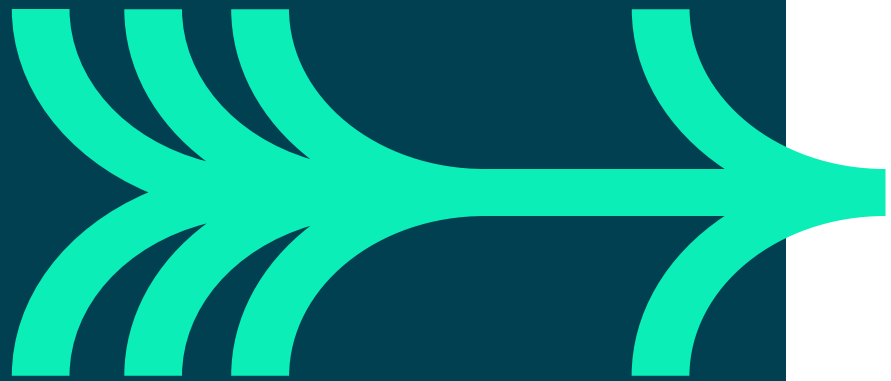
POWERING POTENTIAL

Arrays

→ JavaScript Fundamentals

INTRODUCTION

- Arrays
  - → What are arrays?
  - → Creating arrays
  - → Accessing arrays
  - → Array methods



Creating arrays

- Arrays hold a set of related data, e.g. students in a class
- The default approach is accessed by a numeric index

a is created with no data

c is a 3 element array of string

```
let a = Array();
let b = Array(10);
let c = Array("Tom", "Dick", "Harry");
let d = [1,2,3];
```

b is a 10 element array of undefined

d is shorthand for an array

Creating arrays

Arrays in JavaScript have some idiosyncrasies:

- They can be resized at any time
- They index at 0
  - So **`Array(3)`** would have elements with indexes 0, 1, and 2
- They can be *sparsely* filled
  - Unassigned parts of an array are **`undefined`**
- They can be created in shorthand using just square brackets

Accessing arrays

- Arrays are accessed with a square bracket notation

Access an array via its index

```
let classRoom = new Array(5);
classRoom[0] = "Dave";
classRoom[4] = "Laurence";
```

Elements 1 through 3 are not yet set

- Arrays have a length property that is useful in loops

```
for (let i = 0; i < classRoom.length; i++) {
        console.log(classRoom[i]);
}
```

i has 1 added to it on each iteration of the loop

Array object methods

- Array objects have methods
- **`reverse()`**
- **`join([separator])`**
  Joins all the elements of the array into one string, using the supplied separator or a comma
- **`sort([sort function])`**
  Sorts the array using string comparisons by default
  Optional sort function compares two values and returns sort order

```
let fruit = ['Apples', 'Pears', 'Bananas', 'Oranges'];
let fruitString = fruit.join("---");

// Apples---Pears---Bananas---Oranges
console.log(fruitString);
```

Pop and push array methods

- The **`push()`** method
  - Adds a new element to the end of the array
  - Array's length property is increased by one
  - This method returns the new length of the array

```
let fruit = ['Apples', 'Pears', 'Bananas', 'Oranges'];
console.log(fruit.push('Lemons')); //5

// ['Apples', 'Pears', 'Bananas', 'Oranges', 'Lemons']
console.log(fruit);
```

Pop and push array methods

- The **`pop()`** method
  - Removes the last element from the end of the array
  - The array's length property is decreased by one
  - This method returns the array element that was removed

```
let fruit = ['Apples', 'Pears', 'Bananas', 'Oranges'];
console.log(fruit.pop()); //Oranges

//['Apples', 'Pears', 'Bananas']
console.log(fruit);
```

Shift and unshift array methods

- The **`unshift()`** method
  - Adds a new element to the beginning of the array
  - Array's length property is increased by one
  - This method returns the new length of the array

```
let fruit = ['Apples', 'Pears', 'Bananas', 'Oranges'];
console.log(fruit.unshift('Kiwis')); //5

//['Kiwis','Apples', 'Pears', 'Bananas', 'Oranges']
console.log(fruit);
```

Shift and unshift array methods

- The **`shift()`** method
  - removes the first element from the beginning of the array
  - Array's length property is decreased by one
  - This method returns the array element that was removed

```
let fruit = ['Apples', 'Pears', 'Bananas', 'Oranges'];
console.log(fruit.shift()); //Apples

//['Pears', 'Bananas', 'Oranges']
console.log(fruit);
```

New Methods in ES2015

- **`Array.from()`** creates a real array out of array-like objects

```
let formElements = document.querySelectorAll('input, select, textarea');
formElements = Array.from(formElements);
formElements.push(anotherElement); //works fine!
```

- **`Array.prototype.find()`** returns the first element for which the callback returns true

  A callback is a function passed to another function – the one shown below is an anonymous function:

```
[`Chris`,`Bruford`,22].find(function(n) { return n === `Bruford`}); // Bruford
```

  This is an instance where an arrow function could be used to clean the code:

```
[`Chris`,`Bruford`,22].find( n => n === `Bruford`); // Bruford
```

New Methods in ES2015

- Similarly **`findIndex()`** returns the index of the first matching element

```
[`Chris`,`Bruford`,22].findIndex( n => n === `Bruford`}); // 1
```

- **`fill()`** overrides the specified elements

```
[`Chris`,`Bruford`,22,true].fill(null);       // [null,null,null,null]
[`Chris`,`Bruford`,22,true].fill(null,1,2);   // [`Chris`,null,null,true]
```

New Methods in ES2015

- `.entries()`, `.keys()` & `.values()` each return a sequence of values via an iterator:

```
let arrayEntries = [`Chris`,`Bruford`,22,true].entries();
console.log(arrayEntries.next().value);      // [0,`Chris`]
console.log(arrayEntries.next().value);      // [1,`Bruford`]
console.log(arrayEntries.next().value);      // [2,22]
```

```
let arrayKeys = [`Chris`,`Bruford`,22,true].keys();
console.log(arrayKeys.next().value);         // 0
console.log(arrayKeys.next().value);         // 1
console.log(arrayKeys.next().value);         // 2
```

```
let arrayValues = [`Chris`,`Bruford`,22,true].values();
console.log(arrayValues.next().value);       // `Chris`
console.log(arrayValues.next().value);       // `Bruford`
console.log(arrayValues.next().value);       // 22
```

for...of loop

- The for-of loop is used for iterating over **iterable** objects (more on that later!)
- For an array it means we can loop through the array, returning each element in turn

```
//will print 1 then 2 then 3
let myArray = [1,2,3,4];
for (el of myArray) {
    if (el === 3) break;
    console.log(el);
}
```

- We could also loop through any of the iterables returned by the methods **`.entries()`**, **`.values()`** and **`.keys()`**

QuickLab 16 - Arrays

- Creating and Managing Arrays