Using the unknown type

In this lesson, we will learn about the 'unknown' type; it is similar to the 'any' type but is type-safe.

WE'LL COVER THE FOLLOWING

An `any` type refresher

We learned from a previous lesson that the `any` type could be used when we are unsure of the type of value. The problem with `any` is that no type checks will be carried out on values of type `any`.

What if there was a type like `any` that can be used for values we don't know but was also type-safe? This is what the `unknown` type is!

Understanding the unknown type

In order to understand the `unknown` type, look at the code below and then run it.

</> TypeScript

```
function add(a: unknown, b: unknown) {
  return a + b;
}
```

Why do you think we get type errors when the code is executed?

Show Answer

This confirms that type checking does occur on values of type `unknown`.

In this example function, we want to return the sum of the parameters if they are numbers, otherwise, return `0`. Have a go at doing this in the code above that is currently erroring.

Show Answer

We have to carry out explicit type checks in the code before we can do any useful operations on the `unknown` type or access any of its properties or functions.

Type-safe data fetching

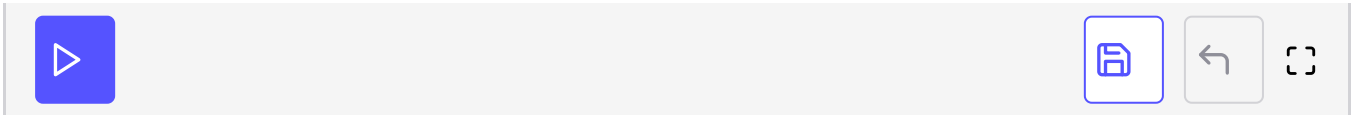
Consider the following function that gets a person object from a web API. Run the code to see if it works.

</> TypeScript

```
async function getData(path: string): Promise<unknown> {
  const response = await fetch(path)
  return await response.json();
}

type Person = {
  id: string;
  name: string;
}

async function getPerson(id: string): Promise<Person | null> {
  const person = await getData("/people/1");
  if (person) {
    return person;
  }
  return null;
}
```



There is some syntax that we are not familiar with yet in the above code in the

There is some syntax that we are not familiar with yet in the above code in the function's return type annotations. We'll learn about this syntax later in this course but for now, here's an explanation of these return types:

- The `getData` function returns a promise of type `unknown`
- The `getPerson` function returns a promise of type `Person` or `null`.

Why do we get a type error on line 14?

Show Answer

We need to check that `person` on line 13 is in fact of type `Person` to resolve the type error. We can use what's called a *type predicate* to check that an object is of type `Person`:

```
function isPerson(person: any): person is Person {
  return "id" in person && "name" in person;
}
```

Notice the return type, `person is Person`. This is a type predicate; it is a special return type that the Typescript compiler uses to know what type a particular value is.

Add the `isPerson` function to the problem code and have a go at referencing it on line 13 to resolve the type error.

Show Answer

Wrap up

The `unknown` type allows us to reduce our use of `any` and create more strongly-typed code. We do write a little more code when using `unknown`, but the confidence we get from knowing our code is type-safe is well worth it.

Well done! We are starting to understand some of the more unusual types in TypeScript now!

In the next lesson, we'll learn about a mechanism that allows us to tell

TypeScript what the type of a variable is that we are consuming.