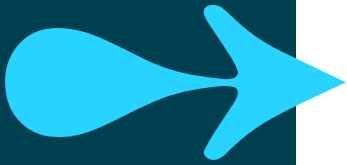




## INTRODUCTION



### Arrays

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

## QA 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

3

As with the rest of JavaScript, the elements in arrays are loosely-typed. This means that each separate element of an array can contain a different type. This is useful and dangerous at once and brings us back to the consideration that type matters.

Array indices in JavaScript start at 0. Arrays have a `length` property equal to the length of the array; note that an array of length of 6 will have elements 0 to 5.

JavaScript arrays are *sparse* – this means that there can be holes in an array. The example above shows an array that initially has values assigned to elements number 0 and 5. This results in an array that contains only two elements. Elements numbered 1 to 4 exist but, if referenced, will yield the special value `undefined`.

Note that once a value has been assigned to an element of an array, that array element is now defined, and cannot be removed. Assigning `null` to it changes the value but does not remove the array element. In fact, simply *referring* to the element is enough to create it.

Arrays in JavaScript will grow dynamically from its initial size. It is also possible to shrink an array by assigning a smaller value to the `length` property than it currently holds. Do not assume that elements with indexes greater than or equal to the new value exist.

## 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 *sparingly* filled
    - Unassigned parts of an array are **undefined**
  - They can be created in short hand using just square brackets

4

As with the rest of JavaScript, the elements in arrays are loosely-typed. This means that each separate element of an array can contain a different type. This is useful and dangerous at once and brings us back to the consideration that type matters.

Array indices in JavaScript start at 0. Arrays have a `length` property equal to the length of the array; note that an array of length of 6 will have elements 0 to 5.

JavaScript arrays are *sparse* – this means that there can be holes in an array. The example above shows an array that initially has values assigned to elements number 0 and 5. This results in an array that contains only two elements. Elements numbered 1 to 4 exist but, if referenced, will yield the special value **undefined**.

Note that once a value has been assigned to an element of an array, that array element is now defined, and cannot be removed. Assigning `null` to it changes the value but does not remove the array element. In fact, simply *referring* to the element is enough to create it.

Arrays in JavaScript will grow dynamically from its initial size. It is also possible to shrink an array by assigning a smaller value to the `length` property than it currently holds. Do not assume that elements with indexes greater than or equal to the new value exist.

## QA 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

5

Arrays allow us to store a related set of data. Arrays store data in a list of elements; we access a particular elements by specifying the name of the array and the element index within square brackets, e.g.

```
myArray[0];
```

The first element of the array is always accessed through [0].

In the above example, a 5 element array is created and elements 1 through 3 are not set. In this situation, they will have a value of **undefined**. If you remember what we have learnt from the module on types, this makes perfect sense. They have been created but not initialised.

Arrays also have a length property. This will always reflect the exact number of elements an array contains. As you can see, this is immensely useful as it allows us to create loops that will always run as many times as is needed.

## 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);
```

6

Array objects have various methods:

- **reverse** reverses the order of the elements in the array, so that the last element in the array becomes the first, and so on. This method operates directly on the array itself, rather than returning a new array
- **join** returns a string formed by joining together all the elements in the array using the supplied separator. If no separator is supplied, a comma is used. The separator may be any string (including an empty one). Undefined array elements are represented by a null string, meaning that two or more separators will appear next to each other
- **Sort** sorts the array. If no function argument is supplied, the array elements are temporarily converted to strings and sorted in standard dictionary order. To make the sort generic, it is possible to supply a function as an argument. This function will be called by the sort routine as necessary to compare two values in the array. The function should have the form: `compare(a, b)` and should return a value less than 0 if a should be sorted less than b, 0 if they are equal, and greater than 0 if a should be sorted greater than b

## 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);
```

7

The JavaScript `Array.push()` method adds a new element to the end of the array. When doing so, the array's length property increases by one. After adding the new element to the end of the array, this method returns the new length of the array. The JavaScript `Array.pop()` method removes the last element from the end of the array. When doing so, the array's length property decreases by one. After removing the last element from the end of the array, this method returns the array element that was removed.



## 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);
```

The JavaScript `Array.push()` method adds a new element to the end of the array. When doing so, the array's length property increases by one. After adding the new element to the end of the array, this method returns the new length of the array. The JavaScript `Array.pop()` method removes the last element from the end of the array. When doing so, the array's length property decreases by one. After removing the last element from the end of the array, this method returns the array element that was removed.



## 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);
```

9

The JavaScript `Array.shift()` method removes the first element from the beginning of the array. When doing so, the array's length property decreases by one. After removing the first element from the beginning of the array, this method returns the array element that was removed.

The JavaScript `Array.unshift()` method adds a new element to the beginning of the array. When doing so, the array's length property increases by one. After adding the new element to the beginning of the array, this method returns the new length of the array.

## 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);
```

10

The JavaScript `Array.shift()` method removes the first element from the beginning of the array. When doing so, the array's length property decreases by one. After removing the first element from the beginning of the array, this method returns the array element that was removed.

The JavaScript `Array.unshift()` method adds a new element to the beginning of the array. When doing so, the array's length property increases by one. After adding the new element to the beginning of the array, this method returns the new length of the array.

TRAINER NOTE: Functions up to and including Arrow Functions is relevant here before continuing

## 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

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

```
[`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 6 - Arrays

- Creating and Managing arrays