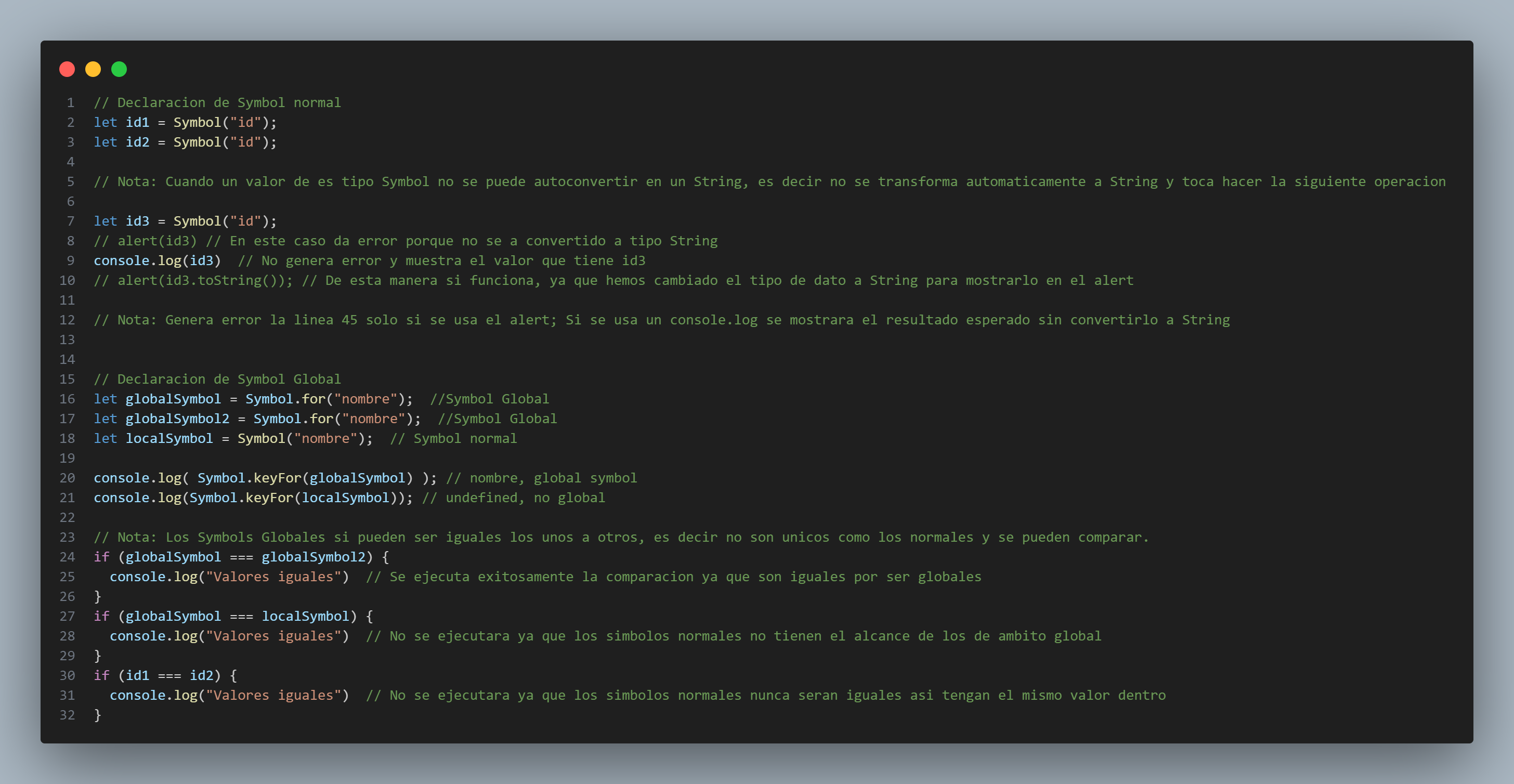
Los symbols son variables únicas, es decir, son datos de valor único que se declaran dentro de una palabra.

Podemos usar los Symbols para hacer objetos con datos reservados y hacer Symbols globales

El Symbol.keyFor utiliza internamente el registro “global symbol registry” para buscar la clave del symbol, por lo tanto, no funciona para los symbols que no están dentro del registro. Si el symbol no es global, no será capaz de encontrarlo y por lo tanto devolverá undefined.

Al usarlos globales el valor si puede ser igual el uno del otro, en caso de ser inexistente lo creara automáticamente

Ejemplos:



1. Que son los sets (de un ejemplo)Los sets funcionan muy parecidos a los arrays, ya que almacenan ordenadamente una lista de datos, de cualquier tipo; fuera de ello permite agregar datos a un map.

