

## 1

### Get Started

7

Meet the Go Language	8
Install the Go Tools	10
Create the Go Workspace	12
Write a Go Program	14
Run a Go Program	16
Format and Comment Code	17
Explore the VS Code Editor	18
Summary	20

## 2

### Store Values

21

Create Program Variables	22
Display Variable Values	24
Convert Data Types	26
Fix Constant Values	28
Point to Stored Values	30
Summary	32

## 3

### Perform Operations

33

Do Arithmetic	34
Assign Values	36
Make Comparisons	38
Assess Logic	40
Juggle Bits	42
Understand Precedence	44
Summary	46

## 4

### Control Flow

47

Test a Condition	48
Switch Cases	50
Loop Several Times	52
Loop While True	54
Break Out of Loops	56
Go to Labels	58
Summary	60

# 5

## Produce Functions

61

Create a Basic Function	62
Add Parameters	64
Pass References	66
Return Values	68
Call Recursively	70
Enclose Anonymously	72
Pass Functions	74
Handle Errors	76
Separate Files	78
Summary	80

# 6

## Build Structures

81

Group Data	82
Attach Methods	84
Embed Structs	86
Encapsulate Features	88
Compose Elements	90
Satisfy Interfaces	92
Embed Interfaces	94
Summary	96

# 7

## Create Arrays

97

Create a Basic Array	98
Loop Through Elements	100
Slice Arrays	102
Make Slices	104
Slices Versus Arrays	106
Map Keys and Values	108
Summary	110

# 8

## Harness Time

111

Get Dates	112
Get Times	114
Format Date and Time	116
Set Date and Time	118
Recognize Zones	120
Delay Time	122
Summary	124

## 9

**Manage Data****125**

Unite Strings	126
Split Strings	128
Find Characters	130
Convert Strings	132
Calculate Areas	134
Evaluate Numbers	136
Round Decimals	138
Generate Randoms	140
Summary	142

## 10

**Handle Input****143**

Get User Input	144
Buffer Input	146
Command Flags	148
Read Files	150
Write Files	152
Use Temporary Files	154
Summary	156

## 11

**Employ Concurrency****157**

Create Goroutines	158
Keep Waiting	160
Make Channels	162
Buffer Channels	164
Select Channels	166
Synchronize Goroutines	168
Use Worker Pools	170
Summary	172

## 12

**Request Responses****173**

Listen for Requests	174
Handle a Request	176
Add Files to Serve	178
Deliver a Static Page	180
Log Received Data	181
Deliver a Dynamic Response	184
Summary	186

# How to Use This Book

The examples in this book demonstrate features of the Go programming language (“golang”), and the screenshots illustrate the actual results produced by the listed code examples. Certain colorization conventions are used to clarify the code listed in the steps...

Program code is colored black but keywords and built-in functions of the Go language are colored blue, literal text and numeric values are red, and code comments are green, like this:

```
package main
import "fmt"
main() {

    // My First Go Program.
    fmt.Println( "Hello World!" )
}
```

During setup of Go, you will select a “GOPATH” location on your computer in which to create programs. Each program will be created within a uniquely named folder in a GOPATH sub-directory named “src”. To identify the source code for the example programs described in the steps, an icon and file path appears in the margin alongside the steps:



src/hello/main.go

## Grab the Source Code

For convenience, the source code files from all examples featured in this book are available in a single ZIP archive. You can obtain this archive by following these easy steps:

- 1 Browse to [www.ineasysteps.com](http://www.ineasysteps.com) then navigate to [Free Resources](#) and choose the [Downloads](#) section
- 2 Next, find [GO Programming in easy steps](#) in the list, then click on the hyperlink entitled [All Code Examples](#) to download the ZIP archive file
- 3 Now, extract the archive contents to the GOPATH/**src** sub-directory on your computer

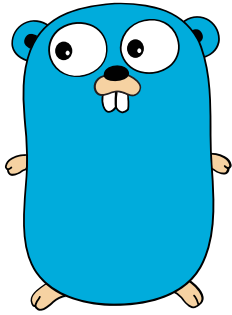
If you don't achieve the result illustrated in any example, simply compare your code to that in the original example files you have downloaded to discover where you went wrong.

