

O'REILLY®

Second
Edition

Learning Go

An Idiomatic Approach to
Real-World Go Programming



Jon Bodner

Learning Go

Go has rapidly become the preferred language for building web services. Plenty of tutorials are available to teach Go's syntax to developers with experience in other programming languages, but tutorials aren't enough. They don't teach Go's idioms, so developers end up recreating patterns that don't make sense in a Go context. This practical guide provides the essential background you need to write clear and idiomatic Go.

No matter your level of experience, you'll learn how to think like a Go developer. Author Jon Bodner introduces the design patterns experienced Go developers have adopted and explores the rationale for using them. This updated edition also includes a new chapter on Go tooling.

This book helps you:

- Write idiomatic code in Go and design a Go project
- Understand the reasons behind Go's design decisions
- Set up a Go development environment for solo developers or teams
- Learn how and when to use reflection, unsafe, and cgo
- Discover how Go's features allow the language to run efficiently
- Know which features you should use sparingly or not at all
- Improve performance, optimize memory usage, and reduce garbage collection
- Learn how to use Go's advanced development tools

"Go is unique and even experienced programmers have to unlearn a few things and think differently about software. *Learning Go* does a good job of working through the big features of the language while pointing out idiomatic code, pitfalls, and design patterns along the way."

—Aaron Schlesinger
Senior Engineer, Microsoft

Jon Bodner is a staff engineer at Datadog, where he leads the effort to simplify onboarding to the company's APM products. He's also the creator of the Proteus data access library. He's been a software engineer, lead developer, and architect for over 25 years.

GO / PROGRAMMING LANGUAGES

US \$65.99 CAN \$82.99

ISBN: 978-1-098-13929-2



Twitter: @oreillymedia
linkedin.com/company/oreilly-media
youtube.com/oreillymedia

Praise for *Learning Go*, Second Edition

“The first edition of *Learning Go* was an excellent starting point for any developer interested in learning Go, and the second edition is even better. This book is thorough without being monotonous, which is perfect for introducing newcomers to the Go ecosystem.”

—Jonathan Hall, *Go Developer and Content Creator*

"*Learning Go* does more than teach Go, it teaches good idiomatic Go. It is the perfect book for programmers familiar with other languages who want to learn Go.”

—Chris Hines, *Senior Principal Software Engineer, Comcast*

Praise for *Learning Go*, First Edition

“Go is unique and even experienced programmers have to unlearn a few things and think differently about software. *Learning Go* does a good job of working through the big features of the language while pointing out idiomatic code, pitfalls, and design patterns along the way.”

—Aaron Schlesinger, Senior Engineer, Microsoft

“Jon has been a critical voice in the Go community for many years and we have been strongly benefitted from his talks and articles. With *Learning Go*, Jon has written the programmers’ guide to learning Go. It strikes just the right balance of giving a good overview of what you need to know without rehashing well understood concepts from other languages.”

—Steve Francia, Go language product lead, Google,
and author of *Hugo*, *Cobra*, and *Viper*

“Bodner gets Go. In clear, lively prose, he teaches the language from its basics to advanced topics like reflection and C interop. He demonstrates through numerous examples how to write *idiomatic* Go, with its emphasis on clarity and simplicity.

He also takes the time to explain the underlying concepts that can affect your program’s behavior, like the effects of pointers on memory layout and garbage collection. Beginners who read this book will come up to speed quickly, and even experienced Go programmers are likely to learn something.”

—Jonathan Amsterdam,
Software Engineer on the Go team at Google

“*Learning Go* is the essential introduction to what makes the Go programming language unique as well as the design patterns and idioms that make it so powerful. Jon Bodner manages to connect the fundamentals of the language to Go’s philosophy, guiding readers to write Go the way it was meant to be written.”

—*Robert Liebowitz, Software Engineer at Morning Consult*

“Jon wrote a book that does more than just reference Go; it provides an idiomatic and practical understanding of the language. Jon’s industry experience is what drives this book, and it will help those looking to be immediately productive in the language.”

—*William Kennedy, Managing Partner at Ardan Labs*

SECOND EDITION

Learning Go

*An Idiomatic Approach to
Real-World Go Programming*

Jon Bodner

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Learning Go

by Jon Bodner

Copyright © 2024 Jon Bodner. All rights reserved.

