Day 00 - Go Boot camp

Statistics being handy

Contents

Chapter I

General rules

- Your programs should not quit unexpectedly (giving an error on a valid input). If this happens, your project will be considered non functional and will receive a 0 during the evaluation.
- We encourage you to create test programs for your project even though this work won't have to be submitted and won't be graded. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded.
- If your code is using external dependencies, it should use Go Modules for managing them

Chapter II

Rules of the day

- You should only turn in `*.go` files and (in case of external dependencies) `go.mod` + `go.sum`
- Your program should accept a sequence of numbers separated by newlines to its standard input. One number is also a sequence.
- You may assume it should only work with integers (the output can be floats though, rounded to 2 decimal places).
- Nevertheless it should print a meaningful error message without runtime panic if fed some unexpected input, say, number out of bounds, letter characters or empty string.
- Your code for this task should be buildable with just `go build`

Chapter III

Intro

Go isn't generally considered a language of Data Science. But it doesn't mean that it can't crunch numbers. In fact, it's comparable to C on basic tasks. Besides, it can be a lot easier to write, partially because GC handles memory management and also Go's standard library is pretty good. We're constantly taught that it can be a bad idea to just trust the gut when dealing with important data. To make heads or tails of a sample of numbers it's usually better to use statistical approach. Data sometimes be can be deceptive, too, like, for example, Anscombe's quartet, but the more metrics we get - the more weighted decision we will able to make at the end, isn't it?

Chapter IV

Exercise 00: Anscombe

So, let's say we have a bunch of integer numbers, strictly between -100000 and 100000. It may probably be a large set, so let's assume our application will read it from a standard input, separated by newlines. Right now let's think of four major statistical metrics that we can derive from this data, so by default we can print all of them as a result, for example, like this:

```
Mean: 8.2
Median: 9.0
Mode: 3
SD: 4.35
```

The order and actual format doesn't really matter as long as we can understand which is which. A couple of notes, though:

1. Input data may or may not be sorted. You don't need to write your own sorting algorithm, luckily, Go already has one in standard library and it works for integers.
2. Median is a middle number of a sorted sequence if its size is odd, and an average between two middle ones if their count is even.
3. Mode is a number which is occurring most frequently, and if there are several, the smallest one among those is returned. You may think of using some structure for storing number counts, and some Go standard container may help.
4. You can use both population and regular standard deviation, whichever you prefer.
5. Calling someone "average" can be mean.

It will also make sense for user to be able to choose specifically, which of these four parameters to print, so need to implement this as well. By default it's all of them, but there should be a way to specify whether print just one, two or three specific metrics out of four when running the program (without recompilation). There is a built-in module in standard library that allows you to parse additional parameters.