Go Homework – note: copying & pasting won't work

1. The Basics
    a. Set up Go on turing as instructed on the Go Language handout.

    b. In your home directory on turing, create a public_html (use lowercase and use an underscore to separate words!). Be sure to correctly set your file permission (should be 705 – or rwx for the first, --- for the middle, and r-x for the third). Note: It might be okay to have the last number be 1 instead of 5. After completing the homework, try it and see.

    c. Now, within your public_html folder, create a Golang folder (Note the subtle change in folder name from what is at your home directory level – this is to differentiate from the top-level Go folder you created in your home directory).

    d. Create a Go file, ***yourLastNameFirstInitialGoBasics.go*** and add the following variables (recall that & creates a reference assigned to a pointer; * is an alias to the reference). I recommend you compile and execute after each part is working before moving to the next.

```
package main
import (
   "fmt"
 )

func main() {
    fmt.Printf("hello,world\n")

    m := 2
    n := &m
    *n = *n *8

    fmt.Printf("m is %d and n is %d\n", m, *n)

  const (
     DaysInYear = 365
     DaysInLeapYear = 366
    )

    fmt.Printf("Days in Year %d\nDays in Leap Year %d\n", DaysInYear, DaysInLeapYear)
}
```

    Save the file in your public_html/Golang folder. To execute, open PuTTY and navigate to your go file (cd public_html/Golang) , compile and execute (from here on I will shorten to just GoBasics but you need to add your last name to the front): **go run GoBasics.go**

e. Now, import the strings package and create four variables: first, second, third, and fifth. To each of these variables, output various string functions. NOTE: do not delete previous code as you may get an error if you are not using an imported package:

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

func main() {
    first := "The quick brown fox"
    second := "jumped over the lazy dog"
    third := first
    third += second
    fifth := "          jumped over the lazy dog.   "

    fmt.Printf("Lower: %s\n", strings.ToLower(first))
    fmt.Printf("Upper: %s\n", strings.ToUpper(second))
    fmt.Printf("First Word: %s\n", strings.Title(second))
    fmt.Printf("Trim: %s\n", strings.TrimSpace(fifth))
    fmt.Printf("Length: %d\n", len(second))
    fmt.Printf("Replace brown with red: %s: \n", strings.Replace(first, "brown",
   "red", -1))
    fmt.Printf("Substring: %s\n", second[16:])
    fmt.Printf("Contains over: %t\n", strings.Contains(second, "over"))
    fmt.Printf("Repeat lazy: %s\n", strings.Repeat(second[16:21], 3))
    fmt.Printf("Brown is at index: %d\n", strings.Index(first, "brown"))
    fmt.Printf("Compare Hello and hello: %d\n", strings.Compare("HELLO",
   "hello"))
}
```

f. Import the math and unicode packages and try the various math and unicode functions.

```
package main
import (
  "fmt"
  "strings"
  "math"
  "math/rand"
  "unicode"
 )
func main() {

    fmt.Println("\n----------\nMath Functions")
    fmt.Printf("Absolute Value: %f\n", math.Abs(-15))
    fmt.Printf("4-Cubed: %f\n", math.Pow(4, 3))
    fmt.Printf("Squareroot of 15: %f\n", math.Sqrt(15))
```

```
        fmt.Printf("Floor of 14.5: %f\n", math.Floor(14.5))
        fmt.Printf("Ceil of 14.4: %f\n", math.Ceil(14.5))
        fmt.Printf("Round of 14.5: %f\n", math.Round(14.5))
        fmt.Printf("15 mod 4 is %f\n", math.Mod(15, 4))
        fmt.Printf("Random integer 0 to 99: %d\n", rand.Int31n(100))
        if math.IsNaN(math.Sqrt(-2)) {
         fmt.Println("Not a number")
        }
        if unicode.IsDigit('5') {
         fmt.Println("5 is a number")
        }