# 1

# Get Started

*Welcome to the exciting world of programming with Go. This chapter introduces the language and creates a workspace where you can run your first Go program.*

- 8** Meet the Go Language
- 10** Install the Go Tools
- 12** Create the Go Workspace
- 14** Write a Go Program
- 16** Run a Go Program
- 17** Format and Comment Code
- 18** Explore the VS Code Editor
- 20** Summary

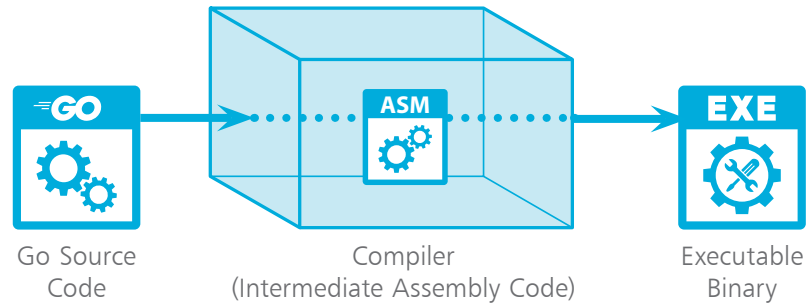


The Go gopher – the iconic mascot of the Go programming language.

# Meet the Go Language

Go is a free open-source programming language created at Google by Robert Griesemer, Rob Pike, and Ken Thompson – best known for development of the Unix operating system. Google released version 1.0 of the Go language (“golang”) in March 2012, since when it has gained widespread popularity.

Go programs are written in plain text, then compiled into machine code by the Go compiler to produce an executable binary version.



The aims of the Go programming language are to be expressive, fast, efficient, reliable, and simple to write. Some programming languages, such as C or C++, are fast and reliable but not simple. Conversely, other programming languages such as Java or Python are simple to write but not so efficient.

Go is similar to the C programming language in many ways, and is sometimes referred to as a “C-like language” or “C for the 21st century”. But Go is much more than that, as it adopts good ideas from many other programming languages, yet avoids features that lead to complexity or unreliability.

Perhaps most importantly, Go introduces the ability to take advantage of multi-core CPU processing for concurrency using “goroutines” and “channels”. This provides the potential for the computer to deal with several things at the same time.

Although the Go language does not have the class structures found in Object Oriented Programming (OOP) languages, such as C++ or Java, its features do provide some degree of encapsulation, inheritance, and polymorphism – the three cornerstones of OOP.



You can discover many more programming language books in the **In Easy Steps** series – including C, C++, C#, Java, and Python. Visit [www.ineasysteps.com](http://www.ineasysteps.com) to find out more.

...cont'd

With so many programming languages to choose from, you may be wondering why you should choose to learn Go programming – so here are some of the advantages that Go offers:

### Simple Syntax

The Go language is concise, like Python. It's as simple to write as Python but is more efficient, like C++. This enables you to write code that is easy to read and maintain.

### Compiled Language

The Go source code is compiled to binary machine code that can be read directly by the computer, instead of being interpreted every time a program runs. This enables the Go programs you write to run faster than programs written for interpreted languages, like Python or PHP.

### The Go Compiler

The Go compiler is fast and provides additional benefits, such as code optimization and error checking – it can detect unused variables in your code, missing imports that your code requires, and mistyped or invalid code. The Go compiler can also generate executable binaries for other operating systems. This enables you to compile your source code to run on multiple machines.

### Concurrency

The Go language provides inherent support for concurrency with goroutines and channels. This enables you to write multi-threaded programs that could perform multiple tasks at the same time.

### Garbage Collection

Automatic memory management is a key feature of the Go language. Its garbage collector runs concurrently with the program. This enables you to write program code without any concern for memory leakage.

### Static Typing

Go is a statically typed language in which variables are explicitly declared to be of a particular fixed type. This enables errors to be caught early in the development process.

You may well recognize other advantages as you gain experience with the Go language, but right now it's time to get started...



