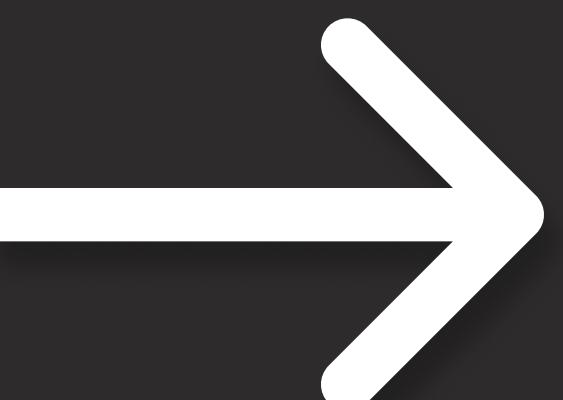


JS

Built-In Data Structures In JavaScript



@oliverjumpertz



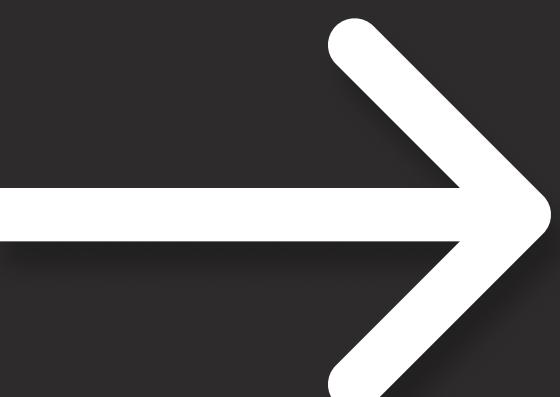
Array

1

Arrays are the **most primitive** data structure in JavaScript.

You usually **use them a lot**.

Arrays are **built-in** and **easy-to-use**.



Array Usage

2



```
// Instantiate an Array.  
const array = [1, 2, 3, 4, 5];
```



```
// You can append elements to  
// the end of the Array.  
array.push(3);
```



```
// Arrays have monad methods.  
// Useful to transform, filter, etc  
array.map((element => element + 1));
```



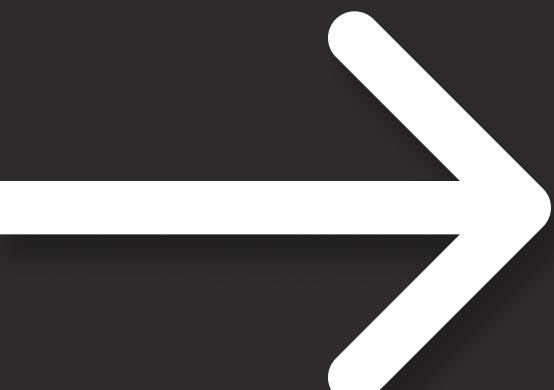
```
// Access an Array element.  
// Indices are zero-based.  
const firstElement = array[0];
```



```
// Overwrite an Array element.  
// Indices are zero-based.  
array[0] = 5;
```



```
// You can also copy  
// parts of an Array.  
array.slice(0, 2);
```

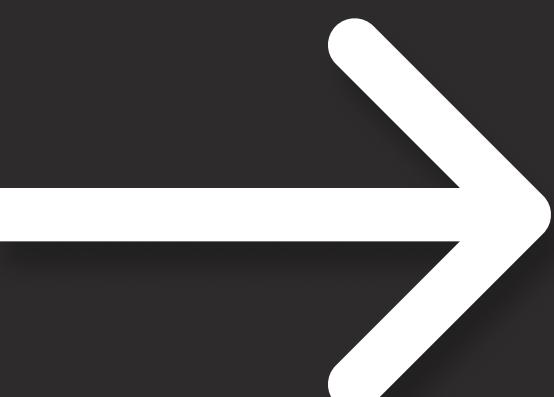


Stack/Queue

3

JavaScript's Arrays' are **more** than only that.

They can also be used as
Stacks and **Queues**
thanks to their methods.



