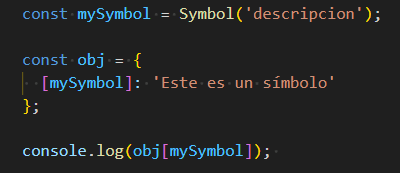
**Symbols**

Los "symbols" son símbolos en el contexto de la programación, específicamente en lenguajes como JavaScript. Un símbolo es un tipo de dato primitivo único e inmutable. Cada símbolo creado es único, lo que significa que dos símbolos nunca serán iguales, incluso si tienen el mismo nombre.

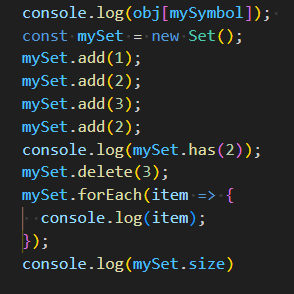
Los símbolos se utilizan principalmente como claves de propiedades de objetos. Son útiles cuando se desea agregar propiedades personalizadas a un objeto sin el riesgo de colisiones con nombres de propiedades existentes o futuras. Por ejemplo:



**Sets**

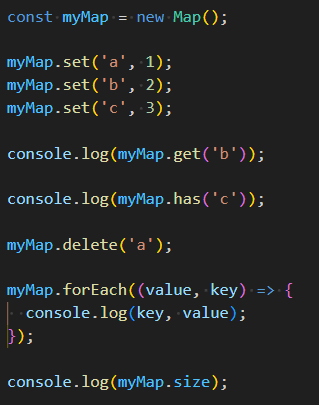
Los conjuntos (sets) son estructuras de datos en programación que representan una colección de elementos únicos, es decir, elementos que no se repiten dentro del conjunto. Los conjuntos se utilizan comúnmente para operaciones como comprobar la pertenencia de un elemento a una colección, eliminar duplicados de una lista y realizar operaciones de conjuntos como unión, intersección y diferencia.

En muchos lenguajes de programación, los conjuntos están implementados como una colección desordenada de elementos únicos. No permiten duplicados y proporcionan métodos para agregar, eliminar y verificar la existencia de elementos en el conjunto.



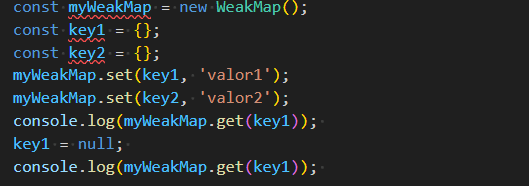
**Maps**

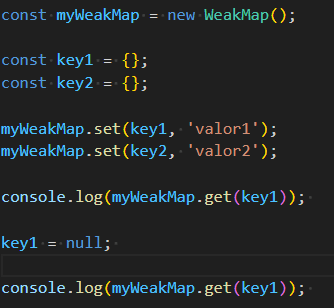
Los Mapas en JavaScript son una estructura de datos que permite almacenar pares clave-valor, donde cada clave puede ser cualquier valor, incluyendo objetos o valores primitivos. Son útiles cuando necesitas asociar datos entre sí de manera eficiente y flexible.



**Weakmaps**

Los WeakMaps en JavaScript son una estructura de datos que te permite asociar pares clave-valor donde las claves son objetos, y las referencias a esos objetos en el mapa son débiles, lo que significa que no impiden que los objetos se eliminen si no hay otras referencias a ellos. Son útiles cuando necesitas asociar datos a objetos de una manera que no afecte el ciclo de vida de esos objetos.

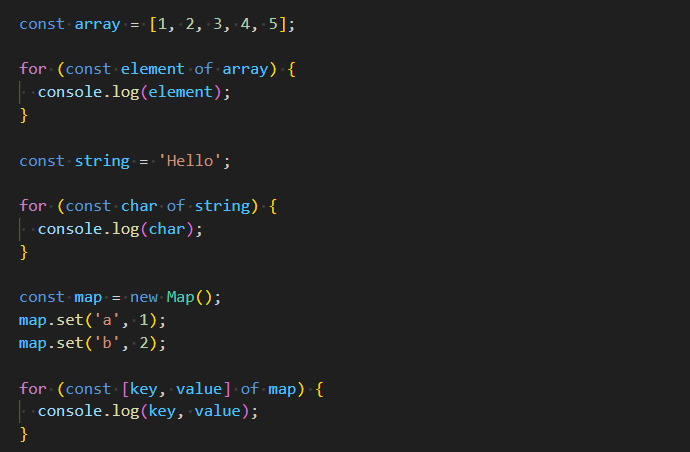




**Iterables**

En JavaScript, un iterable es un objeto que implementa el protocolo de iteración, lo que significa que puede ser recorrido mediante un bucle. Los iterables son una parte fundamental de JavaScript y son utilizados por muchas estructuras de datos y API estándar del lenguaje, como Arrays, Strings, Mapas, Sets, etc.

