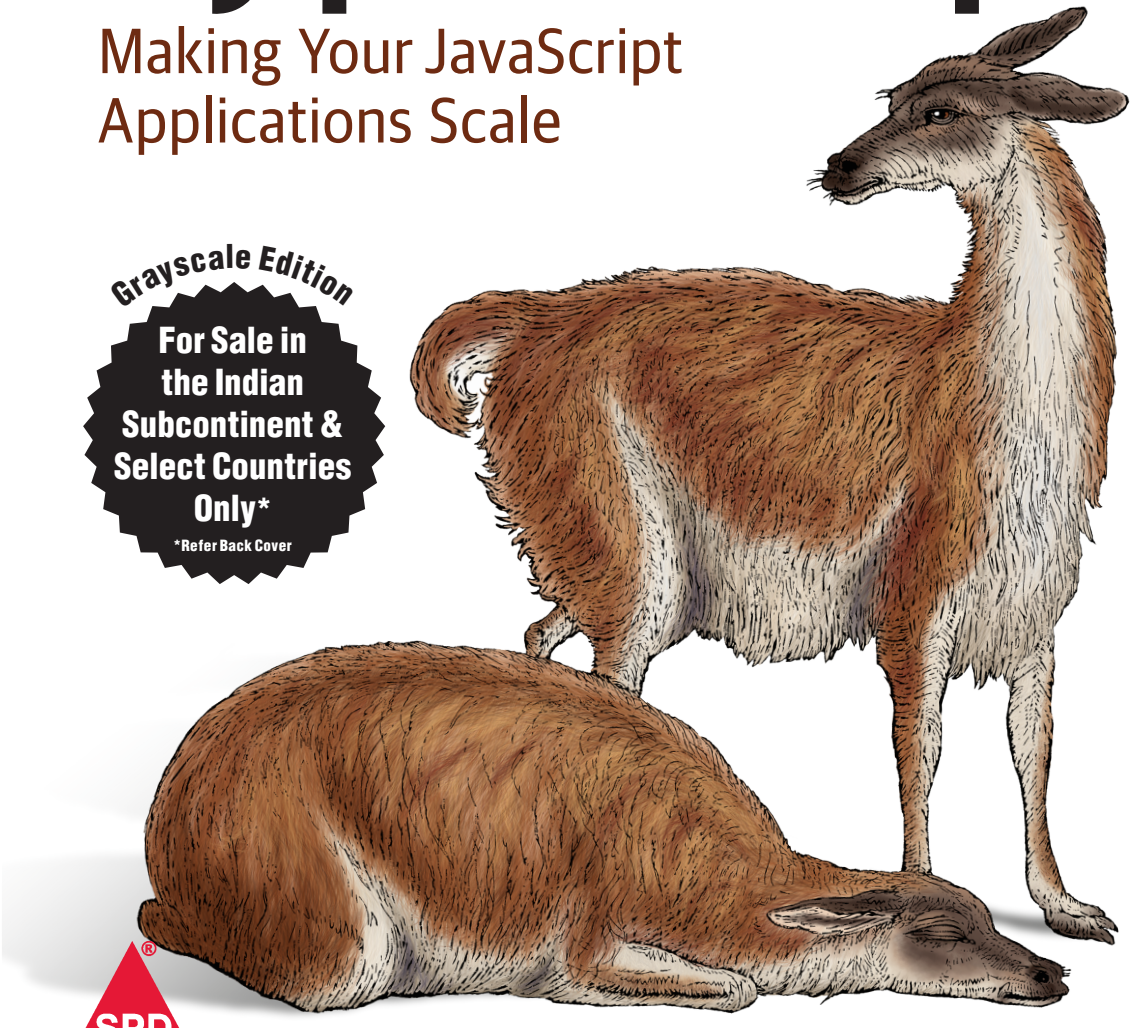
O'REILLY®

Programming TypeScript

Making Your JavaScript Applications Scale



Grayscale Edition

For Sale in the Indian Subcontinent & Select Countries Only*

*Refer Back Cover

SPD®

Boris Cherny

Programming TypeScript

Making Your JavaScript Applications Scale

Boris Cherny

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

SPD®

SHROFF PUBLISHERS & DISTRIBUTORS PVT. LTD.

Mumbai *Bangalore* *Kolkata* *New Delhi*

Programming TypeScript

by Boris Cherny

 ISBN: 978-1-492-03765-1
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles *(http://safari.oreilly.com)*. For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Development Editor: Angela Rufino
Acquisitions Editor: Jennifer Pollock
Production Editor: Katherine Tozer
Copyeditor: Rachel Head
Proofreader: Charles Roumeliotis

Indexer: Margaret Troutman
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

Printing History:
May 2019: First Edition

Revision History for the First Edition 2019-04-18: First Release

See http://oreilly.com/catalog/errata.csp?isbn=9781492037651 for release details.