Sus principales métodos son:

* new Set([iterable]) – crea el set. El argumento opcional es un objeto iterable (generalmente un array) con los valores para inicializarlo.
* set.add(valor) – agrega un valor, y devuelve el set en sí.
* set.delete(valor) – elimina el valor, y devuelve true si el valor existía al momento de la llamada; si no, devuelve false.
* set.has(valor) – devuelve true si el valor existe en el set, si no, devuelve false.
* set.clear() – elimina todo el contenido del set.
* set.size – es la cantidad de elementos.

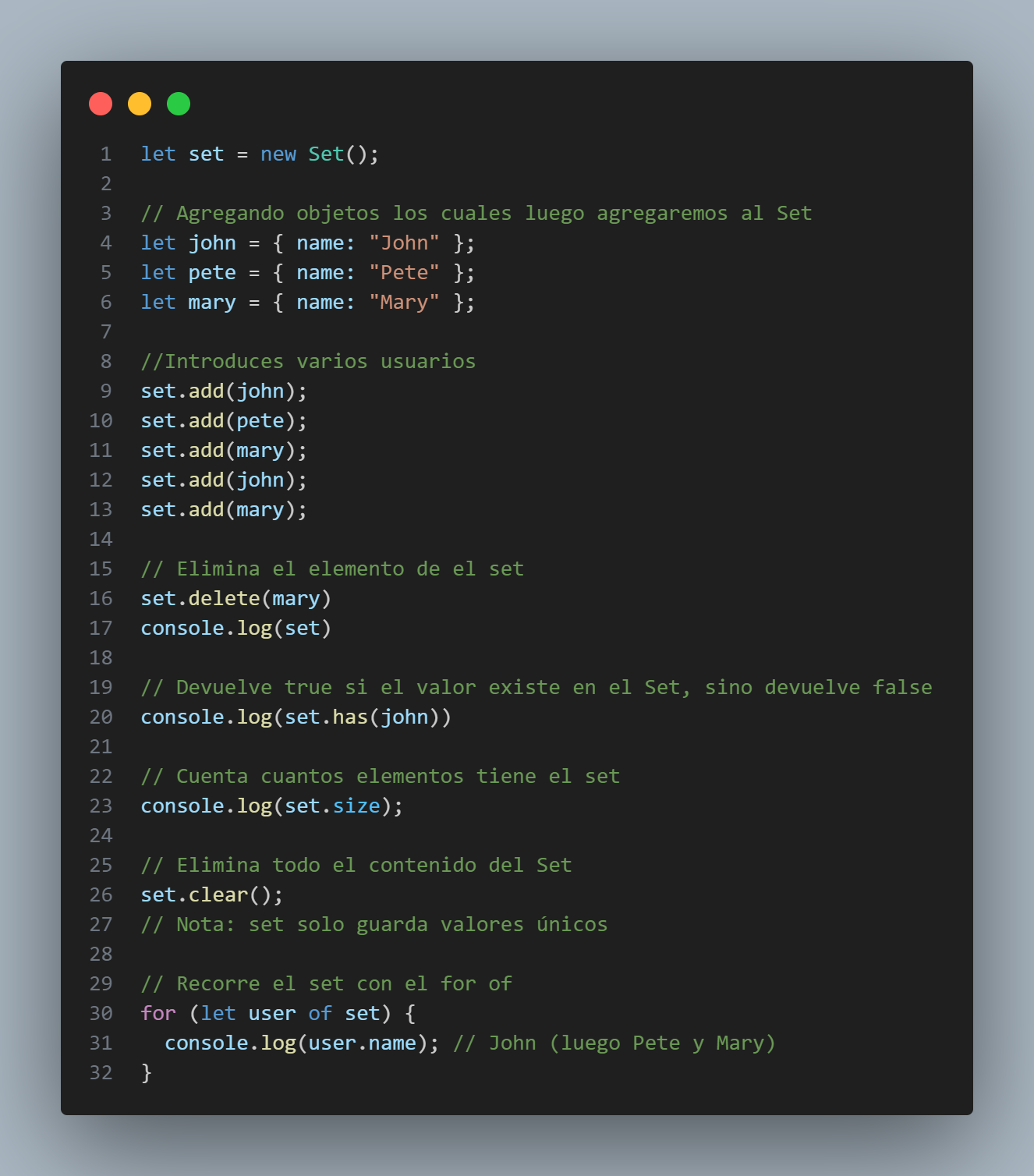
Valores únicos: Un Set no puede contener elementos duplicados. Si se intenta agregar un elemento que ya está presente en el Set, este no se agregará.

Ordenación: Los elementos en un Set se almacenan en el orden de inserción, lo que significa que no hay garantía de ordenación como en un array.