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://oreilly.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Acquisitions Editor: Suzanne McQuade

Developmental Editor: Rita Fernando

Production Editor: Beth Kelly

Copyeditor: Kim Cofer

Proofreader: Sharon Wilkey

Indexer: Sue Klefstad

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Kate Dullea

March 2021: First Edition

January 2024: Second Edition

Revision History for the Second Edition

2024-01-11: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=0636920787686> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Learning Go*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author, and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-098-13929-2

[LSI]

Table of Contents

Preface	xvii
1. Setting Up Your Go Environment	1
Installing the Go Tools	1
Troubleshooting Your Go Installation	2
Go Tooling	2
Your First Go Program	3
Making a Go Module	3
go build	4
go fmt	5
go vet	7
Choose Your Tools	8
Visual Studio Code	8
GoLand	9
The Go Playground	10
Makefiles	12
The Go Compatibility Promise	13
Staying Up-to-Date	14
Exercises	14
Wrapping Up	15
2. Predeclared Types and Declarations	17
The Predeclared Types	17
The Zero Value	17
Literals	18
Booleans	19
Numeric Types	20
A Taste of Strings and Runes	25

Explicit Type Conversion	26
Literals Are Untyped	27
var Versus :=	28
Using const	30
Typed and Untyped Constants	32
Unused Variables	32
Naming Variables and Constants	33
Exercises	35
Wrapping Up	36
3. Composite Types.....	37
Arrays—Too Rigid to Use Directly	37
Slices	39
len	41
append	41
Capacity	42
make	43
Emptying a Slice	44
Declaring Your Slice	45
Slicing Slices	46
copy	49
Converting Arrays to Slices	50
Converting Slices to Arrays	51
Strings and Runes and Bytes	52
Maps	56
Reading and Writing a Map	57
The comma ok Idiom	58
Deleting from Maps	59
Emptying a Map	59
Comparing Maps	59
Using Maps as Sets	60
Structs	61
Anonymous Structs	63
Comparing and Converting Structs	64
Exercises	65
Wrapping Up	66
4. Blocks, Shadows, and Control Structures.....	67
Blocks	67
Shadowing Variables	68
if	71
for, Four Ways	72

The Complete for Statement	72
The Condition-Only for Statement	73
The Infinite for Statement	74
break and continue	75
The for-range Statement	76
Labeling Your for Statements	82
Choosing the Right for Statement	83
switch	84
Blank Switches	87
Choosing Between if and switch	89
goto—Yes, goto	89
Exercises	92
Wrapping Up	92
5. Functions.....	93
Declaring and Calling Functions	93
Simulating Named and Optional Parameters	94
Variadic Input Parameters and Slices	95
Multiple Return Values	96
Multiple Return Values Are Multiple Values	97
Ignoring Returned Values	97
Named Return Values	98
Blank Returns—Never Use These!	99
Functions Are Values	100
Function Type Declarations	103
Anonymous Functions	103
Closures	105
Passing Functions as Parameters	107
Returning Functions from Functions	108
defer	109
Go Is Call by Value	114
Exercises	116
Wrapping Up	117
6. Pointers.....	119
A Quick Pointer Primer	119
Don't Fear the Pointers	123
Pointers Indicate Mutable Parameters	125
Pointers Are a Last Resort	129
Pointer Passing Performance	130
The Zero Value Versus No Value	131
The Difference Between Maps and Slices	131

Slices as Buffers	135
Reducing the Garbage Collector’s Workload	136
Tuning the Garbage Collector	139
Exercises	141
Wrapping Up	141
7. Types, Methods, and Interfaces.....	143
Types in Go	143
Methods	144
Pointer Receivers and Value Receivers	145
Code Your Methods for nil Instances	148
Methods Are Functions Too	149
Functions Versus Methods	150
Type Declarations Aren’t Inheritance	150
Types Are Executable Documentation	151
iota Is for Enumerations—Sometimes	152
Use Embedding for Composition	154
Embedding Is Not Inheritance	156
A Quick Lesson on Interfaces	157
Interfaces Are Type-Safe Duck Typing	158
Embedding and Interfaces	162
Accept Interfaces, Return Structs	162
Interfaces and nil	164
Interfaces Are Comparable	165
The Empty Interface Says Nothing	166
Type Assertions and Type Switches	167
Use Type Assertions and Type Switches Sparingly	170
Function Types Are a Bridge to Interfaces	173
Implicit Interfaces Make Dependency Injection Easier	174
Wire	178
Go Isn’t Particularly Object-Oriented (and That’s Great)	178
Exercises	178
Wrapping Up	179
8. Generics.....	181
Generics Reduce Repetitive Code and Increase Type Safety	181
Introducing Generics in Go	184
Generic Functions Abstract Algorithms	187
Generics and Interfaces	188
Use Type Terms to Specify Operators	190
Type Inference and Generics	193
Type Elements Limit Constants	193

