IN THIS CHAPTER

- Understanding why Go is the wave of the future
- Installing Go on your computer
- Working with an integrated development environment
- Writing a Go program and understanding how it works
- Seeing how Go compares to other languages

Chapter 1
Hello, Go!

Go is an open-source programming language — one of the fastest-growing programming languages around — released by Google in 2009. It's a multipurpose programming language specifically designed to build fast, scalable applications.

TECHNICAL STUFF

Go comes from a pretty impressive team of people: Ken Thompson (designer and creator of Unix and C), Rob Pike (cocreator of UTF-8 and Unix format), and Robert Griesemer (a Google engineer). If you're technically inclined, you may want to check out an article called "Go at Google: Language Design in the Service of Software Engineering" (`https://talks.golang.org/2012/splash.article`), which discusses how Go was initially conceived to solve problems at Google.

In this chapter, I explain why learning Go is important for your career, where Go can be used, and how to get started with Go programming.

TIP

Go is often referred to as Golang because of its web address: `https://golang.org`. However, the official name of the language is Go, so that's how I refer to it throughout this book.

Seeing What Learning Go Can Do for You

You can learn many programming languages today, but Go stands out from the others for a few reasons:

- **Go is easy to learn.** Go's syntax makes it a readable language. It has no support for object-oriented programming (OOP), which means you don't have to worry about classes and inheritance and the complexities that come with that.

  
  TECHNICAL STUFF

  Object-oriented programming (OOP) is a programming paradigm that is based on the concept of *objects* (data). Instead of focusing on the functions and logics, OOP organizes software around data, or objects. A key concept in OOP is *classes* (sort of like templates). Suppose you want to display buttons in your application. Instead of writing the code to display each button individually, you can create a class to represent a generic button and use it to create buttons to display in your application. Each button has its own *properties* (characteristics). Using the concept of *inheritance* in OOP, you can create multiple *subclasses* of the button class to create different types of buttons, such as a rounded button, a rectangular button, and so on.

- **Go has fewer features than other programming languages.** You don't have to worry about the best way to solve a problem — there is only one right way to solve a problem in Go. This makes your codebase easy to maintain.
- **Go excels in concurrent programming.** Go's support for *Goroutines* makes it extremely easy to run multiple functions concurrently.


TIP

Go has no support for *generics* (the ability to specify the actual data type until it's actually used), but this may change as the language evolves.

If you still aren't convinced that you should learn Go, perhaps this next bit of news will motivate you: In the Stack Overflow Developer Survey 2019 (`https://insights.stackoverflow.com/survey/2019`), Go developers were the third highest paid in the industry, behind Clojure and F# developers.

Although Go has been around for quite a while (since 2009), only recently did it get wide adoption by developers, thanks to the proliferation of cloud computing and microservices. Today, Go has been widely used by major companies such as Dailymotion, Dropbox, Google, and Uber.

Here are some examples of where Go can be used:

- **Cloud services:** You can build scalable apps using Go on the Google Cloud Platform (GCP).

- **Networking apps:** With Go's support for Goroutines, you can use Go to build distributed servers and application programming interfaces (APIs).
- **Web services:** You can use Go to build scalable and efficient web services.
- **Command-line apps:** Because Go runs on multiple platforms, you can compile the same codebase and target different platforms (such as those running on macOS and Windows).

Installing Go on Your Machine

You're probably very eager to get started with Go programming on your machine, so let's get to it!

The easiest way to install Go is to go to `https://golang.org/doc/install`. This website automatically detects the operating system (OS) you're using and shows you the button to click to download the Go installer (see Figure 1-1).