First Indian Reprint: May 2019

ISBN: 978-93-5213-834-0

Published by **Shroff Publishers & Distributors Pvt. Ltd.** B-103, Railway Commercial Complex, Sector 3, Sanpada (E), Navi Mumbai 400705 • TEL: (91 22) 4158 4158 • FAX: (91 22) 4158 4141 E-mail:spdorders@shroffpublishers.com•Web:www.shroffpublishers.com Printed at Decco Book Prints Pvt. Ltd., Mumbai.

To Sasha and Michael, who might also fall in love with types, someday.

Table of Contents

Preface

This is a book for programmers of all walks: professional JavaScript engineers, C# people, Java sympathizers, Python lovers, Ruby aficionados, Haskell nerds. Whatever language(s) you write in, so long as you have some experience programming and know the basics of functions, variables, classes, and errors, this book is for you. Some experience with JavaScript, including a basic knowledge of the Document Object Model (DOM) and the network, will help you along the way—while we don't dive deep into these concepts, they are a wellspring of excellent examples, and if you're not familiar with them the examples might not make as much sense.

Regardless of what programming languages you've used in the past, what unites all of us is our shared experience of tracking down exceptions, tracing through code line by line to figure out what went wrong and how we can fix it. This is the experience that TypeScript helps prevent by examining your code automatically and pointing out the mistakes you may have missed.

It's OK if you haven't worked with a statically typed language before. I'll teach you about types and how to use them effectively to make your programs crash less, document your code better, and scale your applications across more users, engineers, and servers. I'll try to avoid big words when I can, and explain ideas in a way that's intuitive, memorable, and practical, using lots of examples along the way to help keep things concrete.

That's the thing about TypeScript: unlike a lot of other typed languages, TypeScript is intensely practical. It invents completely new concepts so you can speak more concisely and precisely, letting you write applications in a way that's fun, modern, and safe.

How This Book Is Organized

This book has two aims: to give you a deep understanding of how the TypeScript language works (theory) and provide bucketfuls of pragmatic advice about how to write production TypeScript code (practice).

Because TypeScript is such a practical language, theory quickly turns to practice, and most of this book ends up being a mix of the two, with the first couple of chapters almost entirely theory, and the last few almost completely practice.

I'll start with the basics of what compilers, typecheckers, and types are. I'll then give a broad overview of the different types and type operators in TypeScript, what they're for, and how you use them. Using what we've learned, I'll cover some advanced topics like TypeScript's most sophisticated type system features, error handling, and asynchronous programming. Finally, I'll wrap up with how to use TypeScript with your favorite frameworks (frontend and backend), migrating your existing JavaScript project to TypeScript, and running your TypeScript application in production.

Most chapters come with a set of exercises at the end. Try to do these yourself—they'll give you a deeper intuition for what we cover than just reading would. Answers for chapter exercises are available online, at *https://github.com/bcherny/programming-typescript-answers*.

Style

Throughout this book, I tried to stick to a single code style. Some aspects of this style are deeply personal—for example:

- I only use semicolons when necessary.
- I indent with two spaces.
- I use short variable names like `a`, `f`, or `_` where the program is a quick snippet, or where the structure of the program is more important than the details.

Some aspects of the code style, however, are things that I think you should do too. A few of these are:

- You should use the latest JavaScript syntax and features (the latest JavaScript version is usually just called "esnext"). This will keep your code in line with the latest standards, improving interoperability and Googleability, and it can help reduce ramp-up time for new hires. It also lets you take advantage of powerful, modern JavaScript features like arrow functions, promises, and generators.

- You should keep your data structures immutable with spreads (`...`) most of the time.[1]
- You should make sure everything has a type, inferred when possible. Be careful not to abuse explicit types; this will help keep your code clear and terse, and improve safety by surfacing incorrect types rather than bandaiding over them.
- You should keep your code reusable and generic. Polymorphism (see "Polymorphism" on page 64) is your best friend.

Of course, these ideas are hardly new. But TypeScript works especially well when you stick to them. TypeScript's built-in downlevel compiler, support for read-only types, powerful type inference, deep support for polymorphism, and completely structural type system encourage good coding style, while the language remains incredibly expressive and true to the underlying JavaScript.

A couple more notes before we begin.

JavaScript doesn't expose pointers and references; instead it has value and reference types. Values are immutable, and include things like strings, numbers, and booleans, while references point to often-mutable data structures like arrays, objects, and functions. When I use the word "value" in this book, I usually mean it loosely to refer to either a JavaScript value or a reference.

