Chapter 1. Introduction

So, you decided to buy a book about TypeScript. Why?

Maybe it's because you're sick of those weird `cannot read property blah of undefined` JavaScript errors. Or maybe you heard TypeScript can help your code scale better, and wanted to see what all the fuss is about. Or you're a C# person, and have been thinking of trying out this whole JavaScript thing. Or you're a functional programmer, and decided it was time to take your chops to the next level. Or your boss was so fed up with your code causing production issues that they gave you this book as a Christmas present (stop me if I'm getting warm).

Whatever your reasons are, what you've heard is true. TypeScript is the language that will power the next generation of web apps, mobile apps, NodeJS projects, and Internet of Things (IoT) devices. It will make your programs safer by checking for common mistakes, serve as documentation for yourself and future engineers, make refactoring painless, and make, like, half of your unit tests unnecessary ("What unit tests?"). TypeScript will double your productivity as a programmer, and it will land you a date with that cute barista across the street.

But before you go rushing across the street, let's unpack all of that a little bit, starting with this: what exactly do I mean when I say "safer"? What I am talking about, of course, is *type safety*.

TYPE SAFETY

Using types to prevent programs from doing invalid things.[1]

Here are a few examples of things that are invalid:

- Multiplying a number and a list
- Calling a function with a list of strings when it actually needs a list of objects

- Calling a method on an object when that method doesn't actually exist on that object
- Importing a module that was recently moved

There are some programming languages that try to make the most of mistakes like these. They try to figure out what you really meant when you did something invalid, because hey, you do what you can, right? Take JavaScript, for example:

```
3 + []              // Evaluates to the string "3"

let obj = {}
obj.foo             // Evaluates to undefined

function a(b) {
  return b/2
}
a("z")              // Evaluates to NaN
```

Notice that instead of throwing exceptions when you try to do things that are obviously invalid, JavaScript tries to make the best of it and avoids exceptions whenever it can. Is JavaScript being helpful? Certainly. Does it make it easier for you to catch bugs quickly? Probably not.

Now imagine if JavaScript threw more exceptions instead of quietly making the best of what we gave it. We might get feedback like this instead:

```
3 + []              // Error: Did you really mean to add

let obj = {}
obj.foo             // Error: You forgot to define the pr

function a(b) {
  return b/2
}
a("z")              // Error: The function "a" expects a
                    // but you gave it a string.
```

Don't get me wrong: trying to fix our mistakes for us is a neat feature for a programming language to have (if only it worked for more than just programs!). But for JavaScript, this feature creates a disconnect between when

you make a mistake in your code, and when you *find out* that you made a mistake in your code. Often, that means that the first time you hear about your mistake will be from someone else.

So here's a question: when exactly does JavaScript tell you that you made a mistake?

Right: when you actually *run* your program. Your program might get run when you test it in a browser, or when a user visits your website, or when you run a unit test. If you're disciplined and write plenty of unit tests and end-to-end tests, smoke test your code before pushing it, and test it internally for a while before shipping it to users, you will hopefully find out about your error before your users do. But what if you don't?

That's where TypeScript comes in. Even cooler than the fact that TypeScript gives you helpful error messages is *when* it gives them to you: TypeScript gives you error messages *in your text editor, as you type*. That means you don't have to rely on unit tests or smoke tests or coworkers to catch these sorts of issues: TypeScript will catch them for you and warn you about them as you write your program. Let's see what TypeScript says about our previous example:

```
3 + []              // Error TS2365: Operator '+' cannot
                    // and 'never[]'.

let obj = {}
obj.foo             // Error TS2339: Property 'foo' does

function a(b: number) {
  return b / 2
}
a("z")              // Error TS2345: Argument of type '"z
                    // parameter of type 'number'.
```

In addition to eliminating entire classes of type-related bugs, this will actually change the way you write code. You will find yourself sketching out a program at the type level before you fill it in at the value level;[2] you will think about edge cases as you design your program, not as an afterthought; and you will design programs that are simpler, faster, easier to understand, and easier to maintain.

Are you ready to begin the journey? Let's go!

---

1 Depending on which statically typed language you use, "invalid" can mean a range of things, from programs that will crash when you run them to things that won't crash but are clearly nonsensical.

2 If you're not sure what "type level" means here, don't worry. We'll go over it in depth in later chapters.