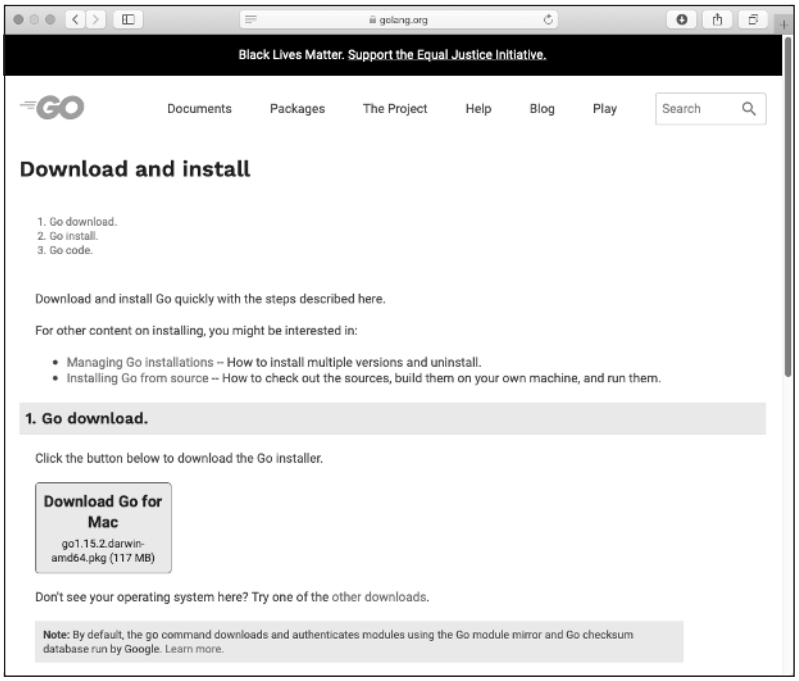

FIGURE 1-1: Downloading the Go installer.

TIP

This book code has been written and tested using Go version 1.15. When you're reading this book, a new version of Go may have been released. In order to ensure that you can follow the examples in this book, I strongly suggest that you install the same version of Go that I've used. You can find it here:

- **macOS:** `https://golang.org/dl/go1.15.8.darwin-amd64.pkg`
- **Windows:** `https://golang.org/dl/go1.15.8.windows-amd64.msi`

TECHNICAL STUFF

If you want to be able to choose the Go installer for each of the supported operating systems (Linux, macOS, and Windows), and even see the source code for Go, go to `https://golang.org/dl/`.

After you've downloaded the Go installer, double-click the installer to start the straightforward installation process. I recommend that you just use the default installation settings — you don't need to change any of those settings.

In the following sections, I show you how to verify that your installation is performed successfully on macOS and Windows.

macOS

On macOS, the Go installer installs the Go distribution in the `/usr/local/go` directory. It also adds the `/usr/local/go/bin` directory to your `PATH` environment variable. You can verify this by entering the following command in the Terminal app (which you can find in the `Applications/Utilities` folder):

```
$ echo $PATH
```

You should see something like the following output (note the added path, highlighted in bold):

```
/Users/weimenglee/opt/anaconda3/bin:/Volumes/SSD/opt/anaco
nda3/condabin:/Users/weimenglee/flutter/bin:/Users/weimeng
lee/go/bin:/Users/weimenglee/.nvm/versions/node/v9.2.0/bin
:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/g
o/bin:/usr/local/share/dotnet:~/.dotnet/tools:/Library/App
le/usr/bin:/Library/Frameworks/Mono.framework/Versions/Cur
rent/Commands
```

TIP

Make sure to restart the Terminal app after you've installed Go in order for the changes to take effect.

To verify that the installation is correct, type the following command in Terminal:

```
$ go version
```

You should see the version of Go installed on your system:

```
go version go1.15.8 darwin/amd64
```

Windows

On Windows, the Go installer installs the Go distribution in the `C:\Go` directory. It also adds the `C:\Go\bin` directory to your `PATH` environment variable. You can verify this by entering the following command in Command Prompt (which you can find by typing **cmd** in the Windows search box):

```
C:\Users\Wei-Meng Lee>path
```