Los iterables permiten trabajar con secuencias de datos de manera uniforme, lo que significa que puedes utilizar la misma sintaxis de bucle (for...of) para iterar sobre diferentes tipos de datos.

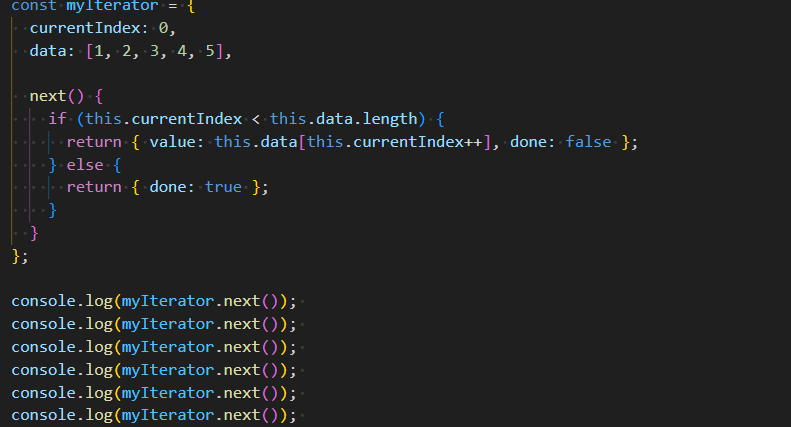


**Iterators**

Los iteradores (iterators) son objetos en JavaScript que proporcionan una forma de acceder a una secuencia de elementos de manera secuencial. Permiten recorrer una colección de elementos uno por uno y realizar alguna operación en cada uno de ellos.

Un iterador debe implementar un método next() que devuelve un objeto con dos propiedades: value, que representa el siguiente elemento de la secuencia, y done, que indica si se ha alcanzado el final de la secuencia.

Los iteradores son ampliamente utilizados en JavaScript para recorrer estructuras de datos como Arrays, Mapas, Sets, etc., y también son utilizados internamente por la sintaxis de los bucles for...of.



**Proxies**

Los Proxies en JavaScript son objetos que permiten interceptar y personalizar operaciones fundamentales en otros objetos (llamados "objetivos"). Estas operaciones incluyen obtener y establecer propiedades, invocar métodos, entre otras. Los Proxies proporcionan una capa adicional de control sobre el comportamiento de los objetos, lo que los hace útiles para la implementación de diversos patrones y técnicas avanzadas.

Los Proxies son útiles cuando necesitas interceptar y modificar el comportamiento predeterminado de los objetos, como implementar validaciones, aplicar lógica adicional antes de acceder a las propiedades, registrar operaciones, entre otros casos de uso.



**Propiedades dinámicas de los objetos**

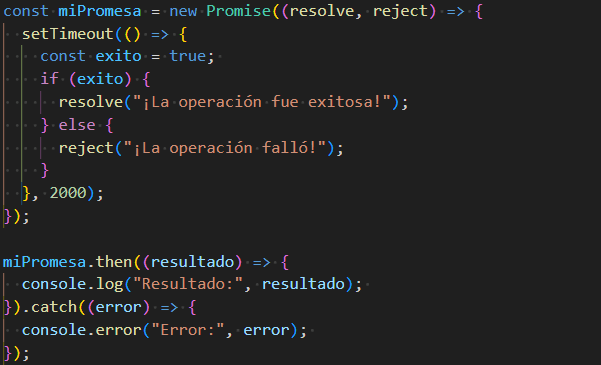
Las propiedades dinámicas de los objetos en JavaScript se refieren a la capacidad de agregar, modificar o eliminar propiedades de un objeto en tiempo de ejecución. Esto significa que puedes manipular la estructura de un objeto durante la ejecución de tu programa.

Esto es útil cuando necesitas manejar datos de manera flexible o cuando deseas construir objetos con propiedades que pueden cambiar dinámicamente en función de la lógica de tu aplicación.



**Promesas**

En JavaScript, las promesas son objetos que representan el resultado eventual (o el fallo) de una operación asíncrona. Se utilizan para manejar operaciones asíncronas de una manera más limpia y fácil de entender que el uso de callbacks anidados. Aquí tienes un ejemplo básico de cómo crear y usar una promesa en JavaScript



**Funciones asincrónicas si wait**

Las funciones asíncronas en JavaScript son funciones que permiten ejecutar código de manera asíncrona, lo que significa que no bloquean la ejecución del programa mientras se espera a que se complete una operación. Esto se logra devolviendo una promesa dentro de la función.