# Install the Go Tools

To get started with the Go programming language, you must first install the Go tools on your PC. These allow you to build, run, and test programs written in the Go language. The Go tools are supplied together with lots of standard packages of useful trusted code that you can import into your own programs. The Go language installers are available for Windows, macOS and Linux.

- 1 Open a web browser and visit <https://golang.org> then download the appropriate installer for your system



- 2 When the download has completed, run the installer to launch the “Go Programming Language Setup Wizard”



The Go installer for Windows should automatically add Go to your system path to make the Go tools available at a command prompt.



- 3 Click **Next** to continue, then accept the license terms
- 4 Accept the suggested **Destination Folder** (at **C:\Go** on Windows), then click **Install** to complete the installation



...cont'd

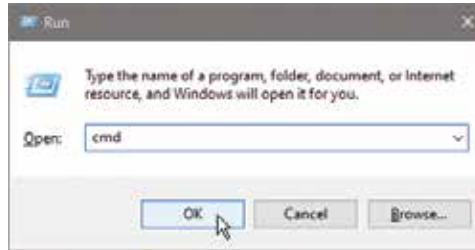
5

To test that the installation was successful, first open a Terminal window – on a Windows system press

**WinKey + R** together, to open a “Run” dialog, then enter `cmd` to open a “Command Prompt” window

6

At the command prompt, type the command `go` then hit **Enter** to see a list of Go tool commands



The “WinKey” is the keyboard key labeled with the Windows logo.



```
cmd Command Prompt
C:\Users\mike>go
Go is a tool for managing Go source code.

Usage:
    go <command> [arguments]

The commands are:
    bug          start a bug report
    build        compile packages and dependencies
    clean        remove object files and cached files
    doc          show documentation for package or symbol
    env          print Go environment information
    fix          update packages to use new APIs
    fmt          gofmt (reformat) package sources
    generate     generate Go files by processing source
    get          add dependencies to current module and install them
    install     compile and install packages and dependencies
    list        list packages or modules
    mod         module maintenance
    run         compile and run Go program
    test        test packages
    tool        run specified go tool
    version     print Go version
    vet         report likely mistakes in packages

Use "go help <command>" for more information about a command.

Additional help topics:
    buildmode   build modes
    c           calling between Go and C
    cache       build and test caching
    environment environment variables
    filetype    file types
    go.mod      the go.mod file
    GOPATH     GOPATH environment variable
    GOPATH-get legacy GOPATH go get
    GOPROXY    module proxy protocol
    importpath import path syntax
    modules     modules, module versions, and more
    module-get  module-aware go get
    module-auth module authentication using go.sum
    module-private module configuration for non-public modules
    packages   package lists and patterns
    testflag   testing flags
    testfunc   testing functions

Use "go help <topic>" for more information about that topic.
```



Although there are quite a few Go tools, you will mostly use only the **run** tool to compile and run your programs.





The `go\src` folder is where you will save the programs you write. The `go\bin` and `go\pkg` will be used later to store executable files and package archives.



The `GO111MODULE` environment variable is set to `ON` by default (this was `OFF` by default in earlier versions of `GO`). The examples in this book cannot be executed as described until this is turned `OFF`. You should enter this command to correct the issue:  
`set GO111MODULE=off`

# Create the Go Workspace

When you install Go, the installer sets a number of Go environment variables. For example, the directory (folder) location of the Go tools is stored in a `GOROOT` environment variable – by default, at `C:\Go` on a Windows PC, and at `/usr/local/go` on Linux and macOS.

The installer also sets a `GOPATH` environment variable for the location of your workspace. By default, this is a directory named “go” within your home directory. For example, on a Windows PC its path is `C:\Users\user\Name\go`, on Linux systems its path is `/home/user/Name/go`, and on macOS it’s at `/Users/user/Name/go` – but the installer doesn’t actually create any directories.

To create the workspace, you can simply add a directory named “go” in your home directory. You must then add sub-directories named “bin”, “pkg” and “src” within the workspace directory – all your Go programs can then be created inside the “src” directory.

It’s useful to have a shortcut on your desktop that will open a command-line in this sub-directory to easily run your programs.