You should see something like the following output (note the added path, highlighted in bold):

```
PATH=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\
Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WIND
OWS\System32\OpenSSH\;C:\Program Files\dotnet\;C:\Program
Files\Microsoft SQL Server\130\Tools\Binn\;C:\Go\bin;
C:\Program Files\Git\cmd;C:\Program Files\Graphviz
 2.44.1\bin;C:\Program Files\CMake\bin;C:\Program
Files\Docker\Docker\resources\bin;C:\ProgramData\DockerDes
ktop\version-bin;C:\Program Files\MySQL\MySQL Shell
8.0\bin\;C:\Users\Wei-Meng Lee\AppData\Local\
Microsoft\WindowsApps;;C:\Users\Wei-Meng Lee\
AppData\Local\Programs\Microsoft VS Code\bin;C:\Users\Wei-
Meng Lee\.dotnet\tools;C:\Users\Wei-Meng Lee\go\bin
```

TIP

Make sure to restart the Command Prompt window after you've installed Go in order for the changes to take effect.

To verify that the installation is correct, type the following command in Command Prompt:

```
C:\Users\Wei-Meng Lee>go version
```

You should now see the version of Go installed on your computer:

```
go version go1.15.8 windows/amd64
```

Using an Integrated Development Environment with Go

To develop applications using Go, you just need a text editor (such as Visual Studio Code, TextEdit on macOS, or even the old trusty NotePad), and you're good to go (pun unintended). However, many developers prefer to use integrated development environments (IDEs) that can help them organize their code, as well as provide debugging support. Here is a partial list of IDEs that work with Go:

- **Visual Studio Code (`https://code.visualstudio.com`):** Visual Studio Code from Microsoft is the mother of all code editors (and my personal favorite). Visual Studio Code is a full-featured code editor that supports almost all programming languages under the sun. Perhaps one of the most useful features of Visual Studio Code is IntelliSense, which helps to complete your statement as you type. It also comes with debugger support and an interactive console, as well as Git integration. Best of all, Visual Studio Code is free and has a very active community of Go developers, allowing you to extend its functionalities through the various plug-ins.
- **GoLand (`www.jetbrains.com/go/`):** GoLand is a cross-platform IDE by JetBrains. It comes with coding assistance, a debugger, an integrated Terminal, and more. GoLand is a commercial IDE, and it has a 30-day trial.
- **The Go Playground (`https://play.golang.org`):** The Go Playground (which isn't really an IDE, but is worth a mention here) is a web service that runs on Go's servers. It receives a Go program, compiles, links, runs it inside a sandbox, and then returns the output. The Go Playground is very useful when you need to test out some Go code quickly using a web browser.


TIP

In this book, I use Visual Studio Code for Go development. To download Visual Studio Code, go to `https://code.visualstudio.com/download`. After you've downloaded and installed Visual Studio Code, launch it, and you should see the screen shown in Figure 1-2.


TIP

In order for Visual Studio Code to recognize your Go language syntax, you also need to install an extension for it. Follow these steps to install the Go extension:

1. **In Visual Studio Code, click the Extensions icon on the Activity Bar (see Figure 1-3).**