Iterabilidad: Los Set son iterables, lo que significa que se puede recorrer a través de ellos utilizando bucles for...of o métodos como forEach.

Métodos para añadir, eliminar y verificar elementos: Los Set proporcionan métodos como add() para agregar elementos, delete() para eliminar elementos y has() para verificar si un elemento está presente.

Tamaño dinámico: El tamaño de un Set puede cambiar dinámicamente a medida que se agregan o eliminan elementos.

Ejemplo:

1. Que son los Maps (de un ejemplo)

El map es igual a un objeto, una colección de datos identificados por claves aunque map permite claves de cualquier tipo

Los métodos y propiedades son:

new Map() – crea el mapa.

map.set(clave, valor)) – almacena el valor asociado a la clave.

map.get(clave) – devuelve el valor de la clave. Será undefined si la clave no existe en map.

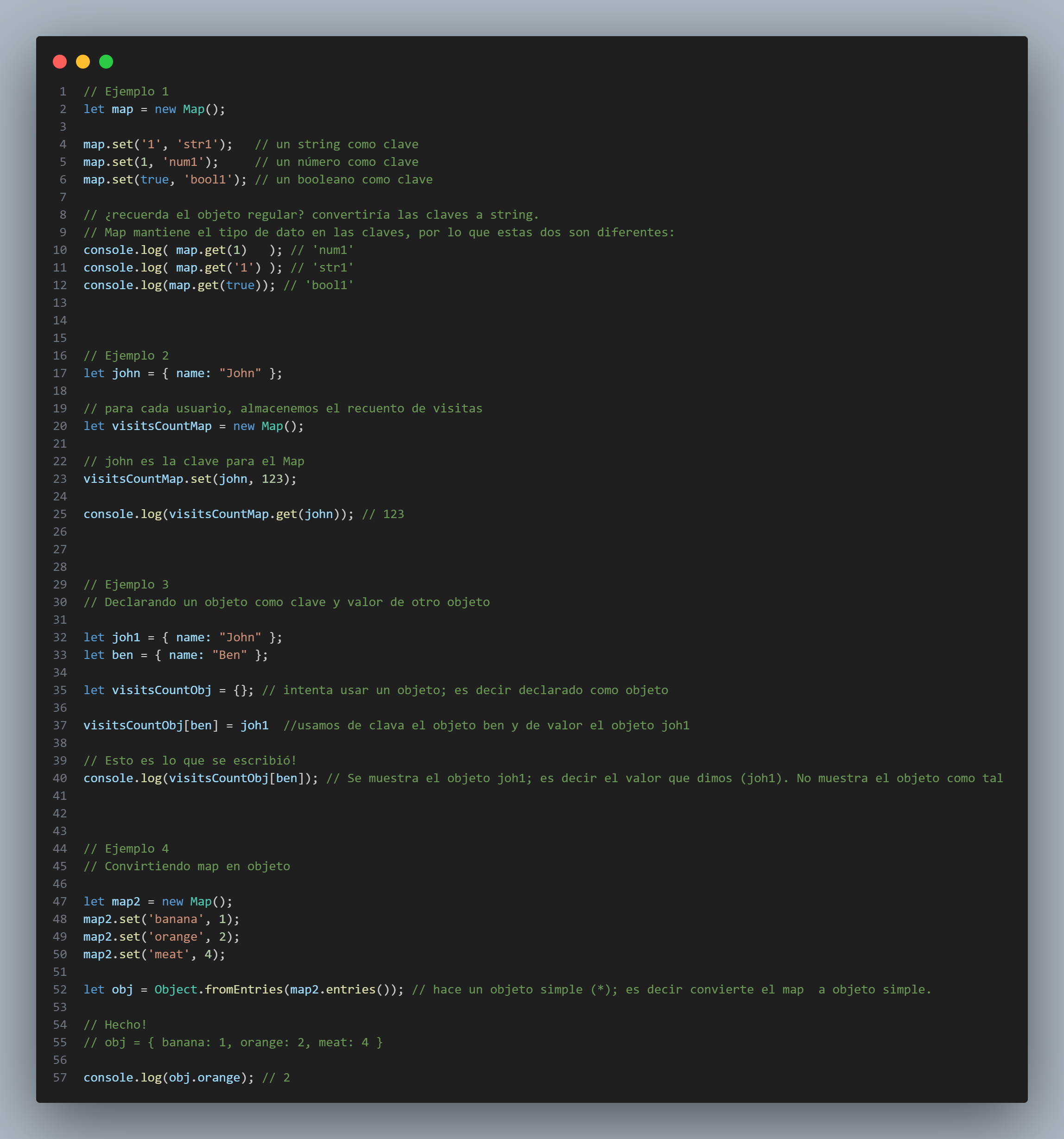
map.has(clave) – devuelve true si la clave existe en map, false si no existe.

map.delete(clave) – elimina el elemento con esa clave.

map.clear() – elimina todo de map.