Stack/Queue Usage

4



```
// You still instantiate an array  
// if you want to use it as a  
// stack or queue.  
const array = [1, 2, 3];
```



```
// Push appends elements to  
// the end of the array.  
// That's the same as putting  
// elements on top of a Stack.  
array.push(5);
```



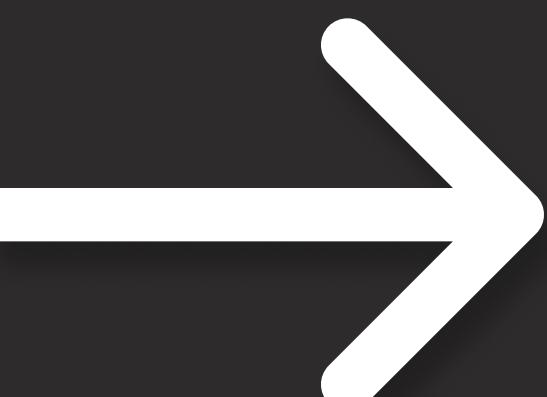
```
// Pop removes the element at  
// the top of the stack and  
// returns it.  
const topElement = array.pop();
```



```
// Unshift adds elements to  
// the beginning of the array.  
// That's the same as putting  
// items into a queue.  
array.unshift(5);
```



```
// When unshift puts elements  
// into the queue, pop removes  
// them as the end of the array  
// is the beginning of the queue.  
const firstElement = array.pop();
```



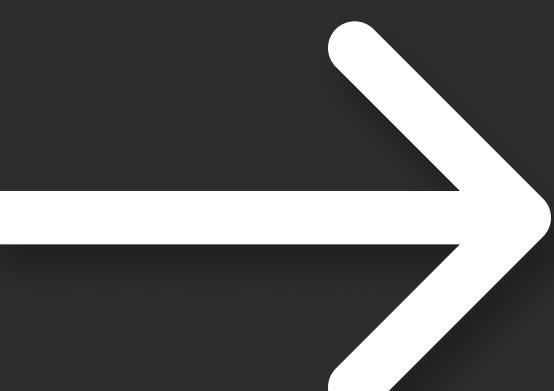
Set

5

A **Set** is a data structure that only contains **unique elements**.

You need to be careful in JavaScript, though.

Unique means: **By reference**, or in other words:
Everything where $x === y$ is true.



6

Set Usage



```
// A Set has a constructor.  
const set = new Set();
```



```
// Has checks whether the Set  
// contains a specific entry.  
// It can only check for  
// identity, though.  
set.has(1);
```



```
// Delete removes an element  
// from the Set  
set.delete(1);
```



```
// You can also create a Set  
// from an Array.  
const set = new Set([1, 2, 3]);
```



```
// Add appends elements to  
// the Set.  
set.add(6);
```



```
// A Set can also be cleared.  
set.clear();
```



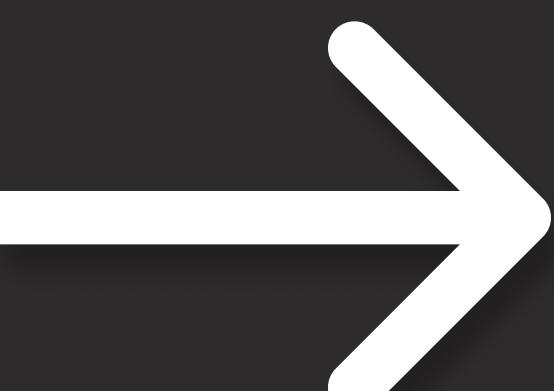
Map

7

A **Map** is a data structure that contains **key-value** pairs.

Its keys have the same **restriction** as a Set's values:
They are only tested for **reference identity**.

This means you cannot reliably use objects as keys.



Map Usage

8



```
// A Map has a constructor.  
const map = new Map();
```



```
// Get retrieves values by  
// their key.  
map.get(key);
```



```
// Has tests whether the Map  
// contains a value with the  
// given key.  
map.has(key);
```



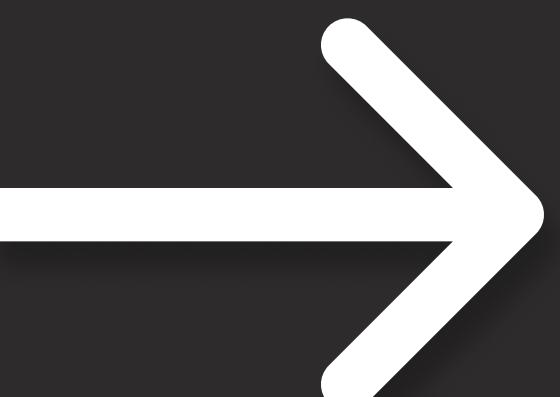
```
// You can also pass a  
// 2-dimensional Array  
// to the constructor.  
const map = new Map([  
  [key, value],  
  [otherKey, otherValue],  
]);
```



```
// Set adds a key-value pair  
// to the Map.  
// It also overwrites  
// existing values.  
map.set(key, value);
```



```
// Delete removes a key-value  
// pair from the Map if present.  
map.delete(key);
```





Found this
useful?

Leave a Like
and Follow
me.



@oliverjumpertz