Creating type aliases

In this lesson, we'll learn what a type alias is and when they are useful.

WE'LL COVER THE FOLLOWING

Understanding the need for type aliases

In the last lesson, we created an object type to hold a name and a score:

```
const tomScore: { name: string; score: number; } = { name: "Tom", score: 7
0 };
const bobScore: { name: string; score: number; } = { name: "Bob", score: 8
0 };
const janeScore: { name: string; score: number; } = { name: "Jane", score:
 90 };
```

Wouldn't the code be more readable if we could somehow name `{ name: string; score: number; }` and use that name:

```
const tomScore: Score = { name: "Tom", score: 70 };
const bobScore: Score = { name: "Bob", score: 80 };
const janeScore: Score = { name: "Jane", score: 90 };
```

Well, this is exactly what a type alias is!

Creating a type alias

We define a type alias as follows:

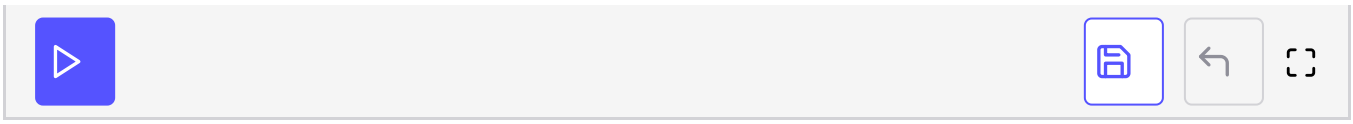
```
type NewName = ExistingType;
```

What would the type alias be for the `Score` example? Try defining this in the code widget below and check that it runs successfully.

</> TypeScript

```
// TODO - create a type alias name Score for { name: string; score: number; }

const tomScore: Score = { name: "Tom", score: 70 };
const bobScore: Score = { name: "Bob", score: 80 };
const janeScore: Score = { name: "Jane", score: 90 };
```



Show Answer

Wrap up

A type alias is a name given to an existing type that can help the readability of our code. If we create a type that is used in several places of our code, they give us a mechanism of reusing a type.

More information on type aliases can be found in the TypeScript handbook.

Good stuff! We are starting to understand how to create a range of different types now.

In the next lesson, we'll learn an alternative method of creating object types.