Understanding type inference

Adding type annotations to our code means we need to write extra code which consumes our time and bloats our code. TypeScript has something called type inference which means, in many cases, it can work out a variable's type without it having a type annotation. In this lesson, we'll learn how and when we can use type inference.

WE'LL COVER THE FOLLOWING

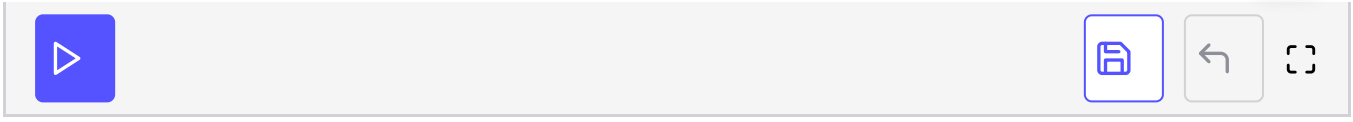
Using type inference on variable declarations

TypeScript will automatically infer the type for a variable if a type annotation hasn't been specified.

The code below declares a variable and initializes it with the value `10` without a type annotation:

</> TypeScript

```
let score = 10;
```



Hover over the `score` variable. What has TypeScript inferred the type to be?

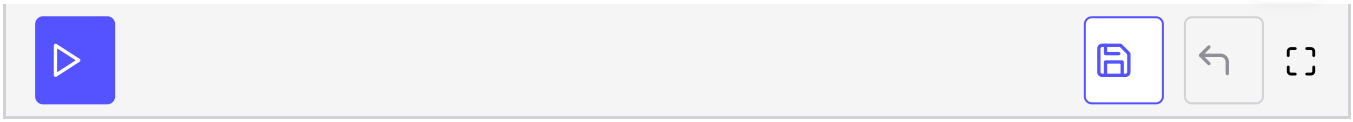
Show Answer

So, TypeScript can automatically infer the type of a variable from the value it is assigned. Great!

Now look at another example below:

</> TypeScript

```
let dateOfBirth = new Date(1990, 4, 7);
```



What do you think TypeScript inferred the type to be?

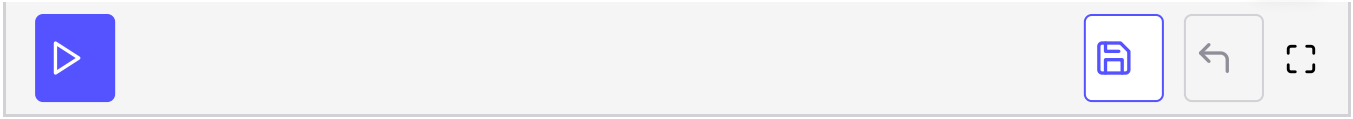
Show Answer

So, TypeScript can infer types that don't exist in JavaScript. Neat!

Let's look at the type of a variable defined with `const`.

</> TypeScript

```
const firstName = "Bob";
```



What is the type of `firstName`? Surely it is `string`, right?

Show Answer

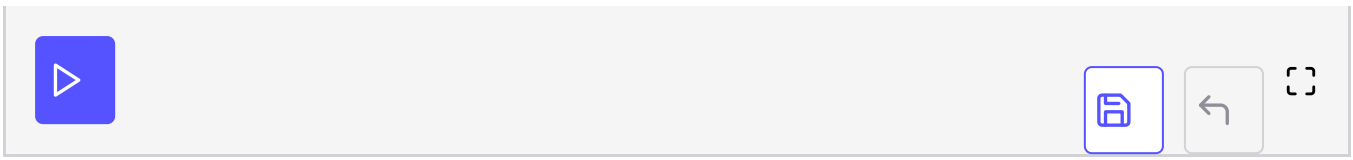
TypeScript infers the type of a string constant to the value of the constant rather than the wider `string` type. This is because a string constant can only be that value. A string type that can only be a specific value is sometimes referred to as a *string literal type*.

Does this apply to all constants, though? Consider the code below:

</> TypeScript

```
const age = 31;
```

```
const created = new Date(2019, 11, 6);
```



What are the types of `age` and `created`?

