← APG1 Bootcamp. Day07

Revie

Submit the project

Finish project

Survival camp

Show all

APG1 Bootcamp. Day00	100%
APG1 Bootcamp. Day01	100%
APG1 Bootcamp. Day02	100%
APG1 Bootcamp. Day03	0%
APG1 Bootcamp. Day04	100%
APG1 Bootcamp. Team00	0%
APG1 Bootcamp. Day05	
APG1 Bootcamp. Day06	
APG1 Bootcamp. Day07	
APG1 Bootcamp. Day08	
APG1 Bootcamp. Day09	
APG1 Bootcamp. Team01	

Private Git project

ssh://git@repos-ssh.21-school.ru:2289/students/Go_Day07.ID_376229/mlarra_student....

Copy link Open

Task

Day 07 - Go Boot camp

Moneybag

Contents

Chapter I

General rules

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 code for this task should be buildable with just `go build`
- All your tests should be runnable by calling standard `go test ./...`

Chapter III

Intro

"There are several areas where we consider reliability and speed critical. Areas that directly affect people's lives - medicine, aircraft safety, finance. Of course that means that we thoroughly look through every detail of our product before releasing it to the public. Ladies and gentlemen, I present you...the Moneybag!"

Chapter IV

Exercise 00: King's Bounty

You keep listening to CEO's voice, but your eyes are looking at the code on your laptop.

Sometimes it seems like people will always use coins to pay for stuff. In laundromats, vending machines or music boxes it is still normal to accept only pieces of metal as payment. But people sometimes hate staying in lines and waiting for someone else collecting coins and putting them in one by one. Why can't people just always use a minimal amount of coins to avoid slowing everyone else down?

This is a pretty well-known problem and your colleague already wrote a code and uploaded it to you for a review:

```
func minCoins(val int, coins []int) []int {
    res := make([]int, 0)
    i := len(coins) - 1
    for i >= 0 {
        for val >= coins[i] {
            val -= coins[i]
            res = append(res, coins[i])
        }
        i -= 1
    }
    return res
}
```

It accepts a necessary amount and a sorted slice of unique denominations of coins. It may be something like [1,5,10,50,100,500,1000] or something exotic, like [1,3,4,7,13,15]. The output is supposed to be a slice of coins of minimal size that can be used to express the value (e.g. for 13 and [1,5,10] it should give you [10,1,1,1]).

The issue is, you have a gut feeling there is something wrong with this code. Your goal here is to write several tests (in `*_test.go` files) for this code, that will show it producing wrong results. Also, you need to write a separate function (you should call it `minCoins2`) which will have the same parameters, but will handle those cases successfully. In case duplicates are present in a slice of denominations or it is not sorted, the function should still give a correct result. In case it is empty, an empty slice should be returned.

Chapter V

Exercise 01: Need for Speed

Now as you have new version of code from EX00, let's test it for performance. Your goals here are:

- get a list of top 10 functions in your code (calling your function with some test data) that your CPU spends the most time executing (you should use Go's builtin tools for that). Submit that list as file `top10.txt`
- write a benchmark version of your tests that will compare the performance of your new code vs. the old one, especially while using relatively big denomination slice. If you find any more optimizations during this phase, feel free to submit newer version of your `minCoins2` function calling it `minCoins2Optimized` (not a required step)

Chapter VI

Exercise 02: Elder Scrolls

Now, as you've fixed the bug and wrote some tests for your code, it's time to generate some documentation for it. Describe in comments in your code how is your solution different from the default one and what optimizations did you use. Then, use any tool that you'll manage to find to generate the HTML documentation based on those comments.

Directions on how to reproduce documentation generation should also be included in comments. Saving HTML pages from the web browser is considered cheating (even though not strictly prohibited, so if you couldn't do it any other way just write it explicitly in the comments).

Submit generated documentation (HTML files + static, like images, js and css) packed into a `docs.zip` archive.