Creating a strongly-typed array

We will learn how to create strongly-typed arrays at the start of this lesson. We will use two different methods to achieve this before learning how to create strongly-typed rest parameters.

WE'LL COVER THE FOLLOWING

Arrays can hold mixed value types

Before we learn how to create a strongly-typed array, let's look at an array that has been declared without a type annotation. The code below constructs an array containing three different values, each with a different type.

</> TypeScript

```
const items = [];
items.push(1);
items.push("two"),
items.push(false);

console.log(items);
```

Why doesn't the code above raise a type error?

Show Answer

Using the Array generic type

We can use a type annotation to specify that an array should only contain items of a specific type. There is an `Array` generic type that we can use to do this. We pass the type we want the items to have in angle brackets after the word `Array`:

</> TypeScript

```
const numbers: Array<number> = [];
numbers.push(1);
numbers.push("two"),
numbers.push(false);

console.log(numbers);
```

Don't worry if the syntax looks a little odd at the moment, we will learn about generic types later in this course.

If we declared the array as follows, would we still get a type error?

```
const items: Array<number> = [1, "two", false];
```

Show Answer

Using the square bracket notation

There's an alternative and arguably simpler method of creating strongly-typed arrays which is to put the type of the array items followed by squared brackets:

```
const items: number[] = [];
```

Use the square bracket notation to create a variable called `numbers` that is assigned to an array of strings containing `"one"`, `"two"`, and `"three"` in the

code widget below.

</> TypeScript

```
// TODO - create an array of strings containing "one", "two" and "three" with a type annotati
```

Show Answer

Using type inference

What is the type of the `array` variable in the example code below?

</> TypeScript

```
const array = [1, 2, 3];
```

Show Answer

So, TypeScript can cleverly infer the type of an array.

Use type inference to create an array of strings containing `"one"`, `"two"`, and `"three"`.

</> TypeScript

```
// TODO - create an array of strings containing "one", "two" and "three" without a type annot
```

Show Answer

Strongly-typing rest parameters

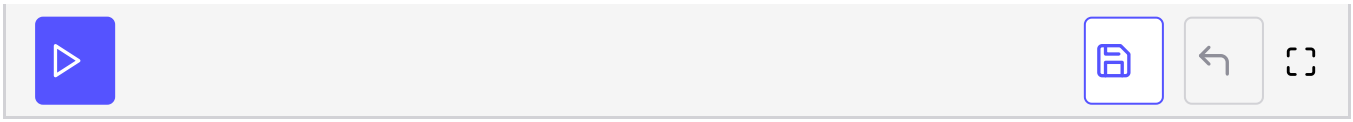
We can use an array type annotation to strongly-type function **rest parameters**.

The plain JavaScript function below takes the name of a person and a varying number of scores and outputs them to the console.

</> TypeScript

```
function logScores(firstName, ...scores) {
  console.log(firstName, scores);
}

logScores("Ben", 50, 75, 85)  // outputs Ben and [50, 75, 85]
```



What type annotations can we add to the function parameters to strongly-type the function above? Try adding them to the code.

Show Answer

With the strongly-typed implementation of the function we try to invoke it with the code below:

```
logScores("Mike", 90, 65, "65")
```

This raises a type error though. What is the problem?

Show Answer

Wrap up

If we assign values when declaring an array, TypeScript can infer the type of the array items. When using type inference, it is crucial to check that TypeScript has inferred the type as we expect by hovering over the value.

If we aren't assigning array items during an array declaration, we can use a type annotation by using the square bracket notation or the `Array` generic type to inform TypeScript of what the type should be.

Array type annotations can also be used to strongly-type a functions rest parameters.

More information on arrays in TypeScript can be found in the TypeScript handbook.

In the next lesson, we will learn what tuples are and how we can use them in TypeScript.