Combining Generic Functions with Generic Data Structures	194
More on comparable	196
Things That Are Left Out	198
Idiomatic Go and Generics	199
Adding Generics to the Standard Library	201
Future Features Unlocked	201
Exercises	201
Wrapping Up	202
9. Errors.....	203
How to Handle Errors: The Basics	203
Use Strings for Simple Errors	205
Sentinel Errors	205
Errors Are Values	208
Wrapping Errors	210
Wrapping Multiple Errors	212
Is and As	214
Wrapping Errors with defer	217
panic and recover	218
Getting a Stack Trace from an Error	220
Exercises	221
Wrapping Up	221
10. Modules, Packages, and Imports.....	223
Repositories, Modules, and Packages	223
Using go.mod	224
Use the go Directive to Manage Go Build Versions	225
The require Directive	226
Building Packages	227
Importing and Exporting	227
Creating and Accessing a Package	227
Naming Packages	230
Overriding a Package's Name	230
Documenting Your Code with Go Doc Comments	231
Using the internal Package	234
Avoiding Circular Dependencies	235
Organizing Your Module	236
Gracefully Renaming and Reorganizing Your API	238
Avoiding the init Function if Possible	239
Working with Modules	240
Importing Third-Party Code	240
Working with Versions	245

Minimal Version Selection	247
Updating to Compatible Versions	248
Updating to Incompatible Versions	248
Vendoring	250
Using pkg.go.dev	251
Publishing Your Module	251
Versioning Your Module	252
Overriding Dependencies	254
Retracting a Version of Your Module	255
Using Workspaces to Modify Modules Simultaneously	255
Module Proxy Servers	259
Specifying a Proxy Server	259
Using Private Repositories	260
Additional Details	260
Exercises	261
Wrapping Up	261
11. Go Tooling.....	263
Using go run to Try Out Small Programs	263
Adding Third-Party Tools with go install	264
Improving Import Formatting with goimports	266
Using Code-Quality Scanners	267
staticcheck	268
revive	269
golangci-lint	270
Using govulncheck to Scan for Vulnerable Dependencies	272
Embedding Content into Your Program	274
Embedding Hidden Files	277
Using go generate	278
Working with go generate and Makefiles	281
Reading the Build Info Inside a Go Binary	281
Building Go Binaries for Other Platforms	283
Using Build Tags	284
Testing Versions of Go	285
Using go help to Learn More About Go Tooling	286
Exercises	286
Wrapping Up	286
12. Concurrency in Go.....	287
When to Use Concurrency	287
Goroutines	289
Channels	291

Reading, Writing, and Buffering	291
Using for-range and Channels	292
Closing a Channel	292
Understanding How Channels Behave	293
select	294
Concurrency Practices and Patterns	297
Keep Your APIs Concurrency-Free	297
Goroutines, for Loops, and Varying Variables	298
Always Clean Up Your Goroutines	299
Use the Context to Terminate Goroutines	300
Know When to Use Buffered and Unbuffered Channels	301
Implement Backpressure	302
Turn Off a case in a select	304
Time Out Code	304
Use WaitGroups	306
Run Code Exactly Once	308
Put Your Concurrent Tools Together	309
When to Use Mutexes Instead of Channels	313
Atomics—You Probably Don’t Need These	317
Where to Learn More About Concurrency	317
Exercises	317
Wrapping Up	318
13. The Standard Library.....	319
io and Friends	319
time	324
Monotonic Time	326
Timers and Timeouts	327
encoding/json	327
Using Struct Tags to Add Metadata	327
Unmarshaling and Marshaling	329
JSON, Readers, and Writers	330
Encoding and Decoding JSON Streams	331
Custom JSON Parsing	332
net/http	336
The Client	336
The Server	337
ResponseController	342
Structured Logging	344
Exercises	347
Wrapping Up	347

