Golang Basics

Introduction to GO

GO was developed by Google in 2007 and was later Open sourced in 2009. To begin with, while using any new programming languages we have two questions 1. Why do we need a new programming language and 2. How it is better/different from others? The answer is quite simple with the changes in infrastructure from physical servers with single core processes to Cloud servers, multicore processors and Big networked computation cluster the infrastructure evolved to be more scalable and distributed, dynamic and has more capacity. However, older languages were not built for this purpose and/or do not use the current infrastructure efficiently which can make applications faster and user-friendly. As an example, we can take the commonly used WhatsApp, we can upload a status in background while consequently chatting with someone or sharing a media on a group as well, this is what the modern infrastructure is capable of that most programming languages that already exist fail to utilize, this is called multi-threading when one thread does a work another thread can work simultaneously to achieve a different task.

Multi-threading comes with its own challenges, the most popular one occurs when multiple users try to accomplish different tasks on a specific resource. As an example, let us consider two different people trying to book tickets online for the same seat, both can't get access to the same seat as only one person can occupy it, this must work in a way that no more than one person can book the same seat. This is called as **Concurrency,** which is defined as dealing multiple things at once, the developers must write the code in such a way that it prevents any conflicts between tasks when multiple tasks are made to run in parallel and in updating the same data.

The purpose of GO is to run on multiple cores and has built-in concurrency, Concurrency in GO is cheap and easy to achieve using GO routines than using other programing languages.

GO Use Case

1. For writing performant applications.
2. Can run on scaled, distributed infrastructure utilizing the resources efficiently.

GO Characteristics

- Simple and Readable syntax of a dynamically typed language like Python.
- Efficiency and Safety of a lower-level, statically typed language like C++.
- Used for server-side or Backend side applications (E.g.: Microservices, Web Apps, etc..).
- **Simple Syntax** : Easy to learn, read and write.
- **Fast**: Faster build time, startup and runtime.
- **Resource Efficient**: Requires very few resources.
- **Compiled Language**: Can compile into simple binary code faster than interpreted languages like Python. Binary file generated can be used across different OS.

GO Setup

Download the latest stable version of GO from Go Installation selecting version based on the local configurations and OS type, open the downloaded file and follow the install instructions, to use code we will need an IDE which will be VS Code . Post installation, open the GO installed folder tavigate to src folder (Directory Path: C:\Users\Excaliber\go\src) copy the directory path and add it to the path variables in the Environment VariablesSetup. Do the same step but this time navigate to the bin folder and repeat the same to set the path variable. For better and easy coding we may use some VS Code extensions, the extensions I make use of for GO are 1. GO Code from Google 2. Auto Import by ElecTreeFyre.

To get started there is one last step: Create a folder with a preferred name and then create file in it using VS code and name it "main.go". Creating this file will produce some pop-ups for additional installation download all and wait for it to complete the installation.

Our First Program – Hello World

In the file created, type the code given below

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

The above code prints a simple "Hello World" message to the console. This is the basic structure for a Go program, first we declare the package followed by the package name(can be same as file name), next we import fmt which is a format package used to format output and so on, then we declare a main function to print the message using print function.

Before we run the code, we need to initialize the folder in which we have our code to be executed. Open terminal in VS Code and change the working directory to the folder we have our main.go in and execute the command "**`go mod init`**" this will create a new file called go.mod file in our folder which will contain the name of the folder and the GO version.

Note: A GO application can have only one main function, because there can be only one entry point per application.

Variables and Constants

Variables are used to store values, say like a box that holds a value within it. We can use variables to declare a value and later these variables can be used throughout the program/application.

Syntax: There are two ways of declaring a variable in GO.

Direct Declaration: `var <var_name> = <value>`
Indirect Declaration: `var <var_name> data_type`
In Indirect declaration we can assign value later.

Constants are variables whose values are set permanently and cannot be changed throughout the program. A constant is declared permanent whereas a variable can be changed.

