

# **FOR THE LOVE OF**

The Go logo is rendered in red, consisting of three horizontal bars above a large, hollow circular letter 'G'.

**“Personable, human, and funny”**

—Kevin Cunningham



**JOHN ARUNDEL**

# Contents

<b>Praise for <i>For the Love of Go</i></b>	<b>9</b>
<b>Introduction</b>	<b>10</b>
What's this . . . . .	10
What you'll need . . . . .	10
Where to find the code examples . . . . .	11
What you'll learn . . . . .	11
The love of Go . . . . .	12
Programming with confidence . . . . .	12
What makes this book different? . . . . .	13
How to use this book . . . . .	13
<b>1. Testing times</b>	<b>14</b>
Creating a new project . . . . .	14
Creating Go files . . . . .	15
Running the tests . . . . .	16
Formatting code with gofmt . . . . .	17
Fixing format with gofmt -w . . . . .	18
Functions in Go . . . . .	19
A failing test . . . . .	19
The testing package . . . . .	21
The signature of test functions . . . . .	22
The test body . . . . .	22
The function under test . . . . .	23
if statements . . . . .	24
Conditional expressions . . . . .	24
Takeaways . . . . .	25
<b>2. Go forth and multiply</b>	<b>27</b>
Designing a new feature, guided by tests . . . . .	27
Testing a null implementation . . . . .	28
Writing the real implementation . . . . .	30
Introducing test cases . . . . .	30
A slice of test cases . . . . .	31
Looping over test cases . . . . .	32
A different kind of problem: division . . . . .	33
Error values in Go . . . . .	33
Test behaviours, not functions . . . . .	34
Testing valid input . . . . .	35

Receiving the error value . . . . .	36
Takeaways . . . . .	37
<b>3. Errors and expectations</b>	<b>39</b>
Returning an error value . . . . .	39
Testing the invalid input case . . . . .	41
Detecting invalid input . . . . .	42
Constructing error values . . . . .	42
The structure of tests . . . . .	43
The development process . . . . .	43
Comparing floating-point values . . . . .	43
A Sqrt function . . . . .	45
Running programs . . . . .	46
The main package . . . . .	46
The go run command . . . . .	46
The go build command . . . . .	47
Takeaways . . . . .	48
<b>4. Happy Fun Books</b>	<b>50</b>
Types . . . . .	50
Variables and values . . . . .	51
Type checking . . . . .	52
More types . . . . .	53
Zero values and default values . . . . .	54
Introducing structs . . . . .	54
Type definitions . . . . .	55
Exported identifiers . . . . .	56
The core package . . . . .	56
The very first test . . . . .	56
Testing the core struct type . . . . .	57
Struct literals . . . . .	57
Assigning a struct literal . . . . .	58
The unfaillable test . . . . .	58
Takeaways . . . . .	59
<b>5. Story time</b>	<b>61</b>
User stories . . . . .	61
What are the core stories? . . . . .	62
The first story . . . . .	62
Struct variables . . . . .	63
The short declaration form . . . . .	63
Referencing struct fields . . . . .	63
Writing the test . . . . .	64
Getting to a failing test . . . . .	65
Assigning to struct fields . . . . .	65
Implementing Buy . . . . .	66

Test coverage . . . . .	67
Test-last development . . . . .	67
Uncovering a problem . . . . .	69
“Covered” versus “tested” . . . . .	70
Juking the stats . . . . .	71
Takeaways . . . . .	71
<b>6. Slicing &amp; dicing</b>	<b>73</b>
Slices . . . . .	73
Slice variables . . . . .	73
Slice literals . . . . .	74
Slice indexes . . . . .	74
Slice length . . . . .	74
Modifying slice elements . . . . .	75
Appending to slices . . . . .	75
A collection of books . . . . .	75
Setting up the world . . . . .	75
Comparing slices (and other things) . . . . .	76
Unique identifiers . . . . .	79
Getting to a failing test . . . . .	79
Finding a book by ID . . . . .	81
Crime doesn’t pay . . . . .	82
Takeaways . . . . .	83
<b>7. Map mischief</b>	<b>85</b>
Introducing the map . . . . .	85
Adding a new element to a map . . . . .	87
Accessing element fields . . . . .	87
Updating elements . . . . .	88
Non-existent keys . . . . .	88
Checking if a value is in the map . . . . .	89
Returning an error . . . . .	89
Testing the invalid input case . . . . .	90
Updating GetAllBooks . . . . .	91
Sorting slices . . . . .	94
Function literals . . . . .	95
Takeaways . . . . .	96
<b>8. Objects behaving badly</b>	<b>98</b>
Objects . . . . .	98
Doing computation with objects . . . . .	99
Testing NetPriceCents . . . . .	100
Methods . . . . .	101
Defining methods . . . . .	101
Methods on non-local types . . . . .	102
Creating custom types . . . . .	103

