Beginning Go Programming

Build Reliable and Efficient Applications with Go

Rumeel Hussain
Maryam Zulfiqar

Apress®

Beginning Go Programming

Build Reliable and Efficient Applications with Go

Rumeel Hussain
Maryam Zulfiqar

Apress®

Beginning Go Programming: Build Reliable and Efficient Applications with Go

Rumeel Hussain
Dubai, United Arab Emirates

Maryam Zulfiqar
Lahore, Pakistan

ISBN-13 (pbk): 978-1-4842-8857-3
ISBN-13 (electronic): 978-1-4842-8858-0
https://doi.org/10.1007/978-1-4842-8858-0

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: James Markham
Coordinating Editor: Mark Powers
Copy Editor: Kezia Endsley

Cover designed by eStudioCalamar

Cover image by Benjamin Davies on Unsplash (www.unsplash.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub (https://github.com/Apress). For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

This book is dedicated to Rumeel's late mother and Abdullah Hassan.

Table of Contents

About the Authors

Rumeel Hussain has a bachelor's degree in computer science and is presently working as a Blockchain Solution Architect at BNB Chain, supporting the development and growth of the BNB Chain ecosystem. He is an information technology enthusiast with more than five years of experience leading and implementing blockchain applications and architectures, analyzing and refactoring modern programming languages like Go, troubleshooting cloud infrastructure, and assessing security risks. His current work is focused on leveraging blockchain technology and crypto to achieve the full potential of Web3 applications.

Maryam Zulfiqar has four years of research experience and has a master's degree in computer science. She is currently working as a Tech Martian at BNB Chain. She also works as a senior researcher and developer. She is passionate about developer education, especially about sharing her knowledge on topics that are "the talk of the town" in the technology field. She has also worked in researcher and teamlead roles for HEC-funded projects targeted at community growth and welfare.

About the Technical Reviewer

Fabio Claudio Ferracchiati is a senior consultant and a senior analyst/developer using Microsoft technologies. He works for BluArancio (www.bluarancio.com). He is a Microsoft Certified Solution Developer for .NET, a Microsoft Certified Application Developer for .NET, a Microsoft Certified Professional, and a prolific author and technical reviewer. Over the past ten years, he's written articles for Italian and international magazines and coauthored more than ten books on a variety of computer topics.

Preface

Google designed the Go language to be comprehensive yet be powerful enough to tackle big problems. It is a multi-paradigm programming language with built-in features for concurrent programming. It enables developers to easily build software that is simple, reliable, and efficient. This book is designed to provide knowledge to beginners and help them start developing great Go-based applications.

In this book, we cover the basics of the Go programming language and provide you with hands-on working experience through clear examples and recipes. From basic syntax formulation to providing insights into using the Go language to build concurrent programs and applications like HTTP servers and communication with REST APIs, this book covers it all.

This is a practical hands-on guide through which you will learn how to write Go code using clear examples that demonstrate the language in action. You'll also learn tips and tricks about the Go programming language for an efficient use.

CHAPTER 1

Introduction

Developed by Google in 2007, Go (aka GoLang) is a programming language designed with simplicity and speed as its focus. The design goals of Go were to create a simple and readable syntax of a dynamically-typed, high-level language programming language like Python, but also have the stability and efficiency of a statically-typed low-level language like C/C++. Supported by Google, Go is an open-source programming language and is easier to learn and get started with. It has built-in support for concurrency, a robust standard library, type safety, multiple built-in types, dynamic-typing capability, garbage collection, and several other advanced features, such as key-value maps and variable-length arrays. Go has the power to let you leverage the might that multi-core processors have to offer, which results in faster-running programs.

Is GoLang Static-Typed or Compiled?

Any programmer's natural concern about a programming language is whether the language is compiled or interpreted. JavaScript is an example of an interpreted language. Web browsers can directly read JavaScript source code and execute it at runtime. There is no pre-computation involved. Even though some interpreted environments can produce intermediate formats such as bytecode, there is no pre-computation step involved.

R. Hussain and M. Zulfiqar, *Beginning Go Programming*,
https://doi.org/10.1007/978-1-4842-8858-0_1

Compiled Programming Language

In contrast to an interpreted language, the source code of a compiled language is transformed into an operating system-specific format. Like C and C++, Go is a compiled language. However, compared to Java, whose source code is compiled into a format capable of running on different operating systems, Go's source code is compiled into an operating system-specific format. That means the code is executable only on the operating system for which it was compiled.