Lastly, you might find yourself writing less-than-ideal TypeScript code in the wild when interoperating with JavaScript, or incorrectly typed third-party libraries, or legacy code, or if you're in a rush. This book largely presents how you *should* write TypeScript, and makes an argument for why you should try really hard not to make compromises. But in practice, how correct your code is is up to you and your team.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, data types, environment variables, statements, and keywords.

1 If you're not coming from JavaScript, here's an example: if you have an object `o`, and you want to add a property `k` to it with the value `3`, you can either mutate `o` directly—`o.k = 3`—or you can apply your change to `o`, creating a *new* object as a result—`let p = {...o, k: 3}`.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/bcherny/programming-typescript-answers*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Programming TypeScript* by Boris Cherny (O'Reilly). Copyright 2019 Boris Cherny, 978-1-492-03765-1."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

For almost 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, conferences, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, please visit *http://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/programming-typescript*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

This book is the product of years' worth of snippets and doodles, followed by a year's worth of early mornings and nights and weekends and holidays spent writing.

Thank you to O'Rielly for the opportunity to work on this book, and to my editor Angela Rufino for the support throughout the process. Thank you to Nick Nance for his contribution in "Typesafe APIs" on page 210, and to Shyam Seshadri for his contribution in "Angular 6/7" on page 207. Thanks to my technical editors: Daniel Rosenwasser of the TypeScript team, who spent an unreasonable amount of time reading through this manuscript and guiding me through the nuances of TypeScript's type system, and Jonathan Creamer, Yakov Fain, and Paul Buying, and Rachel Head for technical edits and feedback. Thanks to my family—Liza and Ilya, Vadim, Roza and Alik, Faina and Yosif—for encouraging me to pursue this project.

Most of all, thanks to my partner Sara Gilford, who supported me throughout the writing process, even when it meant calling off weekend plans, late nights writing and coding, and far too many unprompted conversations about the ins and outs of type systems. I couldn't have done it without you, and I'm forever grateful for your support.

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]

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.

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 a number and an array?

let obj = {}
obj.foo           // Error: You forgot to define the property "foo" on obj.

function a(b) {
  return b/2
}
a("z")            // Error: The function "a" expects a number,
                  // 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 be applied to types '3'
                  // and 'never[]'.

let obj = {}
obj.foo           // Error TS2339: Property 'foo' does not exist on type '{}'.

function a(b: number) {
  return b / 2
}
a("z")            // Error TS2345: Argument of type '"z"' is not assignable to
                  // 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!

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

O'REILLY®

Programming TypeScript

Any programmer working with a dynamically typed language will tell you how hard it is to scale to more lines of code and more engineers. That's why Facebook, Google, and Microsoft invented gradual static type layers for their dynamically typed JavaScript and Python code. This practical book shows you how one such type layer, TypeScript, is unique among them: it makes programming fun with its powerful static type system.

If you're a programmer with intermediate JavaScript experience, author Boris Cherny will teach you how to master the TypeScript language. You'll understand how TypeScript can help you eliminate bugs in your code and enable you to scale your code across more engineers than you could before.

In this book, you'll:

- **Start with the basics:** Learn about TypeScript's different types and type operators, including what they're for and how they're used
- **Explore advanced topics:** Understand TypeScript's sophisticated type system, including how to safely handle errors and build asynchronous programs
- **Dive in hands-on:** Use TypeScript with your favorite frontend and backend frameworks, migrate your existing JavaScript project to TypeScript, and run your TypeScript application in production

Boris Cherny is an engineering and product leader at Facebook. Previously, he worked in VC, ad tech, and at several startups. He is interested in programming languages, code synthesis and static analysis, and building user experiences that people love.

"This is the right book to help you learn TypeScript in depth. *Programming TypeScript* shows all the benefits of using a type system on top of JavaScript and provides deep insight into how to master the language."

—**Minko Gechev**
Engineer, Angular Team at Google

"*Programming Typescript* onboarded me to the TypeScript tooling and overall ecosystem quickly and efficiently. Every usage question I had was covered by concise, real-world examples. The "Advanced Types" chapter breaks down terminology I usually stumble over, and shows how to leverage TypeScript to create extremely safe code that's still pleasant to use."

—**Sean Grove**
Cofounder of OneGraph

PROGRAMMING / JAVASCRIPT

MRP: ₹ 1,150 .00

Twitter: @oreillymedia
facebook.com/oreilly

SHROFF PUBLISHERS & DISTRIBUTORS PVT. LTD.

ISBN : 978-93-5213-834-0

9 789352 138340

First Edition/2019/Paperback/English