Understanding the any type

We will learn about the 'any' type in this lesson.

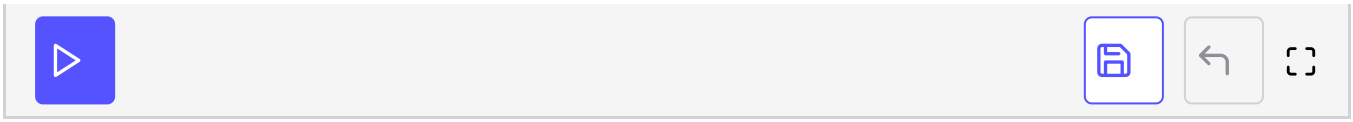
WE'LL COVER THE FOLLOWING

An example

The code below declares a variable called `something` and sets it to various values. From the last lesson, we know that TypeScript will give `something` the `any` type because there is no type annotation or value assignment in the declaration.

</> TypeScript

```
let something;
something = "Bob";
something = 33;
something = false;
something = new Date(2019, 5, 8);
```



What do you think the TypeScript type checker checks in the above code?

Show Answer

TypeScript doesn't carry out type checking on items that have the `any` type. The `any` type is a way of opting out of the type checking process.

Why would you use the `any` type?

One of the main reasons for using TypeScript is to enable type checking. So, why does the `any` type exist, and are there any use cases for using it? Well, when it isn't possible to create a TypeScript type to represent an item, we can use the `any` type. Historically, when TypeScript's type system wasn't as powerful, this would be the case. However, this case rarely happens today.

An example where we may see `any` being used is when we are dealing with dynamic data. This is data that is defined by end-users rather than developers. The ability for end-users to create custom forms is an example of where code would need to deal with dynamic data. The code snippet below is an example of how we could type a custom form's values:

</> TypeScript

```
const formValues: { [field: string]: any } = {
  firstName: "Bob",
  surname: "Smith",
  age: 30
};
```

We haven't covered the syntax shown in the type annotation yet, so don't worry if it looks scary! The type `{ [field: string]: any }` means an object whose property names are of type `string`, with property values of type `any`.

What about when we consume a 3rd party library or a 3rd party web API that doesn't have types? Perhaps we should use `any` in these cases? Well, even in these cases, there is an approach we can use to strongly-type this code. We'll learn about this when we learn about the `unknown` type later in this chapter.

Wrap up

The main take away from this lesson is that no type checking occurs on code that uses the `any` type. We rarely need to use `any` today because TypeScript's type system is so flexible.

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

In the next lesson, we'll learn about the `void` type.