- 1 On a Windows PC, open a Command Prompt window, as described on page 11, or open a Terminal window
- 2 Enter the command `go env GOPATH` to see the current expected workspace location

```

C:\WINDOWS\system32\cmd.exe
C:\Users\mike_>go env GOPATH
C:\Users\mike_\go
  
```

- 3 Next, issue a `mkdir` command to create the “go” workspace directory at the location specified by the `GOPATH` environment variable

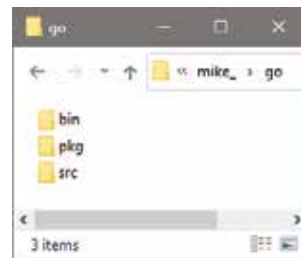
```

C:\WINDOWS\system32\cmd.exe
C:\Users\mike_>mkdir go
C:\Users\mike_>
  
```

...cont'd

- 4 Now, issue further **mkdir** commands to create the “bin”, “pkg” and “src” sub-directories in the workspace directory

```
C:\WINDOWS\system32\cmd.exe
C:\Users\mike_>mkdir go\bin
C:\Users\mike_>mkdir go\pkg
C:\Users\mike_>mkdir go\src
C:\Users\mike_>_
```



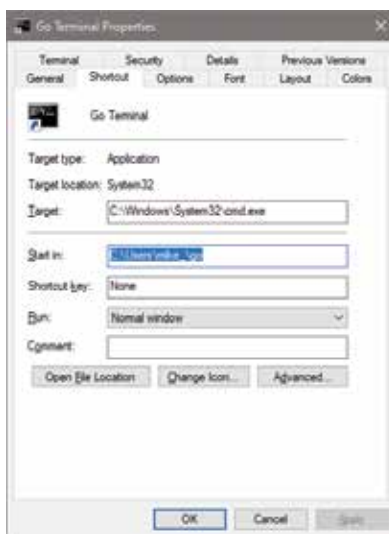
- 5 Right-click on a Windows desktop and select **New, Shortcut** to open a “Create Shortcut” dialog

- 6 Enter **cmd** as the location and click **Next**, then enter **Go Terminal** as the name and click **Finish** to create a shortcut icon on your desktop



The **mkdir** command name is simply short for “make directory”.

- 7 Right-click on the shortcut icon and select the **Properties** item, to open its “Properties” dialog



- 8 Choose the **Shortcut** tab, then enter the location of your workspace directory

- 9 Click **Apply**, **OK** to close the dialog, then double-click the shortcut icon to open a command-line in your “src” folder

```
C:\Users\mike_\go\src>_
```



Go programs can be written in a plain text editor, such as Windows’ Notepad app. It’s useful to have a desktop shortcut that opens a text editor in your “src” directory folder. Repeat the steps on this page, but in Step 7 enter **notepad** as the location and **Go Editor** as the name, to create another useful desktop shortcut.



src/hello/main.go

# Write a Go Program

All Go programs start as plain text files that are later compiled into actual executable programs. This means that Go programs can be written in any plain text editor, such as the Windows' Notepad app or the Nano app on Linux.

Follow these steps to create a simple Go program that will output the traditional first program greeting:

- 1 Create a sub-directory named “hello” in your “src” folder

```
C:\Users\mike_\go\src>mkdir hello
C:\Users\mike_\go\src>
```

- 2 Open a plain text editor, like Notepad, and type this code exactly as it is listed to begin a program  
`package main`

- 3 Two lines below, insert this code exactly as it is listed  
`import "fmt"`

- 4 Two further lines below, precisely add this code  
`func main() {`  
 `fmt.Println( "Hello World!" )`  
`}`

- 5 Save the file in the “hello” folder, and name it `main.go` – the complete program should now look exactly like this:

```
main.go - Notepad
File Edit Format View Help
package main
import "fmt"
func main() {
    fmt.Println("Hello world!")
}
```



Capital P and lowercase L in `fmt.Println`.



The arrangement of files within folders is important in Go. Each main file, and any related files, must be placed in a uniquely named folder to create a package – so here the package is named “hello”.

...cont'd

The separate parts of the program code on the opposite page can be examined individually to understand each part more clearly:

## The Package Declaration

### package main

The package type is declared following the **package** keyword. All Go program code is contained in packages. You may declare your own type for a package that will be a shared library, but you must declare the package “main” if you want the code to be compiled into an executable program.

