FIRST EDITION – CHAPTER 1 REV 1

Kevin Thomas

Forward

“So many languages, so little time!”. Today we are literally all completely overwhelmed with new tech, new vulnerabilities, new requirements, misinformation, disinformation and it makes it near impossible to try to keep up in the world of Cyber and Development.

Why Go? Why should I begin or continue my learning journey with Go? These are very important questions that should be addressed up front.

In today’s world just about every software product is a massive distributed system. The product is made up of hundreds of individual apps tied to dozens of containers that must interoperate with each other. The challenge here is concurrency.

Go tackles this challenge with goroutines which is nothing more than a lightweight thread managed directly by the Go runtime. Go has growable segmented stacks. Instead of giving each goroutine a fixed amount of stack memory the Go runtime gives goroutines the stack space they need on demand which allows the Developer from having to make decisions up front of allocation. This means it will use more memory ONLY when needed and is lightening fast!

Go has a robust standard library that provides built-in functionality to handle just about everything one needs. It also has built-in network capability unlike any other language out of the box.

Go is FAST! Go is up to 200 times faster than PHP, Python, Java and the like. It is a compiled language and extremely efficient and easy to use.

Go is a statically-typed language which prevents many surprises that you find in languages like Python. In Go when you create a variable its type remains with it throughout it’s lifetime.

Go has a web server built-in to the standard library which is production ready and can handle as many connections as the hardware will handle.

“Yea, yea, yea! I heard all this before. I already know Python, C++ and Java and that is all I will ever need! I can do all of these things in these languages so you can keep your shiny newness to yourself!” Like it or not if you are in the Cyber world, Malware authors ARE and WILL be developing in Go as it is easy to use and extraordinarily powerful and reverse engineering it is a literal nightmare. In the near future I will be writing a, “Hacking Go”, FREE tutorial as well to supplement this book so that everyone has a means to properly understand how to reverse engineer these binaries.

It is WORTH learning Go for all of these reasons if you are in this field! Take 30 minutes a day and take SMALL bites and get it done! Carving out time each day, just a SMALL block, and STICKING TO IT is the best advice someone can give if you want to master this realm and change the world. It’s that simple it just takes dedication and a commitment to SMALL blocks of time so you can go about your day.

This tutorial assumes you have Go installed on either Windows, MAC or Linux or you can use the FREE online Go editor **Replit** https://replit.com which is the easiest option if you are not already set up.

I would however STRONGLY recommend Visual Studio Code with the official Google Go extension as this will be the best experience possible for larger projects.

Table Of Contents

Chapter 1: Basic I/O

We are going to dive right into basic input and output.

By the end of the lesson we will have completed the following.

```
* Written a 0001_hello_world.go app which will output "Hello World!"
to the console.

* Written a 0002_basic_io.go app which will demonstrate the ability for us to obtain
keyboard input and dynamically populate logic in the console based on the user
submission.
```

STEP 1: Open Your Go Editor Or replit.

STEP 2: Create File 0001_hello_world.go

STEP 3: Type Code

```
package main

import "fmt"

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

STEP 4: Run File In Terminal

```
go run 0001_hello_world.go
```

STEP 5: Review Output

```
Hello World!
```

We begin our understanding of Go with the very first line of this code. We see a *package main* which tells the Go compiler that the package should compile as an executable program.

We see that we import *fmt* which *fmt* is the format package which is part of the Go standard library.

We then see our *main* function designated by the *func* keyword followed by *main*. There MUST be ONE and only ONE function in your application called *main* where all execution will build from.

This *fmt* package gives us access to the *Println* function. The *Println* function in Go is a *built-in* function which literally prints strings or words into the console. In order for the *Println* function to be executed we need to add a pair of parenthesis () after the function name.

The words or string that goes between the parenthesis are what is called a *function argument*. In our case, we are going to pass a string surrounded by a set of single or double quotes.

The contents of the *Println* function will print out the words 'Hello World!' to the console. Whatever word or words we put inside the parenthesis will determine what gets printed to the console.

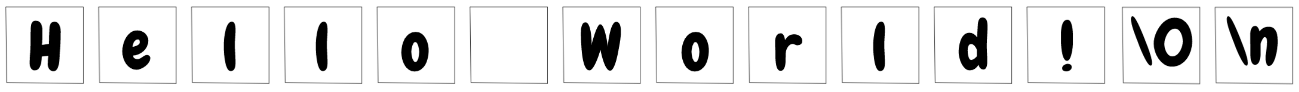
Let's dive into what a string is. A string is a string of characters or letters. Imagine if we had a bunch of little boxes on a table.

So we have our string, *Hello World!* which takes up 12 boxes.

Strangely if we count the boxes we see 14. Let's examine why.

Each box contains a letter or character which we refer to as an element in Go. There is also what we call a null terminating character '\0' and a new line character '\n' that are two additional characters inside the print function. The good news is when the Go team made the *fmt* package as part of their standard library, they built-in what we refer to as *default arguments* inside the *Println* function so you, the developer, does not have to type them every time you want to print something to the console.

Now let's look at the boxes with all of the letters, spaces, null terminating character and new line character.



That is exactly what is going on inside your computer's memory under the hood.

One of the most POWERFUL aspects of Go is that it will NOT let you compile if you have unused non-global or non-package-level variables as this helps out tremendously when projects scale. This combined with Go’s statically-typed functionality helps significantly with keeping code safer and much easier to maintain and debug!

Now that you have a handle on all of the steps to create and save your code we will an additional example as well to help solidify these concepts.

Let's clear out our code and rename our new file to **0002_basic_io.go** and type the following code.

