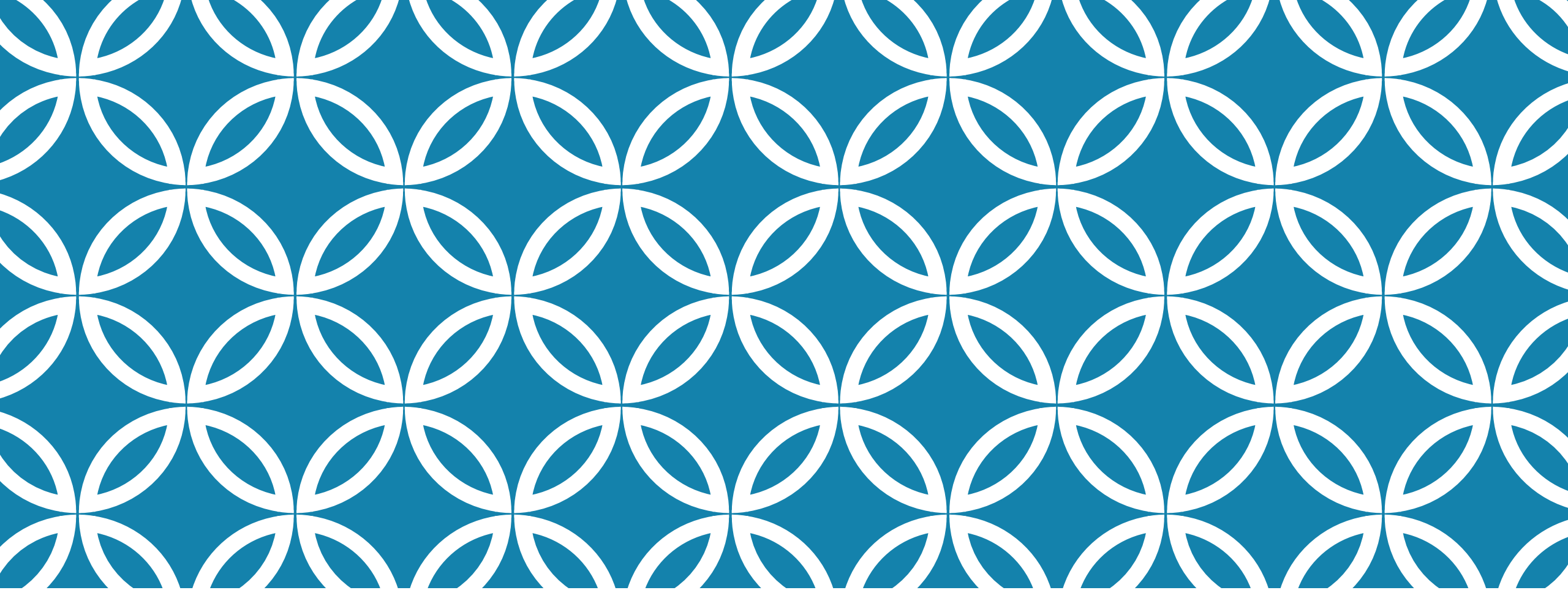


ARRAYS

Andrew Sheehan

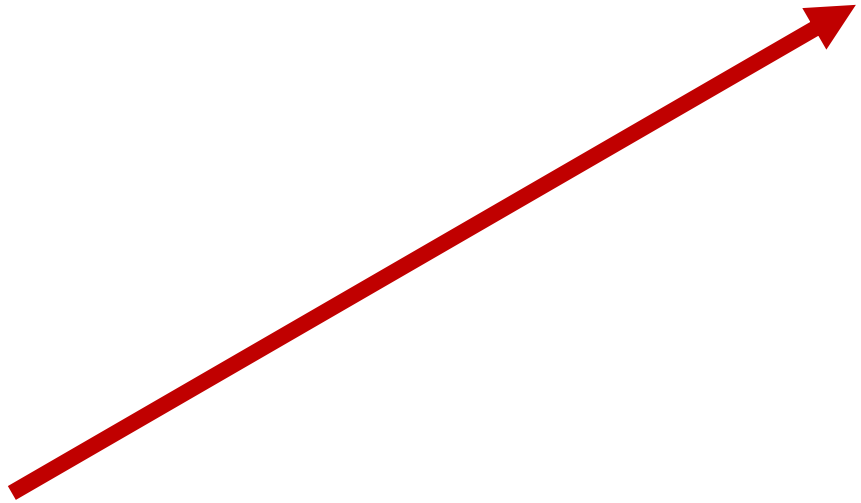
Boston University
Metropolitan College

DATA STRUCTURES

Objects that allow us to group or store ‘things’ (other data types)

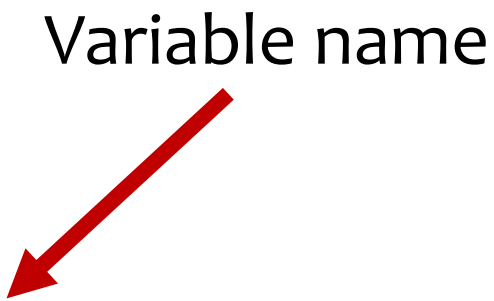
SYNTAX

```
const lastNames = [  ];
```

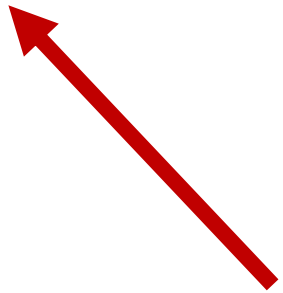


Brackets: indicates an array. This is important to note.