## The Import Declaration

### import “fmt”

The keyword **import** is used to import one, or more, packages into this package to make their features available to this program. The package “fmt” comes from the Go standard library that is included in your Go installation. It provides the **fmt.Println( )** function that is used to output text in this program. Note that the package name must be enclosed in double quote characters. When importing multiple packages, the list of package names must be enclosed within parentheses and each name must appear on its own line, like this:

```
import (  
    “fmt”  
    “strings”  
)
```

## The Function Declaration

```
func main( ) {  
  
    fmt.Println( “Hello World!” )  
  
}
```

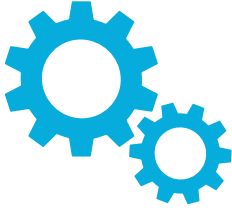
The function name follows the **func** keyword. It must be followed by parentheses and an **{** opening curly bracket on the same line. The function body contains statements that are the actual instructions to perform program tasks. The function body must end with a closing **}** curly bracket. Function names must be unique, but each Go program must have a function named “main” as this is the starting point of all Go programs.



The package name and package type are two separate items – here the package is named **hello**, but the type is **main**.



You can find the Go standard library packages in the “src” directory of your **GOROOT** directory location – for example, at **C:\Go\src**. Additionally, you can learn about each package from the official documentation at <https://golang.org/pkg> – an invaluable resource.



# Run a Go Program

Go program code is compiled and executed using the Go tools. During program development, an error-free Go program can be compiled and run by the **go run** tool. This is useful, but if you want to create an executable binary file version that can be executed repeatedly and distributed, you can use the **go build** tool.

- 1 Open a Command Prompt or Terminal window in your Go “src” folder
- 2 Enter the command **go run hello** to run the program written on page 14 – it should output the greeting

```

C:\Users\mike_\go\src>go run hello
Hello World!
C:\Users\mike_\go\src>
  
```

- 3 Next, enter the command **go build hello** to make an executable binary version of the program on page 14 – it should add an executable file in your “src” folder
- 4 Now, twice enter the command **hello** on Windows, or **./hello** on Linux or macOS – it should output the traditional greeting two times

```

C:\Users\mike_\go\src>go build hello
C:\Users\mike_\go\src>hello
Hello World!
C:\Users\mike_\go\src>hello
Hello World!
C:\Users\mike_\go\src>
  
```



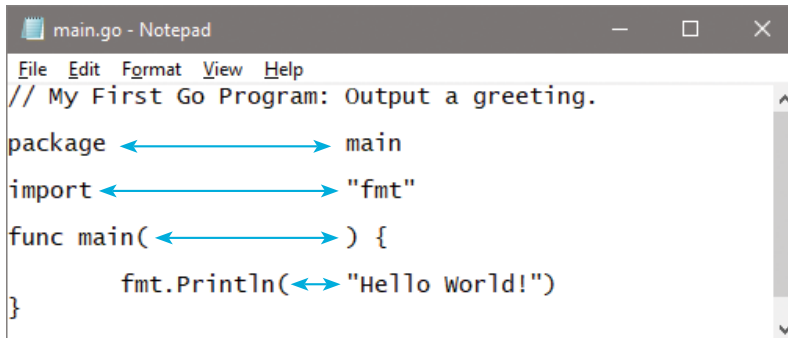
On Windows systems the binary file is given the file extension **.exe**, so the file built here is **hello.exe**.

# Format and Comment Code

It is good practice to always ensure your Go source code is properly formatted by the **go fmt** tool before running a program.