Statically-Typed Language

Go is a statically-typed language. That means its variables must have specific types. Nevertheless, it is not always mandatory to declare variables explicitly. The compiler can infer the types; however, at compilation time, the types are always known.

The capability to run a source code file without recompiling it can sometimes lead programmers to think that Go is an interpreted language. The real magic is in the background, where the source code is compiled into a temporary executable form. The compilation tool, "Go," is required to build applications delivered to users. This is because the compiled executable is in a format that is operating-system-specific. Each Go application consists of a statically-linked runtime. During the compilation process of the application, a runtime component is packaged with the executable format. This is why the size of applications built using the Go language is comparatively larger than the source code. No external virtual machine is involved, so it is necessary to include a runtime package.

Is Go an Object-Oriented Programming Language?

Go is not an object-oriented programming language. Nevertheless, you will find certain important object-oriented features in the Go language.

- Custom interfaces can be defined. In Go, an interface can be thought of as a contract that lays out a set of functions that must be implemented by any program that implements this interface.
- Custom types can be defined. Custom types can also have their own methods (functions). It's worth noting that in Go, practically everything is a type, and each type implements at least one interface.
- Custom structs (data structures) can be defined, containing member fields and methods.

Features that Make GoLang the Premium Choice for Programming

Other features of Go that make it popular among leading companies include the following:

- Go supports environment-adopting patterns identical to those offered by dynamic languages. For example, it supports type inference. That means that `intVariable:= 123` is a valid way to declare a type `int` variable named `intVariable` with a value of `123`.
- Go offers much faster compilation times.

- Go supports concurrency. For example, it offers the `select` statement, *goroutines,* which are similar to lightweight processes, and channels.
- Go programs are much simpler, more concise, and comparatively safer.
- Go supports type embedding and interfaces.
- Without the need for external dependencies, Go supports the production of statically-linked native binaries.

Intentionally Excluded Features from GoLang

In order to keep Go simple and concise, some of the features that are usually supported by similar programming languages are excluded from the Go language. The major reason for the omission is that Go's creators believed that all these factors make programming languages increasingly hard to comprehend and bug-prone. Some of the features that are excluded intentionally are as follows:

- Type inheritance (no classes)
- Generic programming
- Method or operator overloading
- Pointer arithmetic
- Structured exception handling
- Implicit numeric conversions
- Assertions
- Circular dependencies among packages

Go Programs

A Go program can be a minimum of three lines and extend to up to millions of lines. Go programs are written into text files having the `.go` extension, such as `myProgram.go`. Any text editor (like vim, vi, sublime, vs code, etc.) of your choice can be used to write Go programs. For efficient management of dependencies, Go programs are constructed using packages. The traditional compile and link model is used by the Go programming implementation to generate executable binaries.

Summary

The Go programming language was developed by Google in 2007. It was designed as a compiled language with simplicity and speed as its focus. The design goals of Go were to create a simple and readable syntax of a dynamically-typed high-level language programming language but also have the stability and efficiency of a statically-typed low-level language. It is an open-source programming language and is relatively easier to learn and get started with. Go comes with built-in support for concurrency, a robust standard library, type safety, multiple built-in types, dynamic-typing capability, garbage collection, and several other advanced features, including key-value maps and variable-length arrays.

Furthermore, Go has the power to let you leverage the might that multi-core processors have to offer, which results in faster-running programs. One of the important aspects to remember about Go is that it is not an object-oriented programming language. Nevertheless, you do find certain important object-oriented features in the Go language, like custom interfaces, types, and structs (data structures), which can be defined.

Certain features have been excluded from Go to meet its design goals. The major reason for these omissions is that Go's creators believed that all these factors make programming languages increasingly hard to comprehend and bug-prone.

The next chapter takes a dive deep into the programming fundamentals of Go, along with practical recipes that provide hands-on examples of the concepts.

CHAPTER 2

Go Basics

What makes Go an easy-to-learn language is that everything you need to know about a Go program is right on the surface. There is no need to remember any language rules, as they are all in the application code. This chapter covers the basics of the Go programming language, like accepting input, using math operators and packages, managing memory, using different data structures, and understanding the program flow of a Go-based application. It also covers using functions and returning multiple values, as well as reading and writing data to files and handling JSON data.

Ancestors of Go

Go is based on a number of different languages. It was originally designed to be the next generation of the C language. It's capable of performing everything you can do with C, including system programming, application development, and so on.