        if unicode.IsNumber('!') {
         fmt.Println("5 is a number")
        } else {
         fmt.Println("! is NOT a number")
        }
    }
```

2. In order to render a Go file in html, you must use a template that your Go file references. In addition, this template should be in a ***templates*** folder. Because we have not installed on turing what is called a listen and serve module for Go, you are being assigned a specific port from which Go will render in your browser.
    a. On Blackboard, download the Go port numbers file. Look up your name and use the port listed. This port is your "localhost" You will use this in place of the 1111 I use throughout the course.
    b. Now, on turing, within your Golang folder, create a folder called ***templates***
    NOTE: When I list folders and filenames, these MUST BE exactly as written and MUST BE in the location given.
    c. Create two new files ***SportsTemplate.go.*** and ***Sports.html***
    d. Begin by importing the packages listed below – we will use log instead of format; we will also use the http package to render/write to an html page. In addition, we need the template package since we will be writing our html to a template (remember you need to declare a package – we will use package main):

```
package main

import (
   "log"
   "net/http"
   "text/template"
)
```

    e. Now, below your imports, define the *struct* related to sports and their championship game:

```
type SportsLeagues struct {
        Sport string
        Championship string
}
```

f. Next we will create the template relationship with our Go document. We use two functions from the template package we imported. First we use ***MUST*** to declare the connection. Second, we must tell the location of our template - use ***ParseGlob*** to parse the template. We could also use ParseFiles but later in the semester we will be using multiple files which requires ParseGlob, so that's what we will use here. Finally, notice the location we are looking at – all files should be in the ***templates*** folder, which you just created:

```
var tmpl = template.Must(template.ParseGlob("templates/*"))
```

g. We also need to define the function handler for rendering the http. This function expects two parameters – where we are ***writing to*** and where we are ***reading from***. This function will be invoked in our main function. It is in this function we will define a SportsLeagues slice, appending to it various sports and their championship game. Once defined, we are ready to write its contents to our html file. Note that I've named my function the same name as the html file – this is NOT required:

```
func Sports(w http.ResponseWriter, r *http.Request) {

 SL := []SportsLeagues{}
 SL = append(SL, (SportsLeagues{Sport: "Football", Championship: "Superbowl"}))
 SL = append(SL, (SportsLeagues{Sport: "NBA", Championship: "Championship Series"}))
 SL = append(SL, (SportsLeagues{Sport: "MLB", Championship: "World Series"}))
 SL = append(SL, (SportsLeagues{Sport: "FIFA", Championship: "World Cup"}))

  tmpl.ExecuteTemplate(w, "Sports", SL)
}
```

h. Now, we are ready to write our main function. Here, we will log that we have issued our request to the server. Then, we will execute the function with Go's http handler function. Finally, we initiate the listen and serve http function. Turing is set up to be listening for your port – when it receives the request from your Go file, it "serves" or runs your file.

NOTE: *1111* should be replaced with your port. YOU MUST USE THE PORT GIVEN IN THE EXCEL FILE ON BLACKBOARD.

```
func main() {
    go log.Println("Server started on: http://localhost:1111")
    go http.HandleFunc("/", Sports)
    http.ListenAndServe(":1111", nil)
}
```

i. Finally, sftp your SportsTemplate.go file to your *Golang folder*.

j. We cannot run our file yet as we first must complete our template. Open up Sports.html in a text editor. In this file, we will use moustache syntax, which designates actions. All actions are required to have an end – like } closes a block in Java. We will begin by defining the name of the template, which is the name defined in our http function handler in the Go file. We include SOME of the html but will add to this in the next step.

