Introduction to programming in Go

February 11, 2022


ITGilde
“Samen Sterker, Beter, Slimmer en Leuker!”

Contents

1 Introduction to programming in Go

1.1 Go SDK Installation

Information needed for your VM in Azure:

Virtual Machine Name: .....
Virtual Machine IP Address: .....
User Name: .....
User Password: .....
Root Password: .....

Prerequisites

As a normal user, login on to your VM on Azure. The trainer has handed out a student nr to you.

If you are student01 you will use VM st00node01.itgildelab.net.

Log in to you VM with putty, MobaTerm or ssh client; e.g. student01@st01node01.itgildelab.net

Installing Go

You will have access with elevated privileges on the system using sudo. There is no need to use the root account.

```
# Download latest Go 1.17 version

curl -LO https://go.dev/dl/go1.17.6.linux-amd64.tar.gz

# Remove old Go release

sudo rm -rf /usr/local/go

# Unpack Go in /usr/local

sudo tar -C /usr/local -xzf go1.17.6.linux-amd64.tar.gz

# Add Go to user's path:

echo 'export PATH=$PATH:/usr/local/go/bin' > ~/.profile

# Create go local directory tree

mkdir ~/go
mkdir ~/go/src
mkdir ~/go/pkg
mkdir ~/go/bin
```

Log out and login again to have your new profile loaded or source the profile like this:

```
source ~/.profile
```

Check if you have the right version of Go installed

```
go version
```

Create a directory called prj in your homedir and change into it like this

```
mdir ~/prj
cd ~/prj
```

In this directory create a file called `hello.go` with the following contents:

```
package main

import "fmt"

func main() {

        fmt.Println("Hello World")
}
```

Let's initialize the module

```
go mod init helloworld
```

And run the program

```
go run helloworld.go
```

Let's try to build the program

```
go build helloworld.go
```

And run the compiled program

```
./helloworld
```

Now let's alter the program slightly:

```
        package main

import "fmt"
import "rsc.io/quote"

func main() {
        fmt.Println(quote.Hello())
}
```

And try to run it again:

```
go run helloworld.go
```

That didn't work we are missing a dependency it says, let's follow up on the advise:

```
go get rsc.io/quote
```

And try to run it again:

```
go run helloworld.go
```

Much better

Now let's take a look at the contents of `go.mode`

```
cat go.mod
```

What do you see? Try to explain.

More on this later in the course

Using the `go doc` functionality, try to figure out how change the fmt.PrintLn in a call to a log function that prints the same data.

Use `go doc log` to find the apropriate function and have the fmt.Println replaced.

Run the program again and see if the log function is used now

2 Go primitive types

2.1 Strings, integers and floats

Primitive types

In this chapter we are going to practice with Go's primitive types //

Please work from your prj directory. //

The following Go program asks for a name and displays a greeting message. Note that the the 'fmt' package also contains the Scanf function that allows you to read input. As strings are immutable the `name` variable of type string is passed by `reference` to the Scanf function

Save this program in a file called: `input.go`

```
package main

import "fmt"

func main() {

        var name string

        fmt.Print("Enter your first name: ")
        fmt.Scanf("%s", &name)

        fmt.Printf("Good day %s\n", name)
}
```

You can run the program using:

```
go run input.go
```

Test the program by giving it some input

Strings can be concatenated by using the + operator.

Add functionality to your program so that it asks also for your surname and greets you with your fullname (firstname + surname) as a concatenated string.

Run and test your program

Worldclock in Go

The package time contains all kind of date and time functions. You can use it to do date calculations but, let your program sleep, measure time or get the current time. Goal of this excercise is to create a worldclock application that will display the current time in your location as well as the time in a number of other timezones. We will learn how to use arrays, loops, simple error handling as well as some experience with the time package.

The output of your program should look something like this:

```
Europe/Dublin :  2022-02-02 13:23:40.882801541 +0000 GMT
Europe/Amsterdam :  2022-02-02 14:23:40.882801541 +0100 CET
Europe/Helsinki :  2022-02-02 15:23:40.882801541 +0200 EET
America/Denver :  2022-02-02 06:23:40.882801541 -0700 MST
Asia/Kolkata :  2022-02-02 18:53:40.882801541 +0530 IST
Pacific/Auckland :  2022-02-03 02:23:40.882801541 +1300 NZDT
```

In pseudo code:

1. Create an array of strings containing the Locations (e.g. ”Europe/Dublin¨, ”Europe/Amsterdam”)
2. Get the current time using time.Now(). (Use go doc time or https://pkg.go.dev/time to find info over the time package and it’s functions/vars/types)
3. Iterate over de elements in the array of timezone locations
4. Load the location using `time.LoadLocation(name)`
5. Retrieve the time in a timezone/location using: `now.In(loc)`
6. Print location and current time in that location

Extras

- Add error handling when a location does not exist in the Go timezone location database
- Calculate and print the difference in the time between your localzone en the zones in your array/list
- Test if a missing or wrongly named zone is handled correctlt

3

Go conditionals

3.1 Conditionals

Conditionals

In this chapter we are going to practice with Go's conditional statements starting with `if`

Please work from your prj directory.