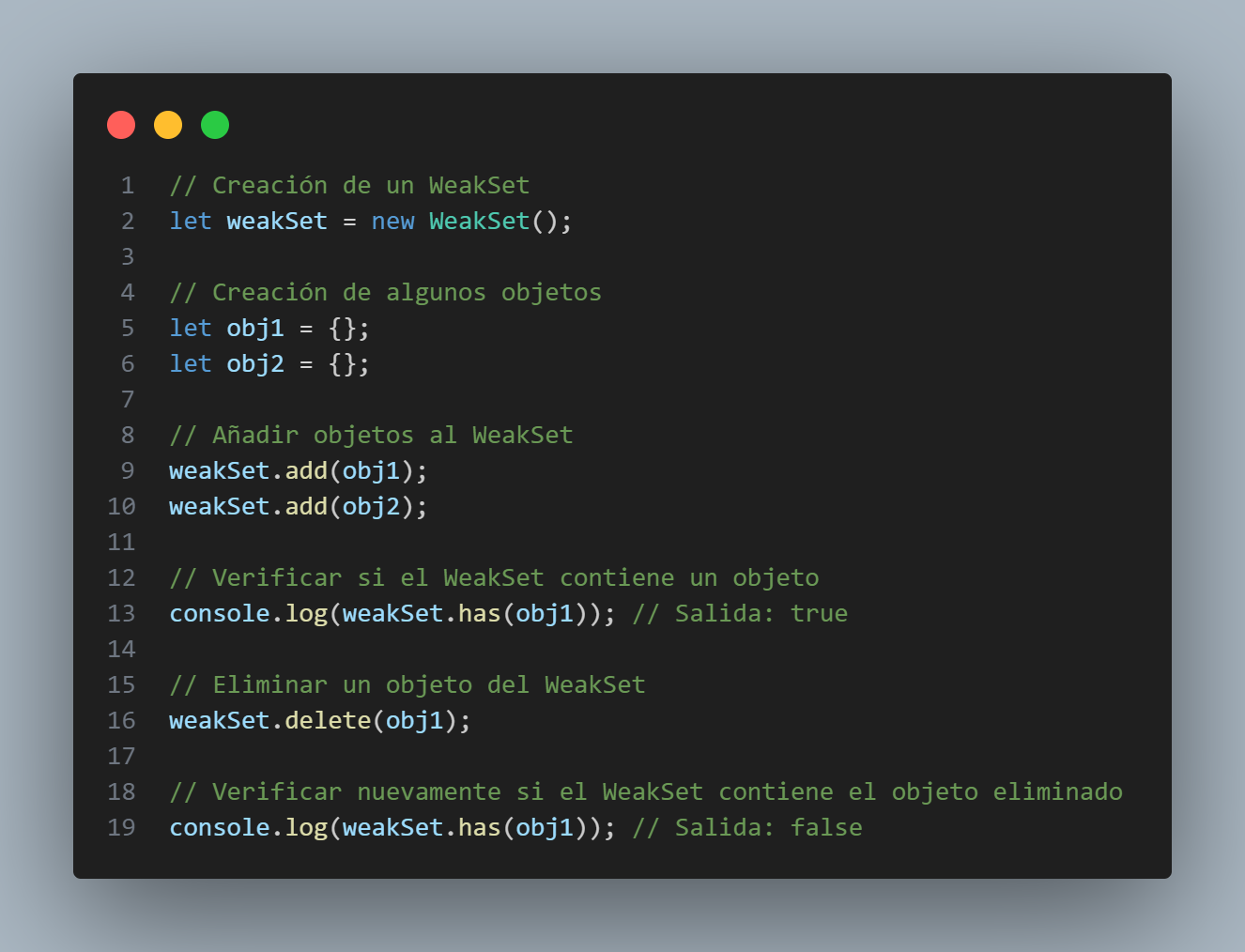
map.size – tamaño, devuelve la cantidad actual de elementos.

