**What is a WeakSet?**

A WeakSet is just like a normal Set with a few key differences:

1. a WeakSet can only contain objects
2. a WeakSet is not iterable which means it can’t be looped over
3. a WeakSet does not have a .clear() method

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

**let** student1 = { name: 'James', age: 26, gender: 'male' };

**let** student2 = { name: 'Julia', age: 27, gender: 'female' };

**let** student3 = { name: 'Richard', age: 31, gender: 'male' };

**const** roster = **new** WeakSet([student1, student2, student3]);

console.log(roster);

*WeakSet {Object {name: 'Julia', age: 27, gender: 'female'}, Object {name: 'Richard', age: 31, gender: 'male'}, Object {name: 'James', age: 26, gender: 'male'}}*

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

roster.add('Amanda');

*Uncaught TypeError: Invalid value used in weak set(…)*

This is expected behavior because WeakSets can only contain objects. But why should it only contain objects? Why would you even use a WeakSet if normal Sets can contain objects and other types of data? Well, the answer to that question has more to do with why WeakSets do not have a .clear() method...

**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**.

WeakSets take advantage of this by exclusively working with objects. 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.

student3 = null;

console.log(roster);

*WeakSet {Object {name: 'Julia', age: 27, gender: 'female'}, Object {name: 'James', age: 26, gender: 'male'}}*

Replay

Mute

Loaded: 100.00%

Remaining Time -0:00

1x

Playback Rate

Picture-in-Picture

Fullscreen

What makes this so useful is you don’t have to worry about deleting references to deleted objects in your WeakSets, JavaScript does it for you! When an object is deleted, the object will also be deleted from the WeakSet when garbage collection runs. This makes WeakSets useful in situations where you want an efficient, lightweight solution for creating groups of objects.

The point in time when garbage collection happens depends 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.