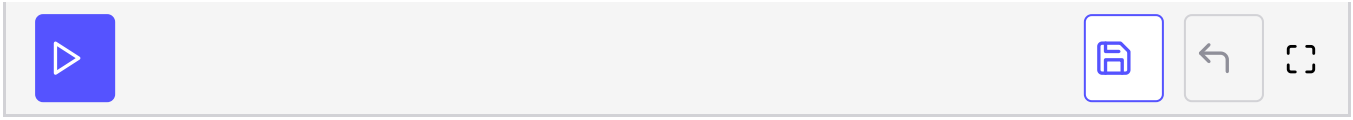
Show Answer

So, when a constant is initialized from a primitive type, TypeScript infers it to be a literal type of the specific value assigned. However, when a constant is initialized from a non-primitive type, TypeScript only infers it to be of the same type as assigned.

What about when the value is assigned from another variable? Have a look at the code below:

</> TypeScript

```
const first = "Bob";
const last = "Smith";
let fullName = first;
fullName += " ";
fullName += last;
```



What do you think TypeScript has inferred the type of `fullName` to be?

Show Answer

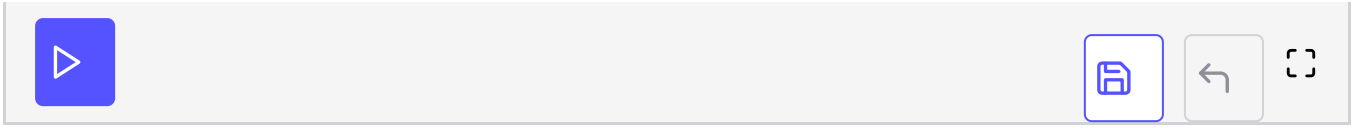
So, it doesn't matter whether we assign a variable to another variable or a specific value; TypeScript will still infer the type in the same manner.

What if we don't assign any value to a variable when it is declared?

</> TypeScript

```
let counter;
```

```
let counter;
counter = 10;
```



What is the type of `counter` in the above example?

Show Answer

Remember that TypeScript infers the type from a value assignment when the variable is declared. In the above example, there is no value assignment, so TypeScript doesn't assign a specific type. If TypeScript was really smart, perhaps it could infer the type from the value assignment on line 2, but at the moment, it doesn't.

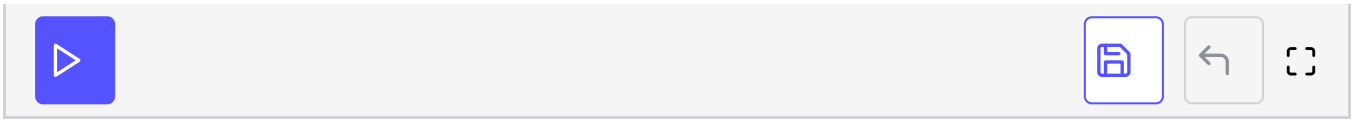
Using type inference with functions

Type inference happens on functions as well. Consider the example below where we have a function without a return type:

</> TypeScript

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

const ten = add(5, 5);
```



What is the return type of the `add` function? Also, what is the type of the `ten` variable?

Show Answer

So, perhaps as we would expect, TypeScript does infer the return type of a function. TypeScript also infers the type of a variable in a declaration where the assignment is from a function. Cool!

What if we don't have type annotations on the function parameters?

</> TypeScript

```
function addTen(a) {
  return a + 10;
}

const fourteen = addTen(4);
```

What is the type of the `a` parameter? What is the return type of `addTen`? What about the type of `fourteen`?

Show Answer

So, TypeScript's inference breaks down on functions when no type annotations are defined on their parameters.

Wrap up

TypeScript's smart type inference can save us time and arguably make our code more readable by not having type annotations for every declaration.

If we don't specify a type annotation, TypeScript will infer the type from the value assignment. It is essential to be aware of this and check that the inferred type is as required.

TypeScript generally can't infer the type of function parameters, so we should supply type annotations for these. The exception is where the parameter has a default value that TypeScript can use to infer the type.

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

Next up, we'll learn about TypeScript's `any` type.