**Nota: con Map podemos usar de clave / valor cualquier tipo de dato, es decir: Strings, números, booleanos y objetos.**

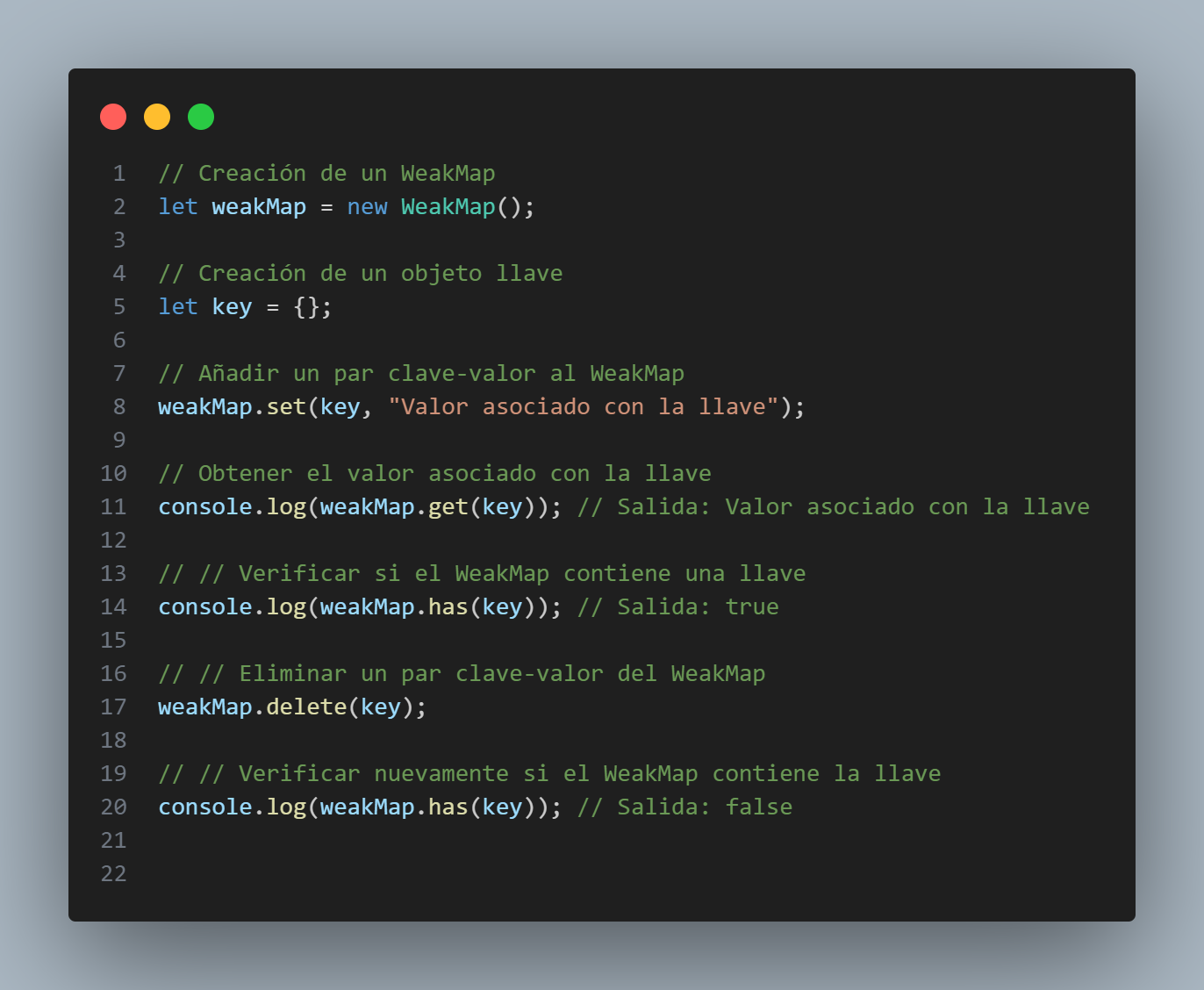
****

1. Que son los WeakSets y WeakMaps (de un ejemplo de cada uno)

* WeakSet es una estructura de datos en JavaScript que almacena objetos únicos. A diferencia de un Set estándar, las referencias a los objetos en un WeakSet son débiles, lo que significa que no previenen que los objetos sean recolectados por el recolector de basura si no hay otras referencias a ellos.

