***TIP****: If you’ve gone through the WeakSets section, then this section should be somewhat of a review. WeakMaps exhibit the same behavior as a WeakSets, except WeakMaps work with key-values pairs instead of individual items.*

**What is a WeakMap?**

A WeakMap is just like a normal Map with a few key differences:

1. a WeakMap can only contain objects as keys,
2. a WeakMap is not iterable which means it can’t be looped and
3. a WeakMap does not have a .clear() method.

You can create a WeakMap just like you would a normal Map, except that you use the WeakMap constructor.

**const** book1 = { title: 'Pride and Prejudice', author: 'Jane Austen' };

**const** book2 = { title: 'The Catcher in the Rye', author: 'J.D. Salinger' };

**const** book3 = { title: 'Gulliver’s Travels', author: 'Jonathan Swift' };

**const** library = **new** WeakMap();

library.set(book1, true);

library.set(book2, false);

library.set(book3, true);

console.log(library);

*WeakMap {Object {title: 'Pride and Prejudice', author: 'Jane Austen'} => true, Object {title: 'The Catcher in the Rye', author: 'J.D. Salinger'} => false, Object {title: 'Gulliver’s Travels', author: 'Jonathan Swift'} => true}*

…but if you try to add something other than an object as a key, you’ll get an error!

library.set('The Grapes of Wrath', false);

*Uncaught TypeError: Invalid value used as weak map key(…)*

This is expected behavior because WeakMap can only contain objects as keys. Again, similar to WeakSets, WeakMaps leverage garbage collection for easier use and maintainability.

**Garbage Collection**

In JavaScript, memory is allocated when new values are created and is "automatically" freed up when those values are no longer needed. This process of freeing up memory after it is no longer needed is what is known as *garbage collection*.

WeakMaps take advantage of this by exclusively working with objects as keys. If you set an object to null, then you’re essentially deleting the object. And when JavaScript’s garbage collector runs, the memory that object previously occupied will be freed up to be used later in your program.

book1 = null;

console.log(library);

*WeakMap {Object {title: 'The Catcher in the Rye', author: 'J.D. Salinger'} => false, Object {title: 'Gulliver’s Travels', author: 'Jonathan Swift'} => true}*

Replay

Mute

Loaded: 100.00%

Remaining Time -0:00

1x

Playback Rate

Picture-in-Picture

Fullscreen

***Erratum****: In the previous video animation, const is use to declare the book1 variable. book1 is then reassigned to null -- however, this wouldn't be possible because the variable identifier cannot be reassigned (i.e., you'll see Uncaught TypeError: Assignment to constant variable.). Instead, we recommend using let rather than const to declare book1, which allows for reassignment.*

What makes this so useful is you don’t have to worry about deleting keys that are referencing deleted objects in your WeakMaps, JavaScript does it for you! When an object is deleted, the object key will also be deleted from the WeakMap when garbage collection runs. This makes WeakMaps useful in situations where you want an efficient, lightweight solution for creating groupings of objects with metadata.

The point in time when garbage collection happens is dependent on a lot of different factors. Check out [MDN’s documentation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management#Garbage_collection) to learn more about the algorithms used to handle garbage collection in JavaScript.