ARRAYS (PART 1)

What have we learned about arrays so far?

1. We create an array using square brackets:

```
const myArray = []; // an empty array
```

2. One array can hold many items of data (like an object)
3. We use *indexes* to access data from an array
4. Indexes begin from *0*

```
myArray[0]; // returns a value
```

We also saw how we can find out the size of an array using the *length* property

We also saw how we can find out the size of an array using the *length* property

```
1 const colours = [ "red", "green", "orange", "yellow" ];
2
3 console.log(colours.length); // 4
```

We also saw how we can find out the size of an array using the *length* property

```
1 const colours = [ "red", "green", "orange", "yellow" ];
2
3 console.log(colours.length); // 4
```

We used this property in loops, so we know how many times to loop through the array

We saw in a previous presentation how *strings* have methods we can use, for example:

```
1 const name = "Luigi";
2
3 const result = name.toUpperCase(); // "LUIGI"
```

Arrays *also* have their own methods

- concat()
- copyWithin()
- entries()
- every()
- filter()
- find()
- findIndex()
- forEach()

- indexOf()
- join()
- keys()
- lastIndexOf()
- pop()
- push()
- reduce()
- reduceRight()

- concat()
- copyWithin()
- entries()
- every()
- filter()
- find()
- findIndex()
- forEach()

- indexOf()
- join()
- keys()
- lastIndexOf()
- pop()
- push()
- reduce()
- reduceRight()

... and many more

Can you remember them all?

Can you remember them all?

Of course not! Neither can I! 🤯

Can you remember them all?

Of course not! Neither can I! 🤯

We will only learn a few 🙂

- push()
- pop()
- forEach()
- filter()

push() allows us to *ADD* items to the *END* of an array

```
1 const fruit = [ "apple" ];
2
3 fruit.push("pineapple");
4
5 console.log(fruit); // [ "apple", "pineapple" ]
```

pop() allows us to *REMOVE* items at the *END* of an array

```
1 const fruit = [ "apple", "pineapple" ];
2
3 fruit.pop();
4
5 console.log(fruit); // [ "apple" ]
```

forEach() allows us to loop through an array - without the complexity of using a loop!

forEach() allows us to loop through an array - without the complexity of using a loop!

```
1 const listOfFruit = [ "apple", "grape" ];
2
3 listOfFruit.forEach((fruit) => {
4         console.log(fruit);
5 });
```

forEach() allows us to loop through an array - without the complexity of using a loop!

```
1 const listOfFruit = [ "apple", "grape" ];
2
3 listOfFruit.forEach((fruit) => {
4         console.log(fruit);
5 });
```

```
// result

apple
grape
```

filter() returns a new array with a filtered list of results

filter() returns a new array with a filtered list of results

```
1 const ages = [ 53, 42, 23, 45 ];
2
3 ages.filter((age) => {
4         return age > 40;
5 });
```

filter() returns a new array with a filtered list of results

```
1 const ages = [ 53, 42, 23, 45 ];
2
3 ages.filter((age) => {
4         return age > 40;
5 });
```

```
// result

[ 53, 42, 45 ]
```

There's one more thing I want to teach you

There's one more thing I want to teach you

Pay attention because this is a question I have asked candidates during interviews! 🤨

How can we tell if a value is an array?

How can we tell if a value is an array?

Using `typeof` we can tell if a value is:

```
1 const value = 12;
2
3 typeof value; // "number"
```

A number

```
1 const value = 12;
2
3 typeof value; // "number"
```

```
1 const value = "Hello";
2
3 typeof value; // "string"
```

A string

```
1 const value = "Hello";
2
3 typeof value; // "string"
```

```
1 const value = true;
2
3 typeof value; // "boolean"
```

A boolean

```
1 const value = true;
2
3 typeof value; // "boolean"
```

```
1 const value;
2
3 typeof value; // "undefined"
```

Undefined

```
1 const value;
2
3 typeof value; // "undefined"
```

```
1 const value = {};
2
3 typeof value; // "object"
```

An object

```
1 const value = {};
2
3 typeof value; // "object"
```

What happens if we try `typeof` on an array?

```
1 const value = [ "cat", "dog" ];
2
3 typeof value; // there is no "array" type ❌
```

```
1 const value = [ "cat", "dog" ];
2
3 typeof value; // there is no "array" type ❌
```

A candidate once gave me this as his answer in an interview. He didn't get the job 🤣

```
1 const value = [ "cat", "dog" ];
2
3 typeof value; // "object"
```

```
1 const value = [ "cat", "dog" ];
2
3 typeof value; // "object"
```

```
1 const value = [ "cat", "dog" ];
2
3 typeof value; // "object"
```

That's right, arrays have the same type as regular objects

So just how do we find out if a value is an array?

So just how do we find out if a value is an array?

There are a few answers, but I will teach you just one of them

So just how do we find out if a value is an array?

There are a few answers, but I will teach you just one of them

`Array.isArray()`

So just how do we find out if a value is an array?

There are a few answers, but I will teach you just one of them

`Array.isArray()`

```
1 const value = [ "cat", "dog" ];
2
3 Array.isArray(value); // true
4
5 const value = 12;
6
7 Array.isArray(value); // false
```

One more thing...

One more thing...

`Array.isArray` is what's known as a *static* method (more on this later in the course)

One more thing...

`Array.isArray` is what's known as a *static* method (more on this later in the course)

Because of this, you can only access it on the `Array` object

One more thing...

`Array.isArray` is what's known as a *static* method (more on this later in the course)

Because of this, you can only access it on the `Array` object

For example:

One more thing...

`Array.isArray` is what's known as a *static* method (more on this later in the course)

Because of this, you can only access it on the `Array` object

For example:

```
1 const value = [ "cat", "dog" ];
2
3 value.isArray(value); // isArray is not a function
4
5 Array.isArray(value); // true
```