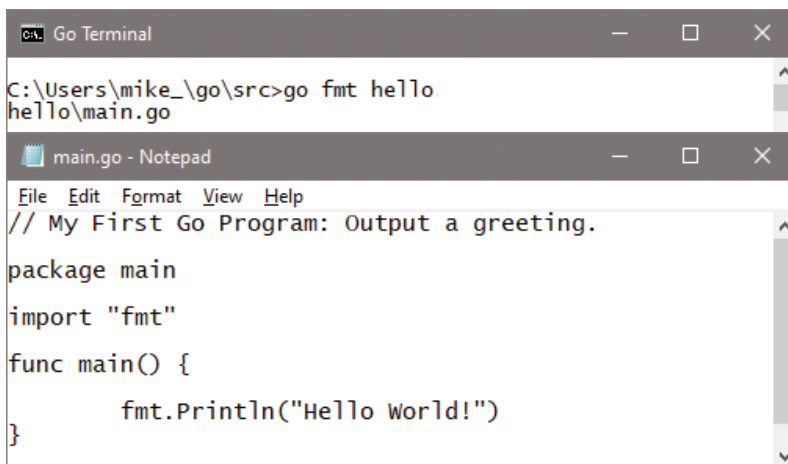
It is also good practice to include explanatory comments in your source code so it can be more easily understood by others, and by yourself when revisiting the code later. Single-line comments can be added after a `//` character sequence – everything on the line after `//` is ignored by the compiler. Multi-line comments can be added between `/*` and `*/` character sequences – this is useful to “comment-out” blocks of code, to hide it from the compiler during development.

- 1 Edit the code in **hello.go** on page 14 to add a comment, and unnecessary tab spacing (bad formatting)



```
main.go - Notepad
File Edit Format View Help
// My First Go Program: Output a greeting.
package      ← →      main
import      ← →      "fmt"
func main(   ← →      ) {
    fmt.Println(↔↔ "Hello world!")
}
```

- 2 Now, at a prompt, enter the command **go fmt hello** then reopen the source code file to see correct formatting



```
Go Terminal
C:\Users\mike_\go\src>go fmt hello
hello\main.go

main.go - Notepad
File Edit Format View Help
// My First Go Program: Output a greeting.
package main
import "fmt"
func main() {
    fmt.Println("Hello world!")
}
```

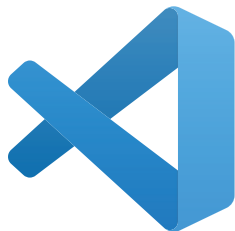


Comments should not be “nested”, one inside another.



The **go fmt** tool provides some simple error checking. For example, omit a final “ double quote and it will provide the line number and position where the error occurs and provide the nature of the error as “string literal not terminated”.



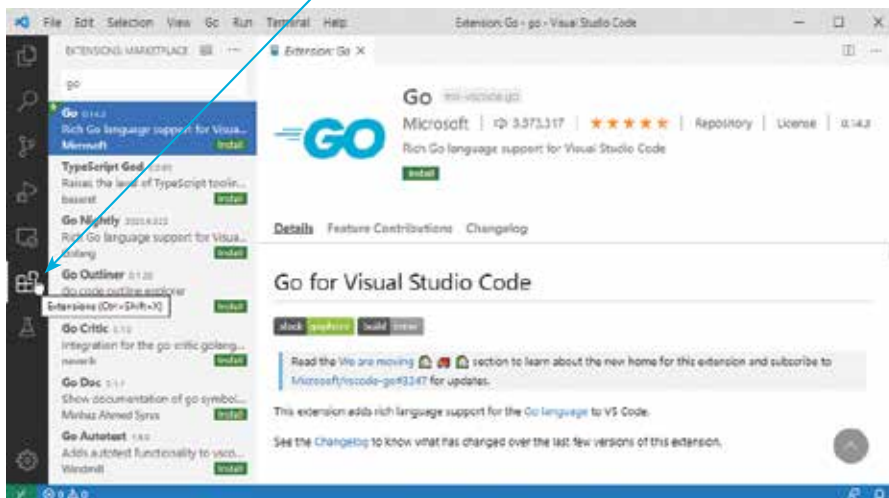


# Explore the VS Code Editor

Although you can create Go programs just fine with a command prompt and plain text editor, many developers prefer to use a specialized code editor app. These offer useful additional features such as syntax highlighting, auto-indentation, bracket-matching and code debugging. Perhaps the best free code editor is the Visual Studio Code (“VS Code”) app from Microsoft. It’s available for download at <https://code.visualstudio.com/download> in versions for Windows, Linux and macOS.