The following Go program asks for a number fromt he user and prints that number

Save this program in a file called: `input2.go`

```
package main

import "fmt"

func main() {

        var nr int

        fmt.Print("Enter a number: ")

        fmt.Scanf("%d", &nr)

        fmt.Printf("The number you submitted is: %d\n",nr)
}
```

You can run the program using:

```
go run input2.go
```

Test the program by giving it some input

Using if/else statements add functionality to your program to:

- Print even if the nr is even
- Print odd if the nr is odd
- Print negative if the nr is below zero
- Print positive if the nr is positive

Run and test your program

Switch statements

Go's switch statements are an alternative and flexible way to get conditional control flow in your program

Write a program in Go using the `switch` statement that determines if a nr read from the console is:

- a factor of 2
- a factor of 3
- a factor of 5

So when the user inputs 9, the program prints that '9 is a factor of 3'

First solve the problem by having at least one of the properties on the list tested per case.

To augment the program, make the program test on all three cases. E.g. 15 will be reported as a factor of 3 as well as of 5

4

Go loops

4.1 Three partition loop

The 3 partition loop

In this chapter we are going to practice with Go's for statement to implement loops

Please work from your prj directory. //

Create a Go program that given an input of `n` will sum up all the nrs from `0` till `n` (incl). E.g. if n=5 it will produce the output 0+1+2+3+4+5 = 15

Save this program in a file called: `loop-1.go`

Nested loop with breaks

Write a program in Go that contains two nested loops, an outer- and an innerloop. The outerloop should run with an iterator variable `i` from 0 till 10 (incl). The innerloop should run with an iterator variable `j` from 0 till 5.

- Break out of the innerloop when `j == 5`
- Break out of the outerloop when `i == 6`
- Print the values of `i` and `j` at any iteration

If you have completed this lab, alter your program so you can break out from the outerloop when `i == 8` in the innerloop

5

Go funcs

5.1 Functions

Simple functions

In this chapter we are going to practice with Go's functions //

Please work from your prj directory. //

The following Go program asks for a name and displays a greeting message. Note that the the 'fmt' package also contains the Scanf function that allows you to read input. As strings are immutable the `name` variable of type string is passed by `reference` to the Scanf function

Save this program in a file called: `input-rev.go`

```
package main

import "fmt"

func main() {

        var name string

        fmt.Print("Enter your first name: ")
        fmt.Scanf("%s", &name)

        fmt.Printf("Good day %s\n", name)
}
```

Create a function called `reverse` with declaration `func reverse(s string) string {}` that will return the s string in reverse order. E.g. if s == "gopher" the returned string will be "rehpog". Alter the program so the user's name will be printed in reverse

Variadic functions

Variadic functions consume a variabele nr of parameters. Below is an example of a declaration of an Avg function with variadic arguments.

```
func Avg(values ...float64) float64
```

Write a program in Go that uses a variadic function to calculate the average of a given set of numbers. Your function should be able to ingest individual numbers as well as slices.

Extra: write another variadic function to calculate the variance of a given set of numbers. You can re-use parts of the Avg() function.

Extra credit for solutons that are recursive. :)

Recursive functions

Recursive functions call themselves and can be a very powerful tool when used wisely. In this section we are going to practive with recursive functions in Go //

Create a Go program that calculates the Fibonacci nr using a recursive function called Fibonacci with declaration: `func Fibonacci(n int64) int64` //

The Fibonacci sequence has the following properties

- n == 0 > F = 1
- n == 1 > F = 1
- n > 1 > F= F(n-1) + F(n-2)

Example: n = 0.. i14 => F: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377

Extra: how would you optimize this recursive function? Try to implement that optimization. Tip: use the Time package to see if the optimization does improve the performance

6

Go packages

In this section we will learn:

- How to use a standard package
- How to use an external package
- How to use vendoring
- How to create and publish your own package

6.1 Using standard packages

Please create a directory called `web` and change into it

Create a file called `web.go` in this directory with the following content

```
package main

import (
        "fmt"
        "net/http"
        "time"
)

func main() {
        http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
                fmt.Printf("Request received")
                w.Write([]byte("Hello world"))
        })
        port := 8080
        s := &http.Server{
                Addr:           fmt.Sprintf(":%d", port),
                ReadTimeout:    10 * time.Second,
                WriteTimeout:   10 * time.Second,
                MaxHeaderBytes: 1 << 20,
        }
        log.Printf("Listening on port: %d\n", port)
        log.Fatal(s.ListenAndServe())
}
```

This program will implement a tiny webserver using the internal/standard package `net/http`. The internal package `time` is used for the timeout parameters of the http.Server method.