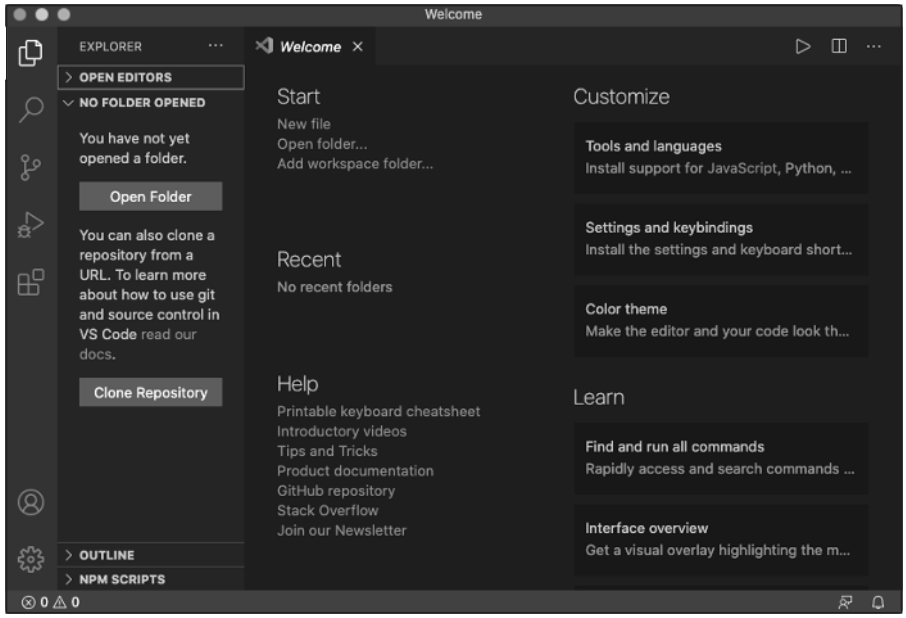

FIGURE 1-2: Launching Visual Studio Code for the first time.

FIGURE 1-3: The Extensions icon is located at the bottom of the Activity Bar.

2. **In the Search box for the Extensions panel, type** Go**.**

   You see a list of the Go extensions available (see Figure 1-4).

3. **Select the first extension and click the Install button on the right.**

   That's it! You're ready to write your first program!

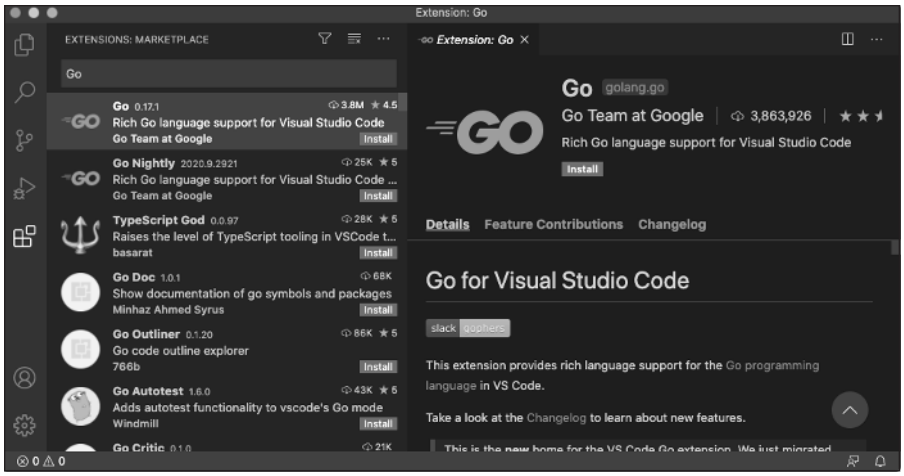

FIGURE 1-4: Searching for Go extensions for Visual Studio Code.

Writing Your First Go Program

To write your first Go program, create a new file in Visual Studio Code by choosing File➪New File. Then enter the following statements (see Figure 1-5):

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, world!")
}
```

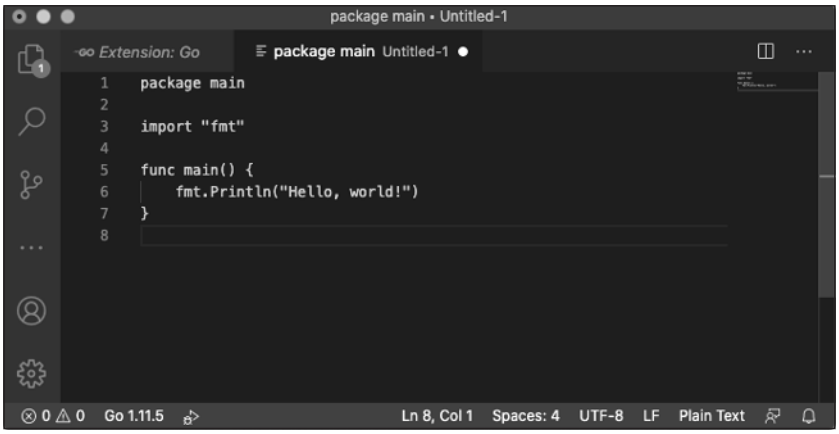

FIGURE 1-5: Writing your first Go program.

When you're done typing, press ⌘+S (Ctrl+S) to save the file. Name the file `main.go`. If this is the first time you're writing a Go program using Visual Studio Code, it may prompt you to download additional plugins for Go. I recommend that you install the plugins.

TIP

For this book, save your files to a folder named with the chapter number you're reading. For example, save `main.go` in a folder named `Chapter 1` in your home directory. On a Mac, that looks like this:

```
~/Chapter 1
    |__main.go
```

In Windows, it looks like this:

```
C:\users\yourname\Chapter 1
    |__main.go
```

After the file is saved, notice that your Go statements will now be color-coded — light blue for keywords such as `package`, `import`, and `func`; orange for strings like `Hello, world!` and `fmt`; and yellow for functions like `main()` and `Println()`.

Compiling and running the program

After you've saved a program, you need to compile it so that you can run it.

You can run the program directly in Visual Studio Code. To do that, launch the Terminal by choosing Terminal➪New Terminal. The Terminal now opens in Visual Studio Code (see Figure 1-6).

Next, change the directory to `Chapter 1`. In macOS, use the following command:

```
$ cd ~/"Chapter 1"
```

In Windows, use the following command:

```
$ cd "C:\users\username\Chapter 1"
```

To compile the `main.go` file, type the following command:

```
$ go build main.go
```

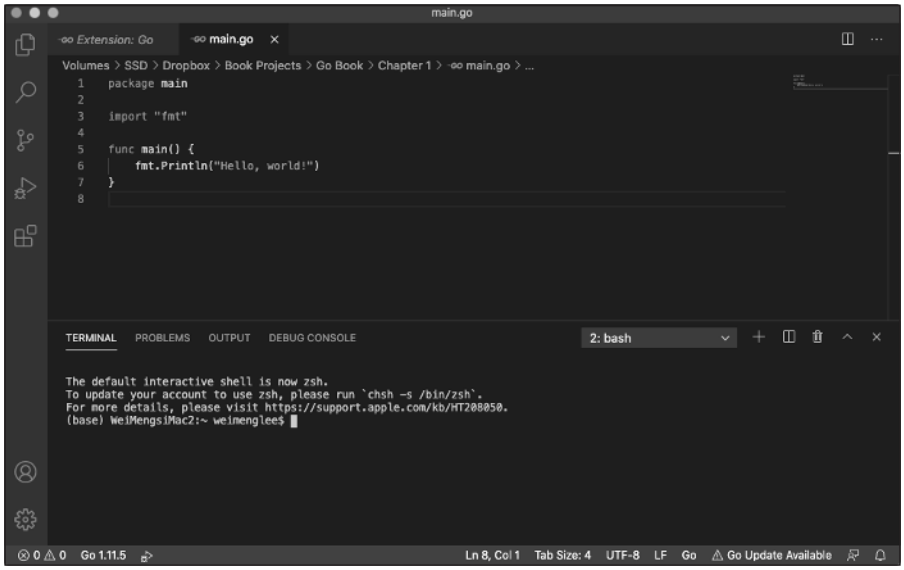

FIGURE 1-6: You can directly access the Terminal in Visual Studio Code.

The preceding command compiles the `main.go` program into an executable file. On macOS, after running the program, you see a file named `main`. To run it, use the following command:

```
$ ./main
Hello, world!
```

If you see the `Hello, world!` string printed, congratulations! You're now officially a Go programmer!

If you're using Windows, the `build` command generates an executable named `main.exe`. To run it, use the following command:

```
C:\users\username\Chapter 1>main
Hello, world!
```

Because you often want to run the program immediately after compiling it, you can use the `run` option to compile and run the program straight away:

```
$ go run main.go
```

Understanding how a Go program works

Now that your Go program works, it's time to understand how. I'll walk you through it line by line. The first line defines the package name for this program:

```
package main
```

Go programs are organized into *packages.* A package is a collection of source files grouped in a single directory. In this example, `main` is the name of the package stored in the `Chapter 1` directory. If you have additional source files (`.go` files), they'll all belong to this `main` package (more on this in later chapters).

TIP

You aren't constrained to using `main` as the package name. You can use any package name you want. But this `main` package name has a special meaning: Packages with the name `main` will contain a function named `main()`, which serves as an entry point for your program to get started. Also, your package name and the name of the file containing it don't need to be the same. I could've named the file `dog.go` instead of `main.go`, and the program would still run (but now the executable would be named `dog` instead of `main`).

The next line imports a package named `fmt`:

```
import "fmt"
```

This package contains various functions to allow you to format and print to the console. It also contains functions to get a user's inputs.

TIP

If you're new to programming, a function is a block of code that's used to perform a single, related task. Turn to Chapter 5 for more about functions.

The next block of code is the main entry point of your program:

```
func main() {

}
```

Because the package name is `main`, this `main()` function will now serve as the starting point for your program to execute.

Finally, the following line calls the `Println()` function from the `fmt` package to print the string `Hello, world!` to the console:

```
fmt.Println("Hello, world!")
```

Making sense of the Go file structure

You should now have a good idea of how the Go program works. Let's dive a little deeper into how Go files are grouped together. As I mention in the previous section, all files in the same directory belong to the same package. So, let's now add another file to the `Chapter 1` directory and name it `show_time.go`. The `Chapter 1` directory should now have one more file:

```
Chapter 1
    |__main.go
    |__show_time.go
```

Populate the `show_time.go` file with the following statements:

```
package main

import (
    "fmt"
    "time"
)

func displayTime() {
    fmt.Println(time.Now())
}
```

Notice that the file is part of the `main` package (the first line reflects that). Also, you can import multiple packages by enclosing them within a pair of parentheses. In this case, we imported the `time` package in addition to the `fmt` package. Finally, we also added a function named `displayTime()`, which displays the current date and time using the `Now()` function from the `time` package.

Because the `displayTime()` function belongs to the `main` package, it can also be called from `main.go`:

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, world!")
    displayTime()
}
```

TIP

You can call functions defined in the same package without needing to import the package.

Because there are now two files in the package (`main`), you don't build the program using a specific filename. Instead, you just need to use the `build` command within the `Chapter 1` directory:

```
$ go build
```

This time around, you'll see that there is a new file named `Chapter_1` (in Windows, you'll see a file named `Chapter_1.exe`) created in the `Chapter 1` directory.

To run it in macOS, type the following in Terminal:

```
$ ./Chapter_1
```

To run it in Windows, type the following command:

```
C:\users\username\Chapter 1>Chapter_1
```

You should now be able to see the output:

```
Hello, world!
2020-10-01 12:01:13.412568 +0800 +08 m=+0.000365532
```

Compiling for multiple operating systems

When you install Go, the Go installer automatically sets up a number of Go environment variables in order for your Go program to work correctly. Specifically, it auto-detects values of the host architecture and OS and sets the `GOHOSTARCH` and `GOHOSTOS` environment variables, respectively. The value of these two variables will be used as the target platform for your program to compile to.

To examine the values of these Go environment variables, use the `env` command:

```
$ go env
GOARCH="amd64"
GOBIN=""
GOCACHE="/Users/weimenglee/Library/Caches/go-build"
GOEXE=""
GOFLAGS=""
GOHOSTARCH="amd64"
```

```
GOHOSTOS="darwin"
GOOS="darwin"
...
...
PKG_CONFIG="pkg-config"
```

To compile your program for another OS, you need to set another two environment variables: `GOOS` and `GOARCH`. These two variables configure the target OS based on the values shown in Table 1-1.

