Using type annotations

In this lesson, we'll learn how to assign types to variables and functions using type annotations.

WE'LL COVER THE FOLLOWING

Assigning types to variables

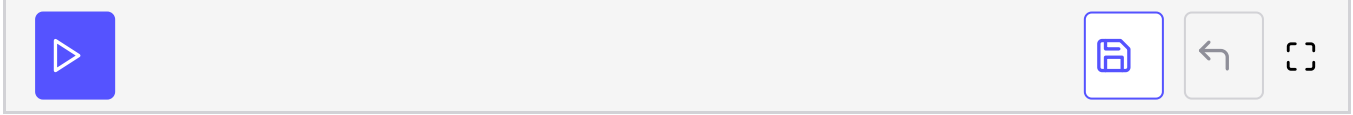
TypeScript's **type annotations** allow us to assign types to variables. The syntax for a type annotation is as follows: put a colon followed by the variable name and type before any assignment. For example, the `score` variable is declared as a `number` in the code below:

```
let score: number;
```

Let's assign a couple of different values to `score`:

</> TypeScript

```
let score: number;
score = 10;
score = "ten";
```



If you run the code above, a type error is raised on line 3. Why is this?

Show Answer

Assigning types to functions

We can add type annotations to function parameters and to its return value.

The syntax for a type annotation on a parameter is just like type annotations on variables, a colon followed by the type is added after the parameter name. For example:

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

In the above `add` function, both the `a` and `b` parameters are of type `number`.

We can also add a type annotation for the return value by adding a colon followed by the type after the parentheses of the function. For example:

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

We have defined that the return value in the above `add` function is of type `number`.

Types can also be added to function expressions in the same way. Update the following function to make it strongly-typed.

</> TypeScript

```
const minus = function(a, b) {
  return a - b
}
```

Show Answer

What about arrow functions? Have a go at making the arrow function below strongly-typed.

</> TypeScript

```
const multiply = (a, b) => a * b;
```

Show Answer

Optional parameters

Observe and then run the following code:

</> TypeScript

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

add(3);
```

Why is a type error raised on line 5?

Show Answer

We can define that a parameter is optional by putting a question mark (`?`) before the colon. It is important to note that optional parameters can only be at the end of the parameter list.

Let's make the `b` parameter optional:

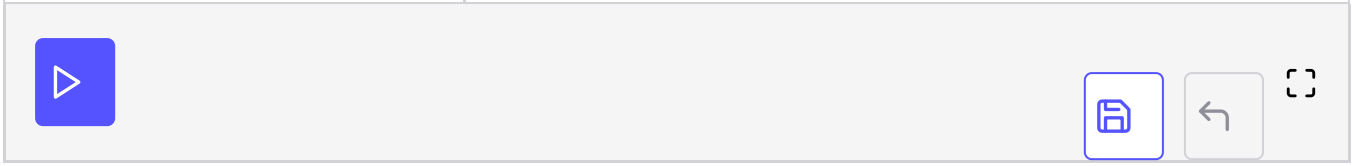
</> TypeScript

index.ts

tsconfig.json

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

add(3);
```



There is a problem with this code, can you spot it? Try running the code for a clue.

Show Answer

How can we use the *nullish coalescing operator* ('??') to resolve this problem? This operator returns the right-hand operand if the left-hand operand is `null` or `undefined`.

Show Answer

Wrap up

TypeScript allows us to declare the type that a variable can hold using a type annotation. We can also declare types of function parameters and function return types using type annotations.

By default, function parameters are required in TypeScript. We can make function parameters optional by placing a question mark (`?`) in front of its type annotation.

A couple of closing questions:

- What do you think happens to type annotations when TypeScript code is transpiled to JavaScript?
- Will TypeScript do any type checking at runtime?

Show Answers

Well done, that's another lesson completed!

In the next lesson, we will learn how TypeScript can assign types to variables and functions without type annotations.