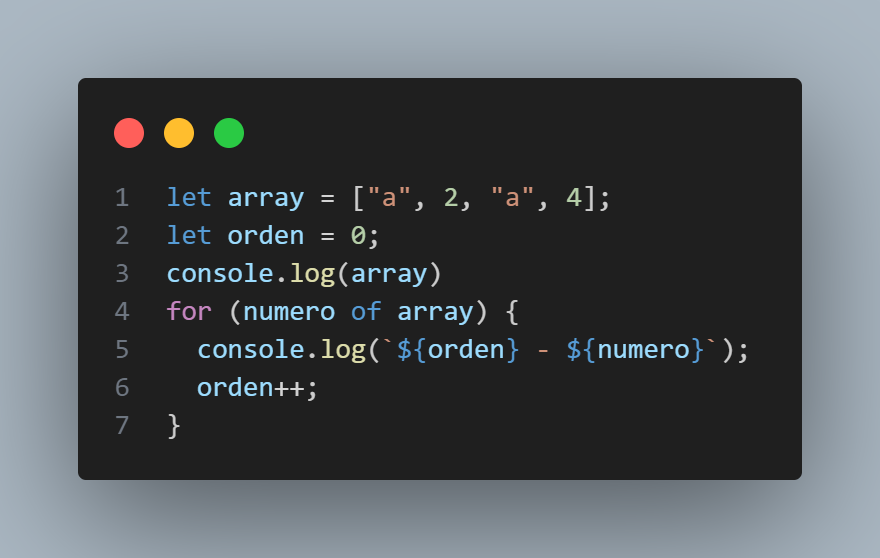


* WeakMap es una estructura de datos en JavaScript que permite asociar pares clave-valor con llaves de objetos, donde las referencias a las llaves son débiles, lo que significa que las llaves pueden ser recolectadas por el recolector de basura si no hay otras referencias a ellas. Esto puede ser útil para evitar la retención de memoria en ciertos casos.



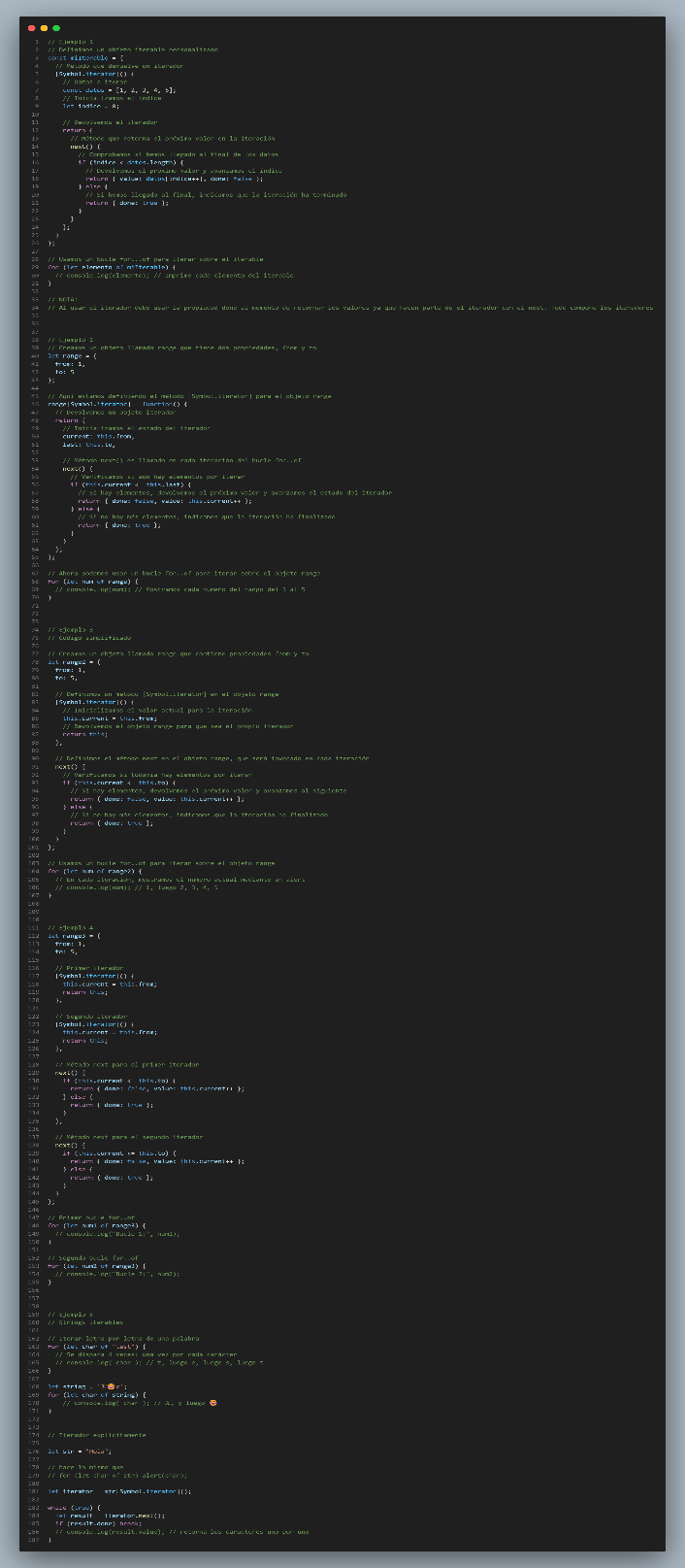
1. Que son los iterables y los iteradores (de un ejemplo)

* Iterable: Es un objeto el cual puede recorrerse desde un inicio hasta un final. Un ejemplo seria un array.



