```
// 15. Create a cheat sheet for Map, Set, WeakSet and WeakMap in JS.

//--------------------------------------Map -----------------------------------------------
// You would start by creating a map:
const demouser = new Map();
demouser.set(u1, 'User');
demouser.set(u2, 'User');
demouser.set(u3, 'Admin');

// The set() method is also chainable, which can save some typing:
demouser
 .set(u1, 'User')
 .set(u2, 'User')
 .set(u3, 'Admin');

// You can also pass an array of arrays to the constructor:
const demouser = new Map([
 [u1, 'User'],
 [u2, 'User'],
 [u3, 'Admin'],
]);

// Now if we want to determine what role u2 has, you can use the get() method:
demouser.get(u2);

// has() method to determine if a map contains a given key:
demouser.has(u1); // true
// The size property will return the number of entries in the map:
demouser.size; // 3

// Use the keys() method to get the keys in a map, values() to return the values, and
for(let u of demouser.keys())
 demouser.log(u.name);

for(let r of demouser.values())
console.log(r);
// entries() to get the entries as arrays where the first element is the key and the sec-
// ond is the value.
for(let [u, r] of demouser.entries())
 console.log(`${u.name}: ${r}`);
```

```
//--------------------------------------Set ---------------------------------------------
//  Set could be quite useful when you don’t want to add the same values
const set = new Set();
set.add(1);
set.add('piu');
set.add(BigInt(10));
// Set(4) {1, "john", 10n}
// Like Map, Set also prevents us from adding the same value.
set.add(5);
// Set(1) {5}
set.add(5);
// Set(1) {5}
// Set is an iterable object, you can use a for-of or forEach
for (const val of set) {
   console.dir(val);
  }
set.forEach(val => console.dir(val));
//delete
set.delete(5);
```

```
//--------------------------------------WeakSet ---------------------------------------------
//  Like WeakMap, WeakSet also loses the access link to inner data if they’re
garbage-collected.
// Not an iterable object — can’t loop over
// Can’t access the data if the referencing data is garbage-collected
// Less supportive methods
let sub = { major: "math" };

const weakSet = new WeakSet();
```

```
weakSet.add(sub);
// WeakSet {{...}}
sub = null;
/* sub is garbage-collected */




 //---------------------------------------WeakMap ------------------------------------------------
 /**A WeakMap is identical to Map except:
 • Its keys must be objects.
 • Keys in a WeakMap can be garbage-collected.
 • A WeakMap cannot be iterated or cleared.
 **/

 const secrets = new WeakMap();
 /**The methods you can use with WeakMap are as follows.
    --delete
    --get
    --has
    --set
   - WeakMap doesn’t support the methods for iterating the object.
**/
secrets.set(my, 'piu');
my = null;
```