Test and run this Go program.

You will most probably get a complaint from Go that it cannot operate out of `$GOPATH/src` and you will need do have a go.mod file. Let's do that:

Execute: `go mod init web` in your directory and try again.

Excercise: change the program such that it will use the log package to writeout the status message. (`Request received`)

Test and run again

6.2 Using external packages

Let's try to use an external package.

Please create a directory called `godb` and change into it

Run `go mod init godb` inside this directory

Create a file called `godb.go` in this directory with the following content

```
package main

import "database/sql"
import "github.com/gopsql/standard"
import _ "github.com/lib/pq"
import "fmt"

func main() {
        c, err := sql.Open("postgres", "postgres://localhost:5432/gopsql?sslmode=disable")
        if err != nil {
                panic(err)
        }
        conn := &standard.DB{c}
        defer conn.Close()
        var name string
        conn.QueryRow("SELECT current_database()").Scan(&name)
        fmt.Println(name)

}
```

Install the gopsql and libpq packages using the following commands in your directory:

```
go get github.com/gopsql/standard
go get github.com/lib/pq
```

Try to run the program. You will get an error about the DB connection, but should not get an error due to missing packages

6.3 Vendoring

Vendoring is the practice in Go to isolate the packages needed for building your library/package or application with static/enclosed package version. E.g. the packages are not downloaded from the repo's but enclosed/packaged in a special directory called vendor within your project.

Please change into your `godb` directory again. With the following command we make a snapshot of the currently used packages and have them placed in a directory called `vendor` in your project

```
go mod vendor
```

Check the contents of the directory and verify that the two external packages (gopsql and libpq) are indeed snapshotted into this directory.

To build your project strictly with the vendored packages use the following command:

```
go build mod -vendor
```

6.4 Create and publish your own package

In this section we learn how to create and publish our own package. In order to publish your package you will need to have a gitlab or github account.

Pre-requisites:

- Create a new repository using your own personal account on github or gitlab. (e.g. gitlab.com/pascal71/pubmod.git)
- Make sure this repo is publically accessible for this lab
- Clone this empty directory in your workspace

E.g: (change the url for your github/gitlab account, package name etc). Here the package is called pubmod

```
mkdir projects
cd projects
git clone http://thegitcave.org/pascal/pubmod.git
Cloning into 'pubmod'...
warning: redirecting to https://thegitcave.org/pascal/pubmod.git/
warning: You appear to have cloned an empty repository.
```

- Initialize your new module/package using go mod init <giturl-to-your-new-package-repo>

```
cd pubmod
go mod init thegitcave.org/pascal/pubmod
go: creating new go.mod: module thegitcave.org/pascal/pubmod
```

- Create a package pubmod.go
- In this package create a function called Hello() that will return a greeting as a string

Example

```
package pubmod

func Hello() string {
        return "Hello from pubmod!"
}
```

Now add the go.mod file and your go code to the repo

```
git add .
git commit -m 'Initial commit'
[master (root-commit) f4f4228] Initial commit
 2 files changed, 10 insertions(+)
 create mode 100644 go.mod
```

```
 create mode 100644 pubmod.go
git push
warning: redirecting to https://thegitcave.org/pascal/pubmod.git/
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 48 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 374 bytes | 374.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To http://thegitcave.org/pascal/pubmod.git
 * [new branch]      master -> master
```

Now it's time to see if we can consume our 'published' package

```
mkdir usepubmod
go mod init usepubmod
go: creating new go.mod: module usepubmod
go: to add module requirements and sums:
        go mod tidy
```

Now create your Go program as usepubmod.go that will consume the pubmod package from your Git repo

E.g:

```
package main

import "fmt"
import "thegitcave.org/pascal/pubmod"

func main() {
        fmt.Println (pubmod.Hello())
}
```

Do not forget to change the import of pubmod so it refers to your own github repo

Execute the following commands to get your freshly published package

```
go mod tidy
go: finding module for package thegitcave.org/pascal/pubmod
go: downloading thegitcave.org/pascal/pubmod v0.0.0-20220209175630-f4f42284e450
go: found thegitcave.org/pascal/pubmod in thegitcave.org/pascal/pubmod v0.0.0-20220209175630-f4f42284
```

Test by running the program

- Add functions to your new package and bump the version tag

- Try to update your package by modyfing the go.mod file and using go mod tidy

This concludes our labs on Go packages

7 Go testing

In this section we will learn:

- How to test a Go program
- How to create simple testcases
- How to create tabledriventest cases

7.1 How to test a Go program

Please create a directory called `sumseq` and change into it

Create a file called `sumseq.go` in this directory with the following content

```
package main

import "fmt"

func Sumseq(n int) int {
        sum := 0
        for i := 1 ; i <= n ; i++ {
                sum = sum + i
        }

        return (sum)
}

func main() {

        fmt.Println(Sumseq(5))
}
```

This program will probably look familiar. It sums up the range of nrs between 1 and n.