```
{{ define "Sports" }}
<!DOCTYPE html>
```

```
<html lang = 'en'>
<head>
<title>Sports Leagues Championship Games</title>
</head>
<body>
  <center>


  </center>
</body>
</html>
{{ end }}
```

k. Between the center tags, we will define our table – ***table*** designates the beginning and ending of the table information, which has a header row (***thead***) and a body (***tbody***). Within each, we designate each row with ***tr*** (table row) and each cell/data with ***td*** (table data). Within the table note the ***{{ range . }}*** action – watch the spaces (***range*** – space – **.** (dot))! This creates a loop where we will step through each value of our slice created in the Go file. The dot is used to designate the reference of the current item – think of it like the ***this*** reference variable in Java. The table body is populated with the struct variables in SportsLeagues. Again, the dot references the current element in the slice – recall that each SportsLeagues element has 2 values – a Sport and a Championship.

Between the center tags add the following code:

```
<h2> Sports Championship Games </h2>
<table border=1>
   <thead>
    <tr>
      <td>Sport</td>
      <td>Championship Game</td>
    </tr>
   </thead>

   <tbody>
   {{ range . }}
     <tr>
        <td>{{ .Sport }}</td>
        <td>{{ .Championship }}</td>
     </tr>
   {{ end }}
  </tbody>
</table>
```

l. Save your file and sftp Sports.html to your *templates folder*.

m. Now, from command line navigate to your ***Golang*** folder and compile and execute your go file: ***go run SportsTemplate.go***

If you have no errors, you should get your log statement saying that it is currently running on the port. Your now ready to see how it renders in a browser.

n. Open a browser. You will need to type in the absolute address. Otherwise, turing will assume it is a secure URL (https), which it is not:
http://turing.cs.olemiss.edu:***1111***/~***webid***
NOTE: 1111 is your port and webid is your own webid

o. Congratulations if all is working! Back in PuTTY, ctrl-c to quit running.

p. Now for the homework. Create two files, ***yourLastNameFirstInitialSECTemplate.go and SECConference.html*** In SECTemplate.go, similar to what you did for SportsLeagues, create a struct called Conference (don't forget to define your package main and import the necessary packages).

```
type Conference struct {
   School string
   Mascot string
   City string
   State string
   Division string
  }
```

q. Create a handler function to populate a slice called ***SEC***, containing the 14 SEC schools.

The following lists the information that should be used to populate your slice:

School	Mascot	City	State	Division
Auburn	Tigers	Auburn	AL	West
LSU	Fighting Tigers	Baton Rouge	LA	West
MSU	Bulldogs	Starkville	MS	West
Texas A&M	Aggies	College Station	TX	West
Alabama	Crimson Tide	Tuscaloosa	AL	West
Arkansas	Razorbacks	Fayetteville	AR	West
Florida	Gators	Gainesville	FL	East
Georgia	Bulldogs	Athens	GA	East
Kentucky	Wildcats	Lexington	KY	East
Ole Miss	Rebels	Oxford	MS	West
Mizzou	Tigers	Columbia	MO	East
South Carolina	Gamecocks	Columbia	SC	East
Tennessee	Volunteers	Knoxville	TN	East
Vanderbilt	Commodores	Nashville	TN	East

r. Create the template file to display all the schools (See Sample Output) – again, use your SportsLeagues example as a guide.
s. Submit GoBasics.go, SportsTemplate.go, Sports.html, SECTemplate.go and SECConference.html

3. Sample Output
NOTE: a) Be sure to include a title (not shown) b) include a header (shown), replacing 1111 with your localhost, and c) be sure ***Location displays the City, State as shown*** (there is a space after the comma)

Sample Output:

NCAA Southeastern Conference (host 1111)

School	Mascot	Location	Division
Auburn	Tigers	Auburn, AL	West
LSU	Fighting Tigers	Baton Rouge, LA	West
MSU	Bulldogs	Starkville, MS	West
Texas A&M	Aggies	College Station, TX	West
Alabama	Crimson Tide	Tuscaloosa, AL	West
Arkansas	Razorbacks	Fayetteville, AR	West
Florida	Gators	Gainesville, FL	East
Georgia	Bulldogs	Athens, GA	East
Kentucky	Wildcats	Lexington, KY	East
Ole Miss	Rebels	Oxford, MS	West
Mizzou	Tigers	Columbia, MO	East
South Carolina	Gamecocks	Columbia, SC	East
Tennessee	Volunteers	Knoxville, TN	East
Venderbilt	Commodores	Nashville, TN	East