ARRAY EXAMPLE




```
const lastNames = [‘Sheehan’, ‘Ali’, ‘Torreto’];
```



Collection of string values

ITERATION
FOREACH

```
const lastNames = [ ‘Sheehan’, ‘Smith’ ];

lastNames.forEach( (item, index) => {
    /* your logic goes in here… */
});
```

Output: Sheehan, 0

Smith, 1

FOREACH()
ANOTHER WAY USING IT

```
1  var a = ['a', 'b', 'c'];
2
3  a.forEach(function(element) {
4      console.log(element);
5  });
```

ARRAY.MAP()

Creates a new array, based on what you need to do with the original array

```
1  var numbers = [1, 5, 10, 15];
2  var doubles = numbers.map(function(x) {
3     return x * 2;
4  });
5  // doubles is now [2, 10, 20, 30]
6  // numbers is still [1, 5, 10, 15]
7
8  var numbers = [1, 4, 9];
9  var roots = numbers.map(Math.sqrt);
10 // roots is now [1, 2, 3]
11 // numbers is still [1, 4, 9]
```

MAP
ANOTHER EXAMPLE

```
1 | var numbers = [1, 4, 9];
2 | var roots = numbers.map(Math.sqrt);
3 | // roots is now [1, 2, 3]
4 | // numbers is still [1, 4, 9]
```

MAP
USING ARROW FUNCTIONS

```
8       let numbers = [1,2,4,8,16];
9       let doubles = numbers.map((value, index, arr) => {
10          return index === 0 ? -1 : value * 2;
11      });
12      doubles.forEach((value, index) => {
13          if ( index === 0 ) {
14              console.info("zeroth doesn't matter.");
15          } else {
16              console.info("doubled is: " + value);
17          }
18      });
```

REDUCE()

A function that reduces to a single value.

```
1  const array1 = [1, 2, 3, 4];
2  const reducer = (accumulator, currentValue) => accumulator + currentValue
3
4  // 1 + 2 + 3 + 4
5  console.log(array1.reduce(reducer));
6  // expected output: 10
7
8  // 5 + 1 + 2 + 3 + 4
9  console.log(array1.reduce(reducer, 5));
10 // expected output: 15
```

REDUCE EXAMPLE

```
var numbers = [65, 44, 12, 4];

function getSum(total, num) {
    return total + num;
}
function myFunction(item) {
    document.getElementById("demo").innerHTML = numbers.reduce(getSum);
}
```

```
const euros = [29.76, 41.85, 46.5];

const sum = euros.reduce((total, amount) => total + amount);

sum // 118.11
```

REDUCE
W/ ARROW FUNCTIONS

```
1 | [0, 1, 2, 3, 4].reduce( (prev, curr) => prev + curr );
```

Your **reducer** function's returned value is assigned to the accumulator, whose value is remembered across each iteration throughout the array and ultimately becomes the final, single resulting value.

ADDING VALUE

```
var fruits = ['Apple', 'Banana'];
var newLength = fruits.push('Orange');
// ["Apple", "Banana", "Orange"]
```

KEYS() IN THE ARRAY

KEYS()

keys() will return an iterator that contains all the keys in the array.

```
1  var arr = ['a', 'b', 'c'];
2  console.log(Object.keys(arr)); // console: ['0', '1', '2']
3
4  // array like object
5  var obj = { 0: 'a', 1: 'b', 2: 'c' };
6  console.log(Object.keys(obj)); // console: ['0', '1', '2']
7
8  // array like object with random key ordering
9  var anObj = { 100: 'a', 2: 'b', 7: 'c' };
10 console.log(Object.keys(anObj)); // ['2', '7', '100']
11
12 // getFoo is property which isn't enumerable
13 var myObj = Object.create({}, {
14   getFoo: {
15     value: function () { return this.foo; }
16   }
17 });
18 myObj.foo = 1;
19 console.log(Object.keys(myObj)); // console: ['foo']
```

undefined is returned when no match

VALUES() IN THE ARRAY

```
const array1 = ['a', 'b', 'c'];
const iterator = array1.values();

for (const value of iterator) {
  console.log(value); // expected output: "a" "b" "c"
}

```

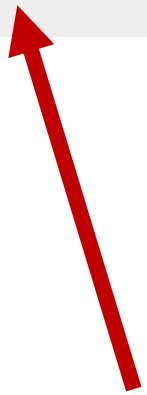
```
const object1 = {
  a: 'somestring',
  b: 42,
  c: false
};

console.log(Object.values(object1));
// expected output: Array ["somestring", 42, false]

```

REMOVE VALUE

```
var fruits = ['Apple', 'Banana'];
var last = fruits.pop(); // remove Orange (from the end)
// ["Apple", "Banana"];
```

REMOVE FROM THE FRONT

```
var first = fruits.shift();
```



Returns the removed item to you.

