Using type assertions

In this lesson, we'll learn how to use type assertions to narrow the type of a variable.

Understanding the need for type assertions

Sometimes we know more about a variable value than TypeScript does. Consider the code below; it would be nice to narrow the type of the value returned from `getAge` to `number`:

</> TypeScript

```typescript
function getAge(id: number): any {
  return 42;
}
function calcDiscount(age: number) {
  return age / 100;
}

const discount1 = calcDiscount(getAge(1));
console.log(discount1);
```

The example is a bit contrived, but we often do consume third party functions where the return type is wider than we need in our use case.

Is there a way to tell the TypeScript compiler that the type can be narrowed in these sorts of cases? Well, yes, that's where type assertions come in!

Angle-bracket syntax

There are two types of syntax for a type assertion. The first is to define the type in angle-brackets just before the value we are asserting. So, the age in the above example can be narrowed to `number` as follows:

</> TypeScript

```
function getAge(id: number): any {
  return 42;
}
function calcDiscount(age: number) {
  return age / 100;
}

const discount2 = calcDiscount(<number>getAge(1));
console.log(discount2);
```

Can you think of a potential problem when using this syntax in a React app?

Show Answer

So, don't use this syntax in files where React components are defined.

"As" syntax

The alternative syntax is to put the type after an `as` keyword. So, the age in our example can be narrowed to `number` as follows with the *as* syntax:

</> TypeScript

```
function getAge(id: number): any {
  return 42;
}
function calcDiscount(age: number) {
  return age / 100;
}

const discount3 = calcDiscount(getAge(1) as number);
```

This is the syntax that is generally used in React apps.

What would happen in the above code if the `getAge` function returned a string? Change the `getAge` function in the above code widget to the implementation below and find out.

```
function getAge(id: number): any {
  return "Forty-two";
}
```

Show Answer

When using type assertions, we must know more about the data being used in the program than TypeScript does, or runtime problems could occur.

Wrap up

We can use type asserts in our code to tell the TypeScript what a type should be when it infers it incorrectly. React apps should use the “as” syntax when using a type assertion.

More information can be found about type assertions in the TypeScript handbook.

Congratulations; we now have a good understanding of the basic types in TypeScript! Next up is a quiz to test what we have learned.