When you download and install VS Code, you have only got the code editor framework. VS Code can actually support many different programming and scripting languages, so you have to install an appropriate extension for the one you want to code with.

- 1 Download and install VS Code
- 2 To open the “Extensions Marketplace” sidebar, first click **View, Appearance, Show Activity Bar**, then click the **Extensions** button on the Activity Bar

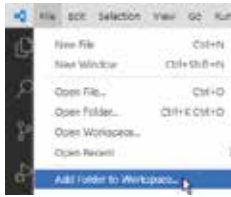


- 3 Next, type “go” into the search box to find the VS Code extension for the Go programming language
- 4 Click **Install** to add the Go language support to VS Code – if asked to add further Go support, select **Install All**



...cont'd

- 5 Click **File, Add Folder to Workspace** and add your “go” workspace folder containing the “src” folder with your source code



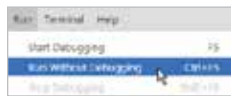
- 6 Click the **Explorer** button on the Activity Bar, then select a file to open in the main VS Code editor window – for example, click the **main.go** file in the “hello” folder



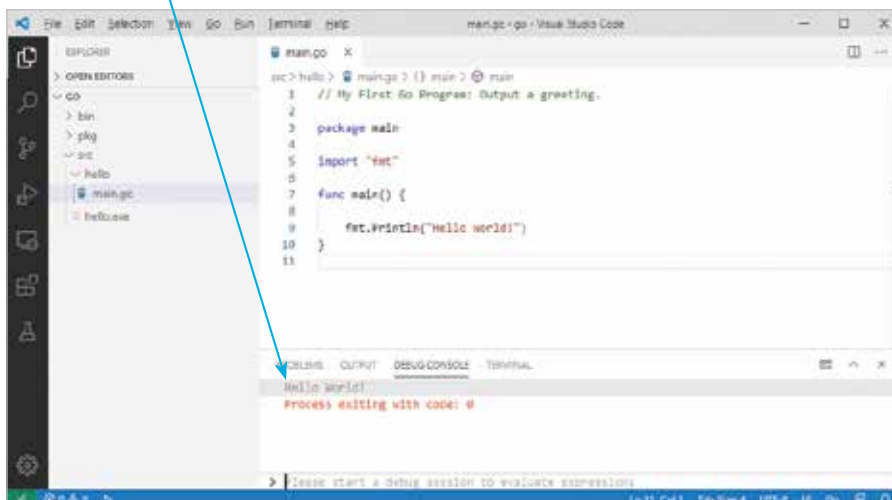
- 7 Click **View, Debug Console** to open an output panel – typically, this will appear below the code editor window, but can be moved



- 8 Click **Run, Run Without Debugging** to see the output appear in the Debug Console



The VS Code app is highly customizable – you change the layout of its components to your liking and choose from several light or dark color themes. The screenshots shown here depict the default Light theme, and the syntax highlighting shown here in the code editor window is the same as that used throughout this book.



# Summary

- Go programs are plain text files that get compiled into machine code to produce an executable binary version.
- The Go tools allow you to build, run, and test programs written in the Go programming language.
- The Go installation includes standard packages of useful, trusted code that you can import into programs.
- The command-line **go** command displays a list of the Go tool commands.
- The **GOROOT** environment variable stores the location on your system where the Go installation is located.
- The **GOPATH** environment variable stores the location on your system where your Go workspace is located.
- The Go workspace is a directory name “go”, containing sub-directories named “src”, “bin”, and “pkg”.
- The name of a Go package is that of the directory containing the Go source code files.
- A Go **package** declaration specifies its type, and must be specified as **package main** to be compiled into an executable.
- An **import** declaration quotes the name of any packages whose features are to be made available to the program.
- A function declaration contains the **func** keyword followed by the function name, ( ) parentheses, and { } curly brackets.
- All Go programs must have a function named “main” as the entry point into the program.
- The **go run** command compiles and runs a program.
- The **go build** command compiles a program and produces an executable binary file that can be distributed.
- The **go fmt** command should be used to correctly format the source code of a Go package.
- VS Code offers features such as syntax highlighting, auto-indentation, bracket-matching, and code debugging.