14. The Context.....	349
What Is the Context?	349
Values	352
Cancellation	358
Contexts with Deadlines	363
Context Cancellation in Your Own Code	367
Exercises	368
Wrapping Up	369
15. Writing Tests.....	371
Understanding the Basics of Testing	371
Reporting Test Failures	373
Setting Up and Tearing Down	373
Testing with Environment Variables	376
Storing Sample Test Data	376
Caching Test Results	377
Testing Your Public API	377
Using go-cmp to Compare Test Results	378
Running Table Tests	380
Running Tests Concurrently	382
Checking Your Code Coverage	384
Fuzzing	386
Using Benchmarks	393
Using Stubs in Go	397
Using httptest	402
Using Integration Tests and Build Tags	405
Finding Concurrency Problems with the Data Race Detector	406
Exercises	408
Wrapping Up	408
16. Here Be Dragons: Reflect, Unsafe, and Cgo.....	409
Reflection Lets You Work with Types at Runtime	410
Types, Kinds, and Values	411
Make New Values	415
Use Reflection to Check If an Interface's Value Is nil	417
Use Reflection to Write a Data Marshaler	417
Build Functions with Reflection to Automate Repetitive Tasks	422
You Can Build Structs with Reflection, but Don't	423
Reflection Can't Make Methods	424
Use Reflection Only if It's Worthwhile	424
unsafe Is Unsafe	425
Using Sizeof and Offsetof	426

Using unsafe to Convert External Binary Data	428
Accessing Unexported Fields	432
Using unsafe Tools	433
Cgo Is for Integration, Not Performance	433
Exercises	438
Wrapping Up	438
Index.....	441

Preface

In the preface of the first edition, I wrote:

My first choice for a book title was *Boring Go* because, properly written, Go is boring....

Boring does not mean trivial. Using Go correctly requires an understanding of how its features are intended to fit together. While you can write Go code that looks like Java or Python, you're going to be unhappy with the result and wonder what all the fuss is about. That's where this book comes in. It walks through the features of Go, explaining how to best use them to write idiomatic code that can grow.

Go remains a small language with a small feature set. It still lacks inheritance, aspect-oriented programming, function overloading, operator overloading, pattern matching, named parameters, exceptions, and many additional features that complicate other languages. So why does a book on a boring language need an update?

There are a few reasons for this edition. First, just as boring doesn't mean *trivial*, it also does not mean *unchanging*. Over the past three years, new features, tools, and libraries have arrived. Improvements like structured logging, fuzzing, workspaces, and vulnerability checking help Go developers create reliable, lasting, maintainable code. Now that Go developers have several years of experience with generics, the standard library has started to include type constraints and generic functions to reduce repetitive code. Even the unsafe package has been updated to make it a little safer. Go developers need a resource that explains how to best use these new features.

Secondly, some aspects of Go weren't done justice by the first edition. The introductory chapter didn't flow as well as I'd like. The rich Go tool ecosystem wasn't explored. And first-edition readers asked for exercises and additional sample code. This edition attempts to address those limitations.

Finally, the Go team has introduced something new, and, dare I say, *exciting*. There's now a strategy that allows Go to keep the backward compatibility required for long-term software engineering projects while providing the ability to introduce

backward-breaking changes to address long-standing design flaws. The new for loop variable scoping rules introduced in Go 1.22 are the first feature to take advantage of this approach.

Go is still boring, it's still fantastic, and it's better than ever. I hope you enjoy this second edition.

Who Should Read This Book

This book is targeted at developers who are looking to pick up a second (or fifth) language. The focus is on people who are new to Go. This ranges from those who don't know anything about Go other than it has a cute mascot, to those who have already worked through a Go tutorial or even written some Go code. The focus for *Learning Go* isn't just how to write programs in Go; it's how to write Go *idiomatically*. More experienced Go developers can find advice on how to best use the newer features of the language. The most important thing is that the reader wants to learn how to write Go code that looks like Go.

Experience is assumed with the tools of the developer trade, such as version control (preferably Git) and IDEs. Readers should be familiar with basic computer science concepts like concurrency and abstraction, as the book explains how they work in Go. Some of the code examples are downloadable from GitHub, and dozens more can be tried out online on The Go Playground. While an internet connection isn't required, it is helpful when reviewing executable examples. Since Go is often used to build and call HTTP servers, some examples assume familiarity with basic HTTP concepts.

While most of Go's features are found in other languages, Go makes different trade-offs, so programs written in it have a different structure. *Learning Go* starts by looking at how to set up a Go development environment, and then covers variables, types, control structures, and functions. If you are tempted to skip over this material, resist the urge and take a look. It is often the details that make your Go code idiomatic. Some of what seems obvious at first glance might actually be subtly surprising when you think about it in depth.

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.