PASSED BY REFERENCE

Modifying an Array in a function
does alter its original declaration

```
var myArray = [ 'zero', 'one', 'two', 'three', 'four', 'five' ];

function passedByReference(arrayObject) {
   arrayObject[1] = 'dog';
}

passedByReference(myArray);

document.writeln(myArray[1]); // output: dog, not one
```

JOINING
WITH A TOKEN

```
1 var a = ['Wind', 'Rain', 'Fire'];
2 a.join();     // 'Wind,Rain,Fire'
3 a.join('-'); // 'Wind-Rain-Fire'
```

FIND()

```
1  function isBigEnough(element) {
2    return element >= 15;
3  }
4
5  [12, 5, 8, 130, 44].find(isBigEnough); // 130
```

Undefined when there is no match.

CONCATENATION

```
var parents = ["John", "Mary"];
var children = ["Susan", "Michael"];

var family = parents.concat(children);
```

USING OF()

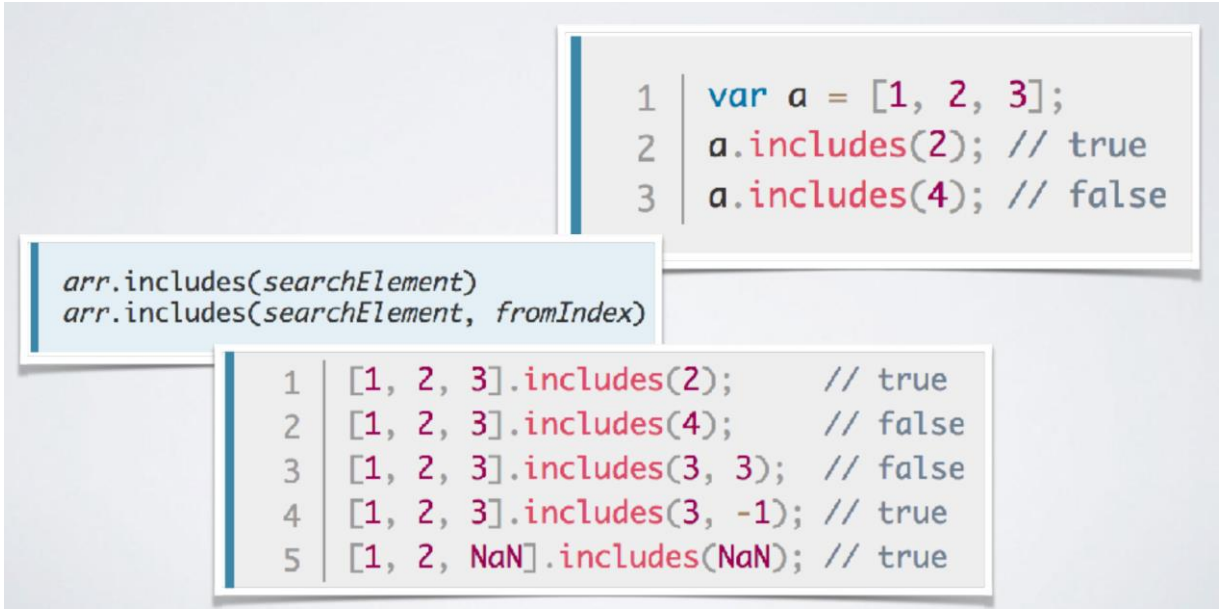
OF()

The of() method creates a new array, with a variable number of args, regardless of type.

```
1  Array.of(7);          // [7]
2  Array.of(1, 2, 3); // [1, 2, 3]
3
4  Array(7);             // [ , , , , , , , ]
5  Array(1, 2, 3);       // [1, 2, 3]
```

```
1  Array.of(1);          // [1]
2  Array.of(1, 2, 3);    // [1, 2, 3]
3  Array.of(undefined); // [undefined]
```

INCLUDES SEARCHING



arrayA.sort()

Sorts the array (numerically or alpha)

arrayB.join(byYourDelimiter)

creates string, separated by your delimiter

arrayC.push(object)

places it on the front of array

arrayD.toString()

Array elements output as string

arrayE.reverse(object)

returns new array in reversed order.

arrayF.slice(from, [to])

returns new array