Test and run this Go program.

Now let's write some tests for it. Kindly create a file called `sumseq_test.go` in the same directory with the following contents:

```
package main

import "testing"

func Test_Sumseq(t *testing.T) {
```

```
        x := 2
        want := 3
        got := Sumseq(x)
        if want != got {
                t.Logf("Testing Sumseq of: %d ", x)
                t.Errorf("Sumseq was incorrect want: %d, but got: %d", want, got)
        }
}
```

This introduces a testcase where we expect an input of 2 lead to a result of 3 from the Sumseq function. Test this with:

```
go test
```

Introduce a bug into sumseq.go by altering the program, for example like this:

```
package main

import "fmt"

func Sumseq(n int) int {
        sum := 0
        for i := 1 ; i <= n ; i++ {
                sum = sum + i
        }

        // this bug was introduced intentionally

        return (sum+1)
}

func main() {

        fmt.Println(Sumseq(5))
}
```

Test again, verify the reported error and after that correct the program again.

7.2 Table driven testing

The following test program demonstrates the use of table driven testing with Go:

```
package main

import "testing"

func TestsumseqTbl(t *testing.T) {
  tables := []struct {
        x int
        n int
  }{
        {0, 0},
        {1, 1},
        {2, 3},
  }

  for _, table := range tables {
     total := sumseq(table.x)
        if total != table.n {
           t.Errorf("sumseq of (%d) was incorrect, got: %d, want: %d.", table.x, total, table.n)
        }
  }
}
```

- Add this test to your `sumseq_test.go` file and execute the test again using `go test`
- Add testcases for input x=3,4,5 and 6 and execute the test again.
- In a previous excercise you created a recursive function for the Fibonacci sequence. Please create a tabledriventest for this function and test it

8

Go Methods

In this section we will learn:

- How to use types in Go
- How to use methods in Go

8.1 Types

Please create a directory called `user` and create a file called `user.go`

The `user.go` file should get the following content:

```
1  package main
2
3  import (
4      "fmt"
5  )
6
7  type User struct {
8      Name  string
9      Email string
10 }
11
12 // String satisfies the fmt.Stringer interface for the User type
13 func (u User) String() string {
14     return fmt.Sprintf("%s <%s>", u.Name, u.Email)
15 }
16
17 func main() {
18     u := User{
19         Name:  "Anne",
20         Email: "anne@itgildelab.net",
21     }
22
23     fmt.Println(u)
24 }
```

Analyze and run this code. Important is to notice the creation of the User type and the 'stringer' method. The fmt.Print functionality of the fmt package is using it to identify the type and print the data accordingly. We will use this in the next excercise.

8.2 Types and Methods

Types can be very powerful in Go. We are going to create program that is able to convert the imperial unit feet to meters and vice versa.

Please create a directory called `meters` and create a file called `meters.go`

Create a Go program using userdefined Types and Methods that converts meters to feet and v.v. and when one of these types is printed it will print the value along with the correct unit. So if the type is Meters it wil print 'm' and if it will be Feet it wil print "ft". Extra credit for foot vs. feet.

Create

- A type called Meters representing meters
- A type called Feet representing feet
- A method called String for Feet to print this type in a correct way with units
- A method called String for Meters to print this type in a correct way with units
- A method ToMeters that converts Feet to Meters (1 ft = 0.3048m)
- A method ToFeet that converts Meters to Feet
- The converted value should also have the converted Type (verify with print)

8.3 Methods and interfaces

In this section we are going to create interfaces that are satisfied by methods

Please create a directory called `shapes` and create a file called `shapes.go`

Create a Go program called shapes.go that:

- Defines a shape called Square with property side
- Defines a shape called Triangle with properties a,b,c representing the length of each side
- Implements a method area that returns the area of the shape (triangle -> heron's formula)
- Implements a method perimeter that returns the perimeter of the shape (triangle -> a+b+c)
- Defines an interface called shape which comprises the methods area and perimeter
- Implements a function Area that has a variable of type shape interface and returns the area for either triangle or square
- Implements a function Perimeter that has a variable of type shape interface and returns the perimeter for either triangle or square

Extra: create a shape called circle and implement methods for area and perimeter and integrate it in the interface shape

Nota bene: Heron's formula for calculating the area/surface of a triangle:

```
// Heron's formula :)

s := (t.a + t.b + t.c) / 2
area := math.Sqrt((s * (s - t.a) * (s - t.b) * (s - t.c)))
```