* Iteradores: Un iterable es un objeto que implementa el protocolo de iteración, lo que significa que puede ser iterado mediante un bucle, como un bucle for...of, y accedido secuencialmente. Los iterables en JavaScript pueden ser arrays, objetos con una propiedad [Symbol.iterator], cadenas de texto, mapas, sets, entre otros.

El protocolo de iteración en JavaScript consiste en que un objeto debe tener un método Symbol.iterator que devuelve un iterador. Un iterador es un objeto con un método next() que devuelve un objeto con dos propiedades: value, que es el siguiente valor en la iteración, y done, que es un booleano que indica si la iteración ha finalizado.



1. Que son los generadores (de un ejemplo de cada uno) Los generadores son creados por funciones generadoras function\* f(…) {…}.

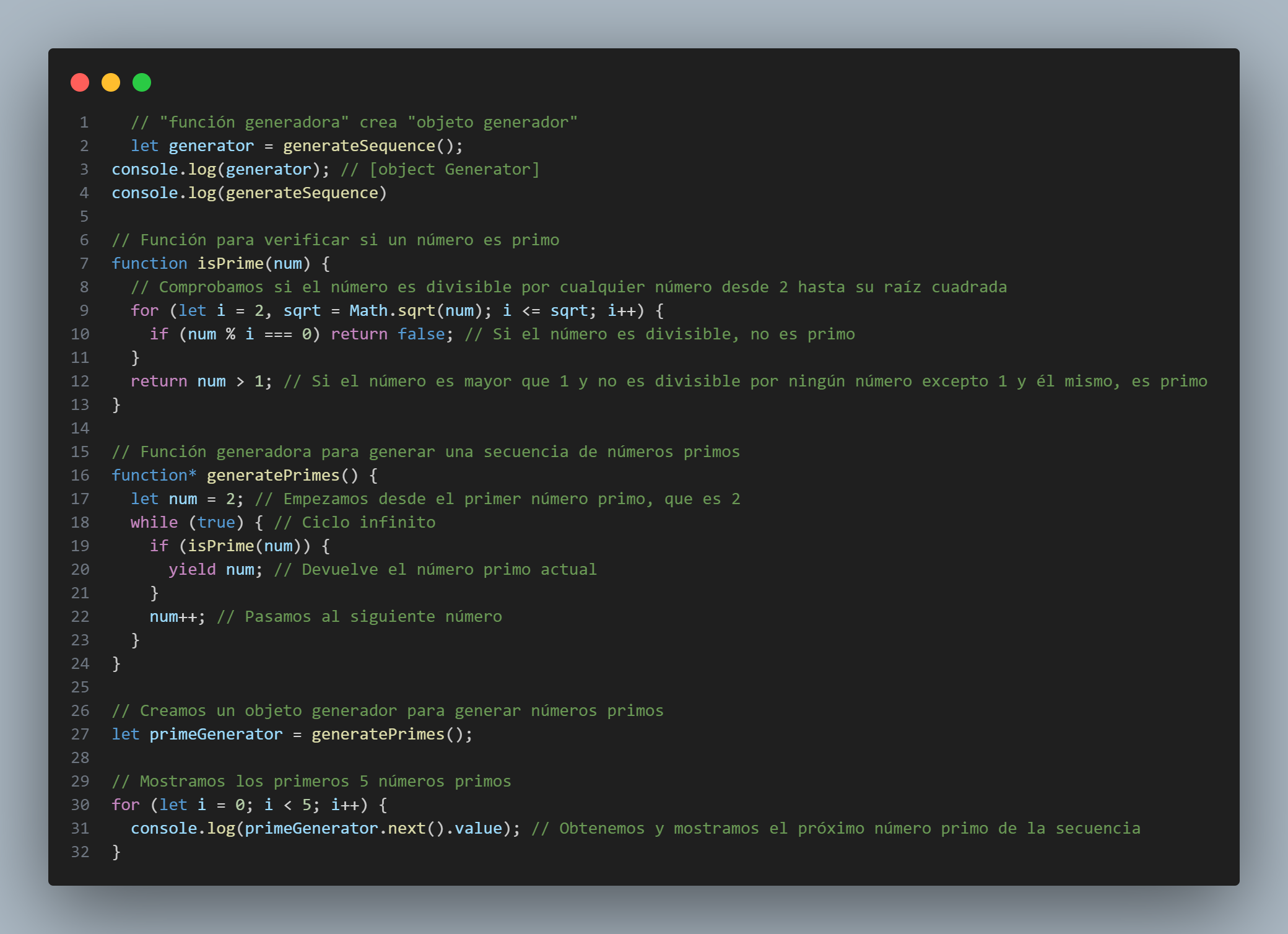
Dentro de los generadores (solo) existe un operador yield.