Syntax: Constants have only one method of declaration in GO

```
const <var_name> = value
```

Print Statements

There are various ways we can output a data into the console using the print statement provided by the **fmt** package in GO. Some of the common ones are:

- **Println –** Used to print the given data and create a new line after printing the statement.
- **Printf –** Used to print data in a certain GO formatted manner using certain place holders.
- **Sprintf –** formats according to a format specifier and returns the resulting string.

Datatypes in GO

Datatypes define the type of value that is being used or assigned to a variable, like other languages GO supports most of the common datatypes. Commonbasic datatypes in GO are:

Integer	**Float**	**Boolean**	**String**
int32,int64	float32	bool	str
uint32, uint64	float64		

User Inputs

Inputs to a variable from an user can be caught from the console using **Scan** function available.

Syntax: `fmt.Scan(&<var_name>)`

Where variable name is the variable, the scanned data will be assigned to. We declare the variable name as a pointer that states the address of the variable in memory.

Pointers

A pointer is a variable that points to the memory address of the given variable that references the actual value. Pointers make it lighter to transport a variable, instead of bringing the entire variable a pointer can be used as a reference to the variable that can be used to perform a particular operation.

A pointer can be declared using **&**in front of the variable name to declare a pointer to that variable, a pointer to a variable can be decoded to get the value by using ***** in front of the pointer name to retrieve the data/value available in that memory address.

Arrays and Slices

When we have many data to be stored it is impossible to keep assigning new variables to each value when a new value is added. To overcome this issue there are data structures that can be used to store large amounts of data by just declaring once and we can keep adding values. In simple words, Data Structures are used to store a collection of elements in a single variable. The two most used data structures in GO to achieve it are Arrays and Slices.

Array is a collection of data of similar datatypes, that is an array can have a list of values that are either integer or a string and so on, but not of different datatypes. Arrays are immutable, that is an array has a fixed size that cannot be changed over time. Let's say we have declared an array of 10 elements then we try to enter 11 elements (1 more than the array was intended for), while assigning the 11th element the array doesn't any more insertion as the size is bound to be 10. Arrays have an index for each element, that helps to identify each element in an array, the indexing in Arrays start with 0 and ends in n-1 where n is the size of the array declared.

Syntax:
```
var <var_name> [array_size] data_type {}
var_name[index] = <value>
```

Slices are used when we are not sure about the size of the list we need, since slices are mutable they can grow and shrink with regard to the data stored within itself.

Syntax:
```
var <var_name> [] data_type
var_name = append (var_name, value)
```

Loops

Looping is a method used to execute a block of code multiple times. Looping methods in GO are simplified and there is only one method that is for loop. The for loop itself can be used to achieve various looping operations.

Syntax:
```
for {}
for <element>, <var_name>:=  range <list_name>{}
```

The for loop can be used with three parameters as we usually make use of them or in Go we can use them in different ways like for loop with no parameters(1-syntax) or like a foreach loop as in 2-syntax.

Conditional Statements

Conditional statements help us to check for a specific condition in our code if the condition is true a block of corresponding code is executed. Conditional Statement in GO is if-else, this statement can also be altered as per the requirements using relational/comparison operators.

Syntax:
```
if condition {
} else {
}
```

The if can be used alone without else if we don't want to execute anything if there is no need to execute while the if condition is not satisfied.

Switch Statement

Switch statements are used to check for multiple conditions based on a single input. Switch statements have a default condition at the end which is executed when no other condition in the switch statement is true.

Syntax:
```
switch var {
case "cond1":
      //code
      break
case "cond2":
      //code
      break
case "cond3":
      //code
      break
default:
      //code
}
```

Functions

Functions help encapsulate code within their own container which logically work together. In other words, we pack a block of code into a function, and it is given a name called function_name which is based on wat it does.