More types and methods . . . . .	104
Creating a Catalog type . . . . .	105
Takeaways . . . . .	108
<b>9. Wrapper's delight</b>	<b>109</b>
The <code>strings.Builder</code> . . . . .	109
Creating a type based on <code>strings.Builder</code> . . . . .	111
Wrapping <code>strings.Builder</code> with a struct . . . . .	112
A puzzle about function parameters . . . . .	114
Parameters are passed by value . . . . .	115
Creating a pointer . . . . .	116
Declaring pointer parameters . . . . .	116
What can we do with pointers? . . . . .	117
The <code>*</code> operator . . . . .	117
Nil desperandum . . . . .	117
Pointer methods . . . . .	118
Takeaways . . . . .	119
<b>10. Very valid values</b>	<b>120</b>
Validating methods . . . . .	120
Automatic dereferencing . . . . .	122
Always valid fields . . . . .	123
Unexported fields and <code>cmp.Equal</code> . . . . .	127
Always valid structs . . . . .	127
A set of valid values . . . . .	130
Using a map to represent a set . . . . .	131
Defining constants . . . . .	132
Referring to constants . . . . .	132
Standard library constants . . . . .	132
When the values don't matter . . . . .	133
Introducing <code>iota</code> . . . . .	133
Takeaways . . . . .	136
<b>11: Opening statements</b>	<b>138</b>
Statements . . . . .	139
Declarations . . . . .	139
What is assignment? . . . . .	139
Short variable declarations . . . . .	140
Assigning more than one value . . . . .	140
The blank identifier . . . . .	141
Increment and decrement statements . . . . .	142
<code>if</code> statements . . . . .	142
The happy path . . . . .	143
<code>else</code> branches . . . . .	144
Early return . . . . .	145
Conditional expressions . . . . .	145

And, or, and not . . . . .	146
bool variables . . . . .	147
Compound if statements . . . . .	147
Takeaways . . . . .	149
<b>12: Switch which?</b>	<b>150</b>
The switch statement . . . . .	150
Switch cases . . . . .	151
The default case . . . . .	151
Switch expressions . . . . .	151
Exiting a switch case early with break . . . . .	152
Loops . . . . .	153
The for keyword . . . . .	153
Forever loops . . . . .	153
Using range to loop over collections . . . . .	154
Receiving index and element values from range . . . . .	154
Conditional for statements . . . . .	155
Init and post statements . . . . .	156
range over integer . . . . .	157
Jumping to the next element with continue . . . . .	157
Exiting loops with break . . . . .	158
Controlling nested loops with labels . . . . .	159
Takeaways . . . . .	160
<b>13: Fun with functions</b>	<b>161</b>
Declaring functions . . . . .	161
Parameter lists . . . . .	162
Result lists . . . . .	162
The function body . . . . .	163
Calling a function . . . . .	163
Using result values . . . . .	164
Return statements . . . . .	164
Function values . . . . .	166
Function literals . . . . .	167
Closures . . . . .	168
Cleaning up resources . . . . .	169
The defer keyword . . . . .	170
Multiple defers . . . . .	170
Named result parameters . . . . .	171
Naked returns considered harmful . . . . .	171
Modifying result parameters after exit . . . . .	172
Deferring a function literal . . . . .	172
Deferring a closure . . . . .	173
Variadic functions . . . . .	173
Takeaways . . . . .	174

<b>14: Building blocks</b>	<b>177</b>
Compilation . . . . .	177
The main package . . . . .	178
The main function . . . . .	178
The init function . . . . .	178
Alternatives to init . . . . .	179
Building an executable . . . . .	179
The executable binary file . . . . .	180
Cross-compilation . . . . .	181
Exiting . . . . .	181
Takeaways . . . . .	182
<b>15. The Tao of Go</b>	<b>184</b>
Kindness . . . . .	184
Simplicity . . . . .	185
Humility . . . . .	186
Not striving . . . . .	187
The love of Go . . . . .	188
<b>About this book</b>	<b>189</b>
Who wrote this? . . . . .	189
Feedback . . . . .	189
Join my Go Club . . . . .	190
Video course . . . . .	190
The Power of Go: Tools . . . . .	190
The Power of Go: Tests . . . . .	190
Know Go: Generics . . . . .	191
Further reading . . . . .	191
Credits . . . . .	192
<b>Acknowledgements</b>	<b>193</b>
<b>A sneak preview</b>	<b>194</b>
<b>1. Packages</b>	<b>195</b>
The universal library . . . . .	195
Packages are a force multiplier . . . . .	196
The universal Go library is huge . . . . .	196
Sharing our code benefits us all . . . . .	197
Writing packages, not programs . . . . .	198
Command-line tools . . . . .	198
Zen mountaineering . . . . .	199
Guided by tests . . . . .	200
Building a hello package . . . . .	200
The structure of a test . . . . .	201
Tests are bug detectors . . . . .	202

So when should a test fail? . . . . .	202
Where does <code>fmt.Println</code> print to? . . . . .	203
Meet <code>bytes.Buffer</code> , an off-the-shelf <code>io.Writer</code> . . . . .	204
Failure standards . . . . .	205
Implementing <code>hello</code> , guided by tests . . . . .	205
We start with a failing test . . . . .	205
Creating a module . . . . .	206
One folder, one package . . . . .	206
A null implementation . . . . .	207
The real implementation . . . . .	207
The naming of tests . . . . .	208
Refactoring to use our new package . . . . .	208
Going further . . . . .	209

# Praise for *For the Love of Go*

*A great way to dive into Go!*

—Max VelDink

*One of the best technical books I have read in a very long time.*

—Paul Watts

*John is both a superb engineer and, just as importantly, an excellent teacher / communicator.*

—Luke Vidler

*A fantastic example of good technical writing. Clear, concise and easily digestible.*

—Michael Duffy

*This book's writing style feels as if John is speaking to you in person and helping you along the way.*

—@rockey5520

*Very well written, friendly, and informative.*

—Chris Doyle

*John's writing is personable, human and funny—his examples are realistic and relevant. The test-driven instruction teaches Go in a deep, meaningful and engaging way.*

—Kevin Cunningham

*The book takes a very easy-to-follow, step-by-step approach to Go. The writing is very friendly and accessible. This would probably be my pick for the absolute beginner to computer programming who wants to learn Go.*

—Jonathan Hall