TABLE 1-1 Environment Variables for the Various Operating Systems

Operating Systems	GOOS	GOARCH
macOS	`darwin`	`amd64`
Linux	`linux`	`amd64`
Windows	`windows`	`amd64`

To compile for macOS, use the following command and options:

```
$ GOOS=darwin GOARCH=amd64 go build -o Chapter_1-mac
```

TIP

In the near future, when Go has been ported natively to Apple Silicon, the value of `GOARCH` would be `arm64`.

To compile for the Windows OS, use the following command and options:

```
$ cd ~/"Chapter 1"
$ GOOS=windows GOARCH=amd64 go build -o Chapter_1-windows.exe
```

TIP

The `-o` option (short for *output*) allows you to specify the name of the executable file.

The preceding command compiles the package in the `Chapter 1` folder to run on Windows and save the executable as `Chapter_1-windows.exe`.

To compile for Linux, use the following command and options:

```
$ GOOS=linux GOARCH=amd64 go build -o Chapter_1-linux
```

TIP

If you use Go on the Raspberry Pi, then you should specify `arm64` for `GOARCH`.

If you're running macOS or Linux, you can use the `file` command to examine the various executables created for each platform:

```
$ file Chapter_1-mac
Chapter_1-mac: Mach-O 64-bit executable x86_64

$ file Chapter_1-windows.exe
Chapter_1-windows.exe: PE32+ executable (console) x86-64
  (stripped to external PDB), for MS Windows

$ file Chapter_1-linux
Chapter_1-linux: ELF 64-bit LSB executable, x86-64, version 1
  (SYSV), statically linked, Go BuildID=bSETwZgNDR5vlulRHnzw/
  KNpENRt9Hipd8Nu7HGDg/v38ZPzDs35yMw33hUxoz/Y_cNfU8fID2cCtz36hCq,
  not stripped
```

Comparing Go with Other Languages

When learning a new programming language, it's always helpful to try to compare it with another language that you may already be familiar with. Doing so allows you to try to map your current knowledge to the new language that you're trying to learn.

In this section, I compare Go with two of the most commonly used programming languages used in the industry, Java and Python. Occasionally, I compare Go with C, because Go is syntactically similar to C. Go is often touted as the language with the speed of C and the productivity of Python.

REMEMBER

If you aren't familiar with any of the languages listed in this section, don't worry! In this book, I cover all the features mentioned here.

Syntax

In terms of syntax, Go is closer to C and Java, which use curly braces to enclose blocks of code. This syntax differs from Python, which uses indentation as a form of denoting blocks of code.

Like Python, Go functions are first-class citizens, whereas in Java everything revolves around classes, and you need a class just to enclose a function.

Unlike Python and Java, Go doesn't have OOP support, and it doesn't support inheritance. But it does have interfaces and structs that work just like classes.

Go is statically typed, like Java. It differs from Python, which is dynamically typed.

Compilation

Whereas Python and Java are compiled to byte code, which is then interpreted and run on a virtual machine, Go compiles directly to machine code, which makes Go take the lead in terms of performance.

Like Python and Java, Go is also garbage collected. In programming, garbage collection (GC) is a form of automatic memory management. The garbage collector tries to reclaim memory occupied by objects that are no longer in use by the program.

Python is extremely memory intensive. Java is not much better because everything is heap allocated. But Go affords more control of memory usage.

Concurrency

Go has parallelism and concurrency built in, which means writing multi-threaded applications is very easy. Both Java and Python support concurrency through threading, but they aren't as efficient as Go. In fact, concurrency is one of Go's main selling points.

Library support

All three languages have huge library support — both standard and third-party libraries. A language's survival depends in large part on its support for third-party libraries. That's why Python has been so hot for the past couple of years — its support for third-party libraries for doing data analytics brings machine learning and deep learning readily to the masses. Although Go doesn't have the scale of third-party library support that Python does because it's young, the number of libraries for Go is growing.