**TYPES**

**From JavaScript to TypeScript**

Invented in 1995, JavaScript was designed as a small scripting language for simple web pages in browsers. It wasn’t until 1999 that JavaScript was capable of supporting the kinds of dynamic web pages we see today, and using JavaScript that way wasn’t common practice until 2005.

To serve its original use-case, JavaScript was designed to be very flexible and easy to use for small applications. These features make JavaScript a great first language to learn, but they also make it less-than-ideal for building larger-scale applications with hundreds or even thousands of files. Stricter programming languages will inform the developer when they change one area of code in a way that will break other areas. JavaScript will not, which often leads to unexpected behavior at runtime.

To address these shortcomings, Microsoft developed TypeScript and released publicly in 2012 to blend the flexibility of JavaScript with the advantages of a stricter language.

TypeScript is a programming language that adds *types* to JavaScript. It allows us to write JavaScript with a set of tools called a *type system* that can spot potential bugs in, clarify the structure of, and help refactor our code. In addition, TypeScript added newer JavaScript language features, such as arrow functions and classes, years before they were added to JavaScript officially.

Today, TypeScript is one of the [most loved languages](https://insights.stackoverflow.com/survey/2019#most-loved-dreaded-and-wanted) in the world, from open source projects such as Angular and Webpack to large-scale development at Amazon and Google. We use it at Codecademy!

**TYPES**

**What is TypeScript?**

So now that we know *why* TypeScript exists, let’s talk about how we use it:

* First, we write TypeScript code in files with the extension **.ts**.
* Next, we run our code through the TypeScript [transpiler](https://en.wikipedia.org/wiki/Source-to-source_compiler" \t "_blank). The transpiler will check that the code adheres to TypeScript’s standards, and it will display errors when it does not.
* If the TypeScript code can be converted into working JavaScript, the transpiler will output a JavaScript version of the file (**.js**).

TypeScript code is a *superset* of JavaScript code—it has all the features of traditional JavaScript but adds some new features.

Given this TypeScript code as input:

let firstName = 'Anders';

The TypeScript transpiler would output this JavaScript code:

let firstName = 'Anders';

That’s right, they’re the same! TypeScript code generally looks roughly the same as the equivalent JavaScript.

What we’ll learn throughout this lesson and course is the real joy of using TypeScript—that it helps us understand more about our code and, particularly, discover bugs.

**Instructions**

**1.**

In this exercise and throughout the lesson, we’ll provide you with a [bash](https://www.codecademy.com/learn/learn-the-command-line) terminal to execute your code. We’ll be working with both TypeScript files (files with the extension **.ts**) and JavaScript files (files with the extension **.js**).

The TypeScript transcompiler can be used on the command line by running the tsc command:

tsc

This will create an equivalent **.js** file in the same directory as well as surface any errors found by the TypeScript transcompiler.

Run the TypeScript transcompiler using the tsc command.

Checkpoint 2 Passed

Stuck? Get a hint

**2.**

Great work! You’ve now created an index.js file (you can confirm this by running the ls command in your terminal).

Run the resultant JavaScript file with the node command:

node index.js

**Type Inferences**

JavaScript allows us to assign any value to any variable. That makes it very flexible to use, which can be good for getting started quickly in coding. In practice, variables that are assigned values of multiple types throughout a program can be very confusing or lead to bugs.

One of the first things we’ll discover with TypeScript is that when we declare a variable with an initial value, the variable can never be reassigned a value of a different data type. This is an example of *type inference*: everywhere in our program, TypeScript expects the data type of the variable to match the type of the value initially assigned to it at declaration.

TypeScript recognizes JavaScript’s built-in “primitive” data types:

* boolean
* number
* null
* string
* undefined

If we try to reassign a variable to a value of a different type, TypeScript will surface an error.

let order = 'first';  
   
order = 1;

Running the TypeScript code above will result in the following error being surfaced in the terminal: Type '1' is not assignable to type 'string'.

TypeScript’s type system is telling us that order is expected to be the primitive type string, but we’re trying to assign it a value of type number. That’s not allowed: variables can only be assigned the types expected by the type system.

We can fix this complaint by changing the new value to be the expected string type:

let order = 'first';  
   
order = '1';

Let’s practice!

**Instructions**

**1.**

Our code has a bug! 🙀 When we run our JavaScript code in **index.js** (node index.js), it prints NaN years instead of the expected 28 years.

Use the tsc command to run the **index.ts** file and surface any type errors.

Checkpoint 2 Passed

Stuck? Get a hint

**2.**

It looks like we’re assigning the wrong type of data to realAge.

Change the assigned value of realAge to be the expected value with the correct type.

**TYPES**

**Type Shapes**

Because TypeScript knows what types our objects are, it also knows what shapes our objects adhere to. An object’s shape describes, among other things, what properties and methods it does or doesn’t contain.