Go Syntax

Go borrows a lot of its syntax from C and its related languages—C++, C#, Java, and so on. Along with this, it also borrows syntax from Pascal, Modula, Oberon, and other similar languages. One of the biggest attractions of Go is that its programs are very concise because of the precise and

R. Hussain and M. Zulfiqar, *Beginning Go Programming*,
https://doi.org/10.1007/978-1-4842-8858-0_2

comprehensive set of keywords and syntax. In order to be effective with any language, you have to know the certain basic syntax rules of that particular language. The following is a list of some of the most important syntax rules set by the designers of the Go programming language.

1. **Case sensitivity**
    - You must spell *identifiers* like functions, variables, and type names in the exact manner mentioned in the official documentation (`https://go.dev/doc/`).
    - Names for packages and variables use lowercase and mixed case.
    - Methods and fields of a type are initial capped.
    - In Go, initial uppercase characters indicate a special purpose.
        - Equivalent to the `public` keyword in other high-level programming languages like C++, C#, Java, and so on, in Go, an initial uppercase character indicates that the symbol is exported.
        - Similarly, equivalent to the `private` keyword, in Go, a lowercase initial character indicates that the particular method or variable is not exported nor accessible by the rest of the application other than the code block it is declared in.
2. **Semicolons can be skipped**
    - For the sake of reducing the amount of typing a developer has to do, as shown in Listing 2-1, Go eliminates semicolons from the ends of lines. This means that line feeds end the statement and no semicolons are required.

Listing 2-1. Semicolons Are Not Required as End-of-Line Indicators

```
var color [2] string
colors[0] = "black"
colors[1] = "white"
```

Even though the language specification requires semicolons, it is not necessary to type them explicitly. Thanks to Lexer, the software component that parses the code and analyses it, semicolons are added during the compilation process as needed.

The Rule Followed by Lexer for Adding Semicolons

When a statement is determined to be complete and Lexer encounters a line feed, meaning it's the end of the statement and the developer has not explicitly added a semicolon, Lexer will do so.

Note You can't always add line feeds freely or break up statements with extra line feeds, as you can in other languages, because in certain cases they can be misinterpreted by Lexer.

---

3. **Code blocks with braces**
    - Code blocks, i.e., multiple lines of code that are intended to be grouped together, are wrapped within braces. See Listing 2-2.

Listing 2-2. Braces Are Used to Indicate Code Blocks

```
sum := 0
for i :=0; i<10; i++{
  sum += i
}
fmt.Println(sum) //prints '45'
```

In Listing 2-2, the variable named `sum` is declared and then assigned a value of zero. The `for` loop is initialized from 0 to 10 and is used to increment the value of the `sum` by i. After the loop terminates, the value stored in the `sum` variable is printed using a function called `Println` or "print line". `Println` is a built-in function of the package called `FMT`. The code that you want to iterate over is wrapped between two braces. Note that it is important to make sure that the opening brace is on the same line as any preceding statement.

4. **Built-in functions**

   - Go supports a set of several built-in functions that are always available in your code without having to import anything. These functions are members of a special package named `Builtin`. The Go compiler assumes that the `Builtin` package is always imported. Example functions include the following:
     - `len(string)` returns the length of the string.
     - `panic(error)` stops execution and displays an error message.
     - `recover()` manages the behavior of a panicking goroutine.

   For more information, you can visit the official documentation of the builtin package at `https://pkg.go.dev/builtin`.

Installing Go

To install the Go compiler, visit https://go.dev/dl/ and download the binary release suitable for your system. Just run the installer to install the compiler on your system.

In order to verify your installation, on Windows OS, open the command prompt and type `go version`, as illustrated in Figure 2-1. The output shows the version of the Go compiler installed on your system.

```
$ go version
go version go1.17.6 windows/amd64
```

Figure 2-1. *Output showing the installed version of Go*

If you type the `path` command at the command prompt, you can see where all the Go executable commands are saved. If you can see the path to the directory where Go is installed, e.g. `C:\Users\UserName\go\bin`, you are ready to build programs in Go.

To find the directory where Go is installed on macOS, open a terminal window and type `export $PATH`. Upon successful execution, a listing of the directory, indicating the path to the `bin` folder where Go was installed, e.g., `user/local/go/bin`, is shown. In addition, along with the addition of the command to execute Go programs to the directory phrase, a file is added to a path under the root. To ensure that you can run Go programs from anywhere, go to your `home` directory and type `go version`. If everything's working correctly, you should see output indicating the version of the Go language that you're using, as shown in Figure 2-2.

