Maps and Sets

Data structures in JavaScript

- Data structures are formats for efficiently collecting or storing data
- So far we've seen Arrays and Objects
- ES2015 introduces two new ones, Maps and Sets

Maps

- Also called "hash maps" in other languages
- Until ES2015 - objects were replacements for maps
- Similar to objects, except the keys can be ANY data type!
- Created using the new keyword

What it looks like

```
let firstMap = new Map();

firstMap.set(1, 'Ash');
firstMap.set(false, 'a boolean');
firstMap.set('nice', 'a string');
firstMap.delete('nice'); // true
firstMap.size; // 2
```

Keys can be any type!

```
let arrayKey = [];
firstMap.set(arrayKey, [1,2,3,4,5]);

let objectKey = {};
firstMap.set(objectKey, {a:1});

firstMap.get(1); // 'Ash'
firstMap.get(false); // 'a boolean'
firstMap.get(arrayKey); // [1,2,3,4,5]
firstMap.get(objectKey); // {a:1}
```

We can easily iterate over the map!

```
firstMap.forEach(v => console.log(v));

// Ash
// a boolean
// [1,2,3,4,5]
// {a:1}
```

Maps also provide:

- .keys() to iterate over all the keys
- .values() to iterate over all the values
- .entries() to iterate over all the [key,value] pairs
- a Symbol.iterator which means we can use a for…of loop to iterate over the keys, values or both!

Here's what it looks like to access everything in a map with .entries() and destructuring!

```
let m = new Map([
  [1, "Ayisha"],
  [2, "Shani"],
  [3, "Michelle"],
])

for(let [key,value] of m.entries()){
    console.log(key, value);
}

// 1 "Ayisha"
// 2 "Shani"
// 3 "Michelle"
```

Why use maps?

- Finding the size is easy - no more loops or Object.keys()
- The keys can be any data type!
- You can accidentally overwrite keys on the Object.prototype in an object you make - maps do not have that issue
- Iterating over keys and values in a map is quite easy as well

- If you need to look up keys dynamically (they are not hard coded strings)
- If you need keys that are not strings!
- If you are frequently adding and removing key/value pairs
- Are key-value pairs frequently added or removed?
- If you are operating on multiple keys at a time

Sets

- All values in a set are unique
- Any type of value can exist in a set
- Created using the new keyword
- Exist in quite a few other languages, ES2015 finally brings them to JavaScript

Creating Sets

- To make a new Set, we call ***new Set()***
- When making a new Set, you can also pass in an iterable object.

```
const hashTags = new Set(["#selfie", "#nofilter"])
```

Adding to Sets

There is only a single method to add items to a set: ***add()***

```
const annoyingHashTags = new Set();
annoyingHashTags.add("#YOLO");
annoyingHashTags.add("#Blessed")
annoyingHashTags.add("#YOLO"); // will not be added!
```

size

Use the ***size*** property to determine the number of values stored in a Set:

```
const annoyingHashTags = new Set();
annoyingHashTags.add("#YOLO");
annoyingHashTags.add("#Blessed")

annoyingHashTags.size //2
```

Checking for an element in a set

- Sets do not support random access,
- but we are able to check if a set contains a given value using ***has()***

```
const annoyingHashTags = new Set();
annoyingHashTags.add("#YOLO");
annoyingHashTags.add("#Blessed");

annoyingHashTags.has("#YOLO"); //true
annoyingHashTags.has("#Selfie"); //false
```

Removing values in a set

To remove a single value from a set, use ***delete()***

```
const annoyingHashTags = new Set();
annoyingHashTags.add("#YOLO");
annoyingHashTags.add("#Blessed");

annoyingHashTags.has("#YOLO"); //true
annoyingHashTags.delete("#YOLO");
annoyingHashTags.has("#YOLO"); //false
```

We can also use clear() to empty a set of all values:

```
annoyingHashTags.clear();
```

Iterating over a set

- Sets are iterable objects, so we can use them with for…of loops or the spread operator.
- Values in a set are ordered by insertion order. Here’s one example of looping over a Set:

```
const annoyingHashTags = new Set();
annoyingHashTags.add("#Selfie");
annoyingHashTags.add("#Blessed");
annoyingHashTags.add("#NoFilter");

for(let val of annoyingHashTags) {
   console.log("Please don't use", val);
}
```

When would you use sets?

- Removing duplicate values
- Uniqueness required
- Efficiently checking if an item is in a collection (much better than arrays)