The built-in types in JavaScript each have known properties and methods that always exist. All strings, for example, are known to have a .length property and .toLowerCase() method.

"OH".length; // 2  
"MY".toLowerCase(); // "my"

TypeScript’s tsc command will let you know if your code tries to access properties and methods that don’t exist:

"MY".toLowercase();  
// Property 'toLowercase' does not exist on type '"MY"'.  
// Did you mean 'toLowerCase'?

Through this knowledge of type shapes, TypeScript helps us quickly locate bugs in our code.

Let’s try it!

### Instructions

**1.**

More bugs! 🙀🙀🙀

We already compiled our **index.ts** file. Try running **index.js** by entering node index.js into the bash shell. The code throws an error instead of printing the following expected output:

"MURIEL!"  
7

Use TypeScript’s tsc command to see type errors in the index.ts file.

Stuck? Get a hint

**2.**

The TypeScript compiler is surfacing two errors. Using the error messages to guide you, fix both of the errors in the **index.ts** file.

Stuck? Get a hint

**3.**

Confirm that everything is now working properly. Run tsc again to compile your TypeScript code and then run node index.js to run the JavaScript program.

**Any**

There are some places where TypeScript will not try to infer what type something is—generally when a variable is declared without being assigned an initial value. In situations where it isn’t able to infer a type, TypeScript will consider a variable to be of type any.

Variables of type any can be assigned to any value and TypeScript won’t give an error if they’re reassigned to a different type later on.

let onOrOff;  
   
onOrOff = 1;  
onOrOff = false;

In the code above, we declared the variable onOrOff without an initial value. TypeScript considers it to be of type any, and, therefore, doesn’t produce an error when we change the variable’s assignment from a number value to a boolean value.

### Instructions

**1.**

Declare a variable with let named guess, without assigning it a value.

Checkpoint 2 Passed

Stuck? Get a hint

**2.**

We’re thinking of a color… take a guess at what it is. On a new line, assign this color (as a string) to the guess variable.

Checkpoint 3 Passed

Stuck? Get a hint

**3.**

Now we’re thinking of a number… Take a guess at what it is. On a new line, reassign the guess variable to your number guess.

Checkpoint 4 Passed

Stuck? Get a hint

**4.**

Now let’s verify that TypeScript has no problem with the assignments of our guess variable. Run tsc and notice the lack of error messages!

**Variable Type Annotations**

In some situations, we’d like to declare a variable without an initial value while still ensuring that it will only ever be assigned values of a certain type. If left as any, TypeScript won’t be able to protect us from accidentally assigning a variable to an incorrect type that could break our code.

We can tell TypeScript what type something is or will be by using a type annotation.

Variables can have type annotations (also known as type declarations) added just after their names. We provide a type annotation by appending a variable with a colon (:) and the type (e.g., number, string, or any).

let mustBeAString : string;  
mustBeAString = 'Catdog';  
   
mustBeAString = 1337;  
// Error: Type 'number' is not assignable to type 'string'

In the code above, we explicitly declare mustBeAString to be of type string without assigning it an initial value. This enables us to assign it the value 'Catdog' without complaint, but when we later attempt to assign it a numerical value, TypeScript will give us an error message telling us that a number is being improperly assigned to a variable of type string.

We’re becoming familiar with how useful type annotations can be! Some developers may find that type annotations make the code wordy or hard for others to understand, however, they get automatically removed when compiled to JavaScript.

### Instructions

**1.**

The phoneNumber variable is meant to be able to store non-numeric characters (for compatibility with international phone numbers). Add a type declaration to inform TypeScript that it should be a string.

Stuck? Get a hint

**2.**

Use the tsc command to check the file for type errors.

Stuck? Get a hint

**3.**

Error! It looks like one of the values assigned to phoneNumber isn’t valid. Fix that value per TypeScript’s type complaints.

(And notice that the randomness introduced by Math.random() has no impact on whether TypeScript finds the type error.)

**Review**

All right! You’ve taken some first steps into the wonderful world of type safety with TypeScript. 💪

To recap, you’ve learned:

* *TypeScript* is a superset of JavaScript that adds types.
* The *type system* refers to TypeScript’s understanding of how your code is meant to function: mainly what data types should be stored under your variables.
* TypeScript expects the data type of the variable to match the type of the value initially assigned to it at its declaration—this is known as *type inference*.
* An object’s *shape* describes, among other things, what properties and methods it does or doesn’t contain. TypeScript knows not only what type something is but also what shape that type has.
* When it isn’t able to infer a type, TypeScript will consider a variable to be of type any.
* *Type annotations* are little pieces of code that indicate what type a variable is meant to be.

let youAreAwesome: boolean;  
youAreAwesome = true;