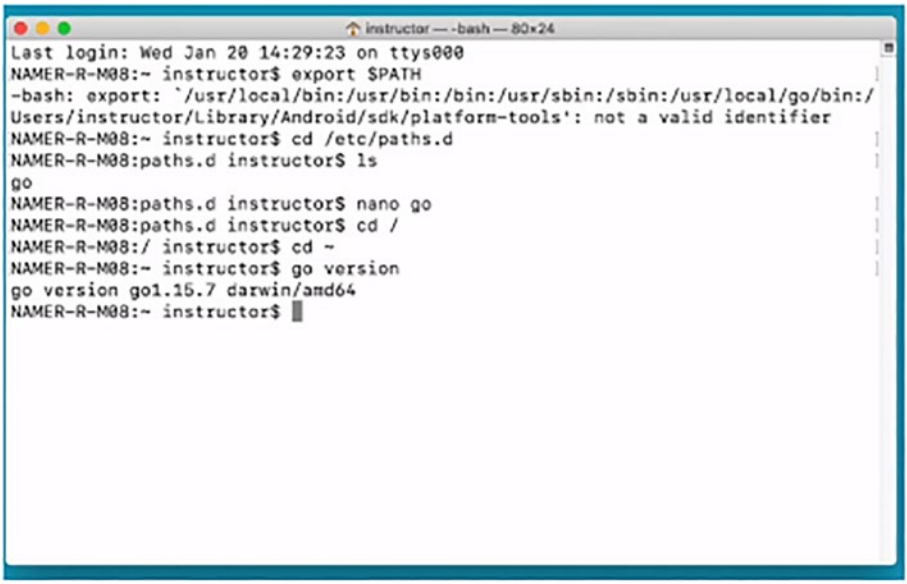

Figure 2-2. *Checking the PATH variable and version to ensure correct installation of Go Compiler*

If you can see the version, you're ready to start programming using the Go language. The first element you'll want to explore is the Go Playground.

Go Playground

One of the fastest ways to start coding and developing Go-based programs is by using the Go Playground (`https://go.dev/play/`), a web-based IDE. As shown in Figure 2-3, without having to install anything, Go Playground allows users to edit, run, and experiment with the Go programming language. When the Run button is clicked, the Go Playground compiles and executes the Go code on Google servers and outputs the result of the execution.

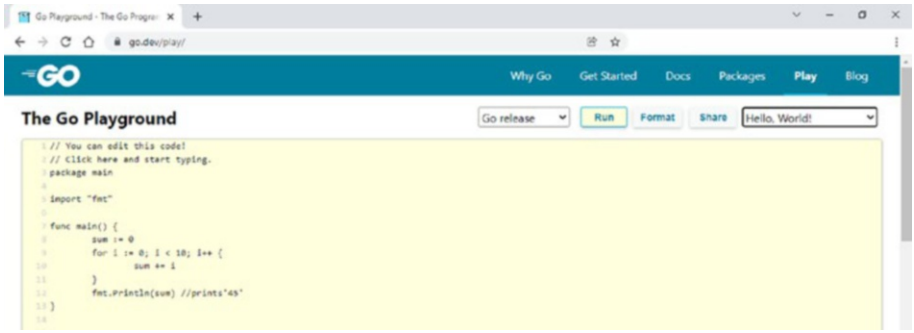

Figure 2-3. *The Go Playground IDE*

The Go Playground is a completely free-to-use service with no limitations. There are no requirements for user registration or licensing fees. Furthermore, it has no limitations on the number of source code files you can work with or the number of times you can run your code. It is a great way to test your Go code without creating or compiling any local source files. Nonetheless, there are other several IDEs available that can be used to develop Go applications.

Developing Go Applications Using IDEs

There isn't a single integrated development environment (IDE) for good programming that's endorsed or even developed by the Go development team. There are, however, many plug-ins for commercial and open-source IDEs that have been created by the Go community or by IDE vendors, which you choose mostly depending on what development environments you're already familiar with. In this book, we use Go Playground and Visual Studio Code to write and run Go programs. Let's jump into how to get started programming Go applications.

Getting Started Programming Go Applications

Once you've installed the Go development tools, you are ready to create as many Go applications as you like. Usually, a Go program is constructed of the following parts:

- Declaring package name
- Importing package(s)
- Declaring and defining function(s)
- Variable(s)
- Expressions and statements
- Comment(s)

Let's Print Hello World!

Prior to learning about the Go programming language's basic building blocks, knowing about the bare minimum structure of Go programs is important. Listing 2-3 illustrates how to print the `"Hello World!"` message on the screen as output. Even though the program is concise and has minimum functionality, it is adequate for understanding Go's program structure.

Listing 2-3. Basic Program that Illustrates Different Parts of a Go Program

```
package main

import (
    "fmt"
)
```