A function can be called by its name depending upon the number of times we want to execute the function. Every program has at least one function that is the **main()** function. Functions in GO can be declared in the following ways: 1. Function with no parameters, 2. Function with parameters (input only, input and output)

Syntax:
```
func func_name () {
      //code
}

func func_name (param1 param1_type) {
      //code
}

func func_name (param1 param1_type) output_type {
      //code
}
```

Packages

A package is a collection of GO files that work together to achieve a specific task. The use of packages help reduces the code size bound within a single file. Globally declared variables can be used between the files within the same package.

A function is accessible by other files within the same package. In GO, packages can share functions and data, to share a function we need the function name to start with capital letter, same applies for variables for a variable to accessible between packages the name of the variable has to be capitalized.

Maps

A map is a set of key-value pairs, it allows storing multiple key-value pairs. Maps unique keys to values. The values from a map can be retrieved using their respective key.

Syntax:

```
var var_name = make(map[key_type]value_type)
map_name[“key1_name”] = value1
map_name[“key2_name”] = value2
map_name[“key3_name”] = value3
```

Creating a map to carry a list of maps can be useful to store many related data under one roof, say for example user info for an online game, user data has details like name, age, device type and so on which are stored in a map but there can be multiple users for a single game so to store every user data uniquely we use a slice of empty maps that contain maps within them.

Syntax:

```
var var_name = make([]map[key_type]value_type, 0)
```

Here the 0 is added to denote the initial size of the slice

All keys and values in a map have same datatype.

Structs

As we saw earlier even though map comes in handy to store multiple data elements to one key, we still cannot be able to have mixed datatypes within our map.

Struct stands for Structure, can hold mixed datatypes. Struct is created as pre-defined helping with suggestions and finding errors prior ahead to compilation.

Syntax:

```
type struct_name struct {
      var_name1  datatype
      var_name2  datatype
      var_name3  datatype
}
```

Concurrency – Go Routines

Usually when an application performs multiple tasks, it does those tasks in a logical order in the form of one after another. But since we want the application to perform faster, we can make multiple tasks perform simultaneously until their resources are not being blocked by another task. That is we can do one task while another task is being processed simultaneously on background. For example, listening to music while browsing. Concurrency is simple that multiple tasks run simultaneously without waiting for another task to be completed.

This is achieved by using Go Routines in Go. Concurrency in GO is simple and cheap. A go routine can be created in many ways one simple way is to add the keyword “go” in front of the function name while calling the function.

Syntax:

go func_name(parameters)

Synchronization is an important part when it comes to Go Routines, while multiple tasks are performed simultaneously, we need to make sure that these tasks do not interfere with each other causing the code to stop. To prevent this scenario, we need to tell the main execution to wait for any Go routines to complete execution before terminating its own execution. This is achieved using a **WaitGroup**, it waits for the GO routines to execute before the execution is terminated.

The WaitGroup has three parts: 1. Adding a routine to WaitGroup each time when a new one is created, 2. Making the main function wait for WaitGroup to be 0 and 3. Intimate WaitGroup each time a routine is completed

Syntax:

```
var wg = sync.WaitGroup{}
wg.Add(number of routines starting)
wg.Wait()
wg.Done()
```

The WaitGroup has three parts: 1. Adding a routine to WaitGroup each time when a new one is created, 2. Making the main function wait for WaitGroup to be 0 and 3. Intimate WaitGroup each time a routine is completed.

String Functions

In Go string have many functions that aren't supported by many languages some of the common ones are listed below the entire list of functions can be found in Go Documentation.

```
strings.Contains(str,value)
```

Checks for a particular value in a given string

```
strings.Fields(str)
```

Divides the given string into words divided by letters and saves them as an array tat can be accessed using index.

```
strconv.FormatUint(uint(var_name), 10)
```

Formats an integer to string

Demo Application – Ticket Booking Application

Criteria:

1. Welcome the guests when application is started.
2. Thank you message when a ticket is booked successfully.
3. Get User Details and number of tickets to be booked.
4. Check tickets availability and update the remaining tickets
5. Keep account of tickets every time it is booked.
6. Validate User Inputs.
7. Process and output FirstName of user with number of tickets registered
8. Generate Tickets booked
9. Email to notify the user about the number of tickets registered.