Homework 2: Writing Functions in Go
02-201: Programming for Scientists

In this homework, you will write several functions to compute various quantities. You should write your functions at the indicated locations in the provided file `functions.go`. You must work **independently** on this homework. You can discuss general solution techniques with your classmates, but must not show or share code with others or see others' code.

Due: Friday, September 25, at the end of recitation.

Tip: When you submit, you must put all your functions into the `functions.go` file. However, while you are writing them, it might be easier to write them all in separate files so you can use `go run` to test them one at a time. When you submit, you must *not* have a `main()` function in the file that you submit.

Tip: Read the code in `functions_test.go` to get some idea about what the signatures of your functions should look like.

1. Set up your Go workspace if you haven't already

Create a directory called `go` wherever you want to store your Go files. For this example, I'll choose `/Users/carlk/Desktop/go`. Then open a command line and run the following on Linux or Mac:

```
carlk$ export GOPATH=/Users/carlk/Desktop/go
carlk$ cd $GOPATH
```

or the following for Windows:

```
> set GOPATH=C:\Users\carlk\Desktop\go
> cd %GOPATH%
```

You should replace the path `/Users/carlk/Desktop/go` above with the path to wherever you want to put your Go workspace.

Inside of that `go` directory, you should create a `src` directory. Each homework will be placed inside a subdirectory of this `src` directory.

2. Get the template for this assignment

You can download the `functions.go` and `functions_test.go` templates in a zip file from Piazza. Put the `functions` directory inside your `go/src` directory described above.

3. Write the following functions in the file `functions.go`

3.1 Sum of the first n integers

Gauss' formula for the sum of the first n integers is

$$\frac{n(n+1)}{2}.$$

Write a function to compute this quantity for any positive n. To do this, you should edit functions.go file where indicated to include the body of the function that has been started:

```
func SumOfFirstNIntegers(n int) int {
    // WRITE YOUR CODE HERE
}
```

3.2 Time to Run

Write a function `TimeToRun(marathonHours, marathonMinutes, miles)` that takes: the time a runner ran a marathon in possibly fractional hours (`marathonHours`) and possibly fractional minutes (`marathonMinutes`) and a possibly fractional number of miles and return the time in **days** it should take the runner to run miles if he or she runs at the same pace as they did in the marathon.

For example: `TimeToRun(3.1, 23.2, 107.1)` should return 0.5938.

Your function should also print out the answer in the format:

```
You could run 107.1 miles in 0.5938 days.
```

Recall that there are 26.2 miles in a marathon.

3.3 Generalized Fibonacci sequences

A generalized Fibonacci sequence is defined by two starting integers a_0 and a_1 using the rule:

$$a_i = a_{i-1} + a_{i-2}$$

for $i \geq 2$.

Write a function `GenFib(a0, a1, n)` that takes 3 integers and returns the nth number in the generalized Fibonacci sequence defined by a_0 and a_1.

For this and subsequent problems, you will have to write the function signature in addition to the body of the function. Please write your functions at the indicated places within the `functions.go` file.

3.4 Kth Digit

Implement a function `KthDigit(n,k)` that takes an integer n, and a positive integer k and returns the kth decimal digit of n, with digit number 1 being the rightmost (least significant) digit.

For example: `KthDigit(123, 1) = 3` and `KthDigit(124,4) = 0`.

Tip: Try not to use any loops, and try to use the `math` package.

3.5 Reversing Integers

Write a function `ReverseInteger(n)` that takes an integer, and returns the integer formed by reversing the decimal digits of n. For example:

- 1234 $\rightarrow$ 4321
- 20000 $\rightarrow$ 2
- 1331 $\rightarrow$ 1331
- -60 $\rightarrow$ -6

3.6 Growth of a population

Suppose we have a population of animals with birth rate r and a maximum population size K. We can model the size of the population, as a fraction of K, using the following equation:

$$x(t) = rx(t-1)\left[1 - x(t-1)\right]$$

where $x(t)$ is the fraction (between 0 and 1) of the maximum population size at time t. The intuition behind this equation is that as the population gets closer to its maximum, the effective birth rate $r\left[1 - x(t-1)\right]$ falls. Write a function `PopSize(r, x0, max_t)` that prints out the size of the population $x(t)$ for $t = 0 \ldots max_t$.

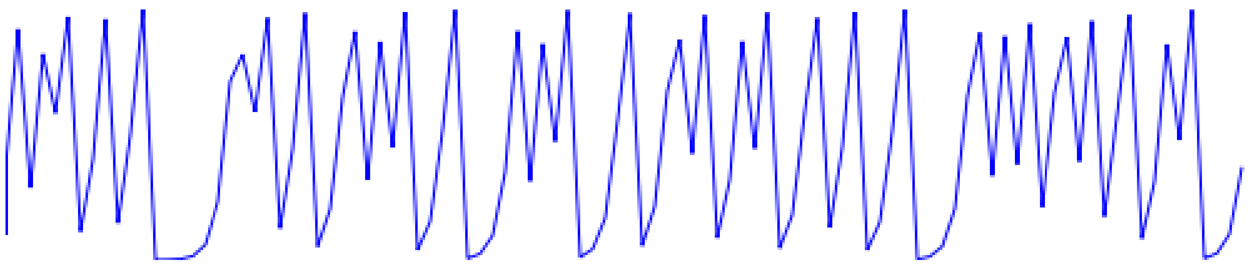
Your function should also return the final population size.

If $x(t)$ ever becomes negative, you should reset it to 0; if $x(t)$ ever increases past 1.0, you should reset it to 1.0.

Example output of `PopSize(2.9, 0.1, 20)`:

```
0.261
0.5593491
0.714785284554651
0.5912151164624551
0.7008714273333482
0.607986942075084
0.6911825789896902
0.6190027423234675
0.6839312072265337
...
```

An interesting thing about this equation is that very complex behavior can be generated depending on the parameter r:



4. Test your functions

As part of this assignment, we have provided a file `functions_test.go` that contains several test functions that call the functions you wrote above. These test functions can be run by executing the following command from within the directory containing the `functions.go` and `functions_test.go` files:

```
go test -v
```

This will run each of the Test... functions in the file `functions_test.go`. If your functions return the correct values, the output of the `go test -v` command will end with:

```
PASS
ok   functions 0.005s
```

This tells you that all the tests passed and ran in 0.005 seconds.

If there are any errors, go back and revise your functions. If you have syntax errors, these will be printed out by the `go test -v` command.

Tip: Write the signatures for all the functions first (with an empty function body or a function that just returns 0). The "go test" command won't work until you have at least the correct signature for each of your functions.

Tip: Edit the functions in `functions_test.go` to test your functions in different ways.

Tip: Do not assume that if `go test -v` reports PASS that your functions are 100% correct. There many be other inputs on which your code fails; you should test it under various inputs.

5. Submit your work to Autolab

Submit just the file `functions.go` containing your solutions to AutoLab. The file you submit should **NOT** contain a `main()` function.