El código externo y el generador pueden intercambiar resultados a través de llamadas next/yield.

En JavaScript moderno, los generadores rara vez se utilizan. Pero a veces son útiles, porque la capacidad de una función para intercambiar datos con el código de llamada durante la ejecución es bastante única. Y, seguramente, son geniales para hacer objetos iterables.

Además, en el próximo capítulo aprenderemos los generadores asíncronos, que se utilizan para leer flujos de datos generados asincrónicamente (por ejemplo, recuperaciones paginadas a través de una red) en bucles for await ... of.

En la programación web, a menudo trabajamos con datos transmitidos, por lo que ese es otro caso de uso muy importante.



1. Que son los proxies (de un ejemplo)

Los Proxies en JavaScript son objetos especiales que permiten la personalización del comportamiento fundamental de otros objetos. Actúan como intermediarios entre el código que interactúa con un objeto y ese objeto mismo, permitiendo interceptar y controlar las operaciones básicas como la lectura, escritura, enumeración de propiedades, entre otras. Los Proxies ofrecen una forma flexible de definir el comportamiento de un objeto, permitiendo implementar lógica adicional o restringir ciertas operaciones según sea necesario.

Los Proxy toman dos argumentos en su constructor el primer argumento es el objeto objetivo que se va a observar y el segundo argumento es el objeto de configuración especifica del comportamiento personalizado para diferentes operaciones en el objeto objetivo

- target – es el objeto a envolver, puede ser cualquier cosa, incluso funciones.

- handler – configuración de proxy: un objeto que “atrapa”, métodos que interceptan operaciones. Ejemplos, la trampa get para leer una propiedad de target, la trampa set para escribir una propiedad en target, entre otras.

Definición proxy: Objeto que envuelve a otro objeto e intercepta operaciones básicas sobre él, como lo son: Asignar, eliminar y acceder a las propiedades; los proxy necesitan dos componentes, un objeto destino y un objeto controlador. Siendo el objeto objetivo el que intercepta las operaciones. Mientras el objeto controlador es responsable de mantener las trampas o métodos utilizados para capturar estas operaciones.

El proxy de JS es una capacidad que permite la creación de objetos capaces de modificar y personalizar operaciones realizadas sobre otros objetos.

Definición de trampas: Son métodos que interceptan operaciones en el objeto destino, estas trampas incluidas get, set, has, deleteProperty.

* get: esta trampa intercepta los intentos de leer una propiedad en el objeto de destino. Se necesitan dos argumentos: el objeto de destino y la propiedad a la que se accede. La trampa devuelve el valor de la propiedad.
* set: esta trampa captura cualquier esfuerzo para establecer una propiedad en el objeto de destino. Requiere tres parámetros: el objeto de destino en sí, la propiedad que se está estableciendo y el valor actualizado de esa propiedad. El mecanismo tiene la capacidad de alterar el valor que se está estableciendo, o puede generar un error para prohibir que se establezca el valor.
* tiene: esta trampa intercepta los intentos de verificar si existe una propiedad en el objeto de destino. Se necesitan dos argumentos: el objeto de destino y la propiedad que se está verificando. La trampa devuelve un valor booleano que indica si la propiedad existe o no.
* deleteProperty : esta trampa intercepta los intentos de eliminar una propiedad del objeto de destino. Se necesitan dos argumentos: el objeto de destino y la propiedad que se elimina. La captura puede eliminar la propiedad o generar un error para evitar que se elimine la propiedad.

Los proxys permiten invalidarse, lo que hace que sus trampas ya no intercepten operaciones en el objeto objetivo. Para construir un objeto Proxy que pueda invalidarse, utilice la Proxy.revocable()función.

Es un método estático que se utiliza para crear proxys revocables; un proxy revocable es un objeto que actúa como un proxy, pero con la capacidad de ser revocado, es decir puede cortarse la conexión entre el proxy y el objeto original



1. Que son las propiedades dinámicas de los objetos (de un ejemplo)

En Javascript pre-ECMAScript 6 se lanzo el paquete de métodos donde se agrego la función Objet.

Object.defineProperty permite definir propiedades dinámicas en un objeto, una propiedad dinámica puede devolver un valor diferente según ciertas condiciones (condiciones que son definidas en algún punto del código).

Las propiedades dinámicas tienen como fin agregar propiedades a un objeto en tiempo de ejecución utilizando la sintaxis de corchetes [ ]