```
package main

import "fmt"

func main() {
	// We introduce the concept of a variable to which
	// we reserve a little box in our computer's memory
	// to hold the string which we are going to type
	// when prompted to provide our favorite food and
	// favorite drink
	fmt.Printf("What is your favorite food? ")

	// Create a string var for favorite food
	var favoriteFood string

	// Take input from the user
	fmt.Scanln(&favoriteFood)

	// Ask the user for their favorite drink
	fmt.Printf("What is your favorite drink? ")

	// Create a string var for favorite food
	var favoriteDrink string

	// Take input from the user
	fmt.Scanln(&favoriteDrink)

	// Here we  provide a response back based to the
	// console based on our two variables which we
	// inputted above
	fmt.Printf("I love %s and %s as well!", favoriteFood, favoriteDrink)
}
```

I want to introduce the concept of adding comments. We see a *//* and then everything after the *//* on a line is what we call a *comment*. These are helpful to remind us what is going on in our code.

When we start out we can use as many comments as we want. As we get more comfortable we will tend to use fewer comments as we will get a better handle of what is going on by looking at the Go code.

A variable holds a value in those little boxes like we saw earlier and we can use this to store any information we want during our app's run. The difference here is that variables can change during our app and not stay constant.

In Go we want to pay close attention to how we name variables. We use Pascal Case when we create variables that are local or private to the package it lives in. Therefore *privateVariable* would be the proper way to handle this. This is similar to C++’s private class mechanism.

When we want to create a public or global variable that will be available to all packages within the application we use Camel Case. Therefore *publicVariable* would be the proper way to handle this. This is similar to C++’s public class mechanism.

Go is not an object-oriented language but still has many of the advantages of OOP as we will see throughout this tutorial. OOP is a great user-level abstraction however the CPU ultimately is procedural and does not see classes in it’s machine code. This is another reason why Go is so efficient and fast as it is closer to the hardware when compiling.

We are also introducing the concept of basic input in Go which we refer to as *Scanln*. This is a built-in function like the *Println* function that allows us to capture whatever we type and then stored into a variable.

We also use the *Printf* function which handles string formatting such that we can get input without a new line and display values using format specifiers like *%s*.

We see *fmt.Printf("What is your favorite food? ")* and all this does is display the words, What is your favorite food? and then allow us to type a string response and then it will be stored in *favoriteFood*. For example if we typed *pizza* then the string *pizza* would be stored in the *favoriteFood* variable as we create it by typing *var favoriteFood string* which will be of type string.

We repeat the process for *favoriteDrink* in the exact same way.

Finally we use the *printf* function again and we use what we refer to as a *format method*. Notice we see *%s* and *%s* which are placeholders for our variables so what will happen is that if we used the word *pizza* for *favoriteFood* it would replace the *%s* with *pizza* and if we used *Pepsi* for *favoriteDrink* the second *%s* would be replaced with *Pepsi*.

Run your program in the console.

```
What is your favorite food? pizza
What is your favorite drink? Pepsi
I love pizza and Pepsi as well!
```

It is now time for our first project!

Project 1 - Create a Candy Name Generator app - You are hired as a contract Go Software Developer to help Mr. Willy Wonka rattle off whatever candy title he comes up with in addition to a flavor of that candy. When the program is complete, Mr. Willy Wonka will be able to type into the console a candy title he dreams up in addition to the flavor of that candy. For example, Scrumpdiddlyumptious Strawberry.

STEP 1: Prepare Our Coding Environment

Let's clear out our code and rename our new file to **p_0001_candy_name_generator.go** and follow all of the steps we learned so far.

Give it your best shot and really spend some time on this so these concepts become stronger with you which will help you become a better Go Developer in the future.

The real learning takes place here in the projects. This is where you can look back at what you learned and try to build something brand-new all on your own.

This is the hardest and more rewarding part of programming so do not be intimidated and give it your best!

If you have spent a few hours and are stuck you can find the solution here to help you review a solution. Look for the **Part_1_Basic_IO** folder and click on **p_0001_candy_name_generator.go** in GitHub.

https://github.com/mytechnotalent/Fundamental-Go

I really admire you as you have stuck it out and made it to the end of your first step in the Go Journey! Great job!

In our next lesson we will learn about Go data types and numbers!