Homework 3 Part 1 Instructions

September 10, 2018

1 Overview

We'll start with some more Euler problems in this one - and these can be done pretty quickly with a little trick! Part 2 of this homework will be the more difficult part, but we'll first learn an essential skill for it: file reading.

Again, do not mess with `testcases.py` or `PHGN311TestLib.py`. These are the files I have written for you all to be able to use test cases for immediate feedback. **Tampering with these files to force passing tests will result in a zero for the assignment**. There will be no way to regain those points if this happens. If there are any issues using these files email me ASAP (dscarbro@mines.edu).

Partial credit will be given when appropriate. If tests fail but style is good and code is close or on the right track, I will give partial credit. At best, make sure your code doesn't crash when it runs (failing tests are ok). If you can't get code to not crash but still want it considered for partial credit then comment out the troublesome parts. **Test your submission to make sure it does not crash!** I was lenient on this in homework 01, but will start applying a penalty for this now.

2 Score and time limit Summary

- Problem 16 - 18 pts
  - Test 1 - 15 pts - 30 seconds
  - Style - 3 pts
- Problem 20 - 18 pts
  - Test 1 - 15 pts - 30 seconds
  - Style - 3 pts
- Problem 25 - 18 pts
  - Test 1 - 15 pts -120 seconds
  - Style - 3 pts
- Correct submission - 10 pts

Total score: 64 pts

BONUS POINTS!!

If you solve problem 25 with an algorithm that doesn't just check numbers one by one, then there is the opportunity for up to 15 bonus points. Things that will get you these bonus points are: making it so that you can eliminate a large number of steps, implementing a way to make it so intermediate fibonacci number steps aren't recalculated every time, or anything else that speeds it up. Part of this will include commenting the bits that are unique to your implementation. If you are going for bonus points, add a note to the grading notes in your header comments.

3 Using the test suite

NOTE: Mostly the same, with adjustments again

Start by navigating a terminal to the directory containing all the .py files for this assignment. Enter `ls` to see all the files in the directory. To run the test suite we will use `testcases.py`. This program utilizes handy command line arguments. This lets you determine certain variables as the program starts up, a really common feature of programs meant for use in the terminal. All programs with this feature should have some "help" with it. Now run the command

```
python ./testcases.py --help
```

or

```
python ./testcases.py -h
```

`Note`: if your `python --version` is 2.7 then be sure to run with `python3` instead.

Now looking at the result, the help message says that this will run our test cases. Now look where it says

```
-p 0,16,20,25 [0,16,20,25 ...], --part 0,16,20,25 [0,16,20,25 ...]
```

These are the long (`--part`) and short (`-p`) commands to specify what part or parts of the assignment we want to run the test cases for. The numbers in curly braces represent the choices we have for this argument, and these choices repeated in the following square brackets means that there can be multiple choices. Think of curly braces as a *dictionary* of unique options, and the square brackets as a *list* of whatever combination of choices you want from the dictionary. The help also says we can use 0 for the parameter to run all tests, or just omit `-p` or `--part` entirely (meaning there is a default defined - 0 in this case). Try running

`python ./testcases.py` *or* `python ./testcases.py -p 0`

```
python ./testcases.py --part 16
```

```
python ./testcases.py -p 20 25
```

If your test case is running for a while and you want it to stop, enter a Ctrl-Z/Ctrl-C into the terminal. This usually stops the program, but you may find it's still using resources if you check the task manager (or use `top` on a Linux machine (and maybe Mac?)). If you can see Python is using a ton of resources, kill it in the task manager (or test your growing comfort with the shell and see what you can find

online). On Linux check out the `kill` and `pkill` commands.

4 Euler Problems

You will start this assignment with Euler problems 16, 20, and 25. For the first two the calculation part is straightforward. Python handles really long numbers quite smoothly - if you were doing this in C++ you might actually hit the upper limit of how long an int type can be. In C++ you'd fix this by declaring a variable with the type "long". How creative.

The tricky part for 16 and 20 is getting each digit from the resulting number. Here's a hint - if you cast this result to a string, you can index it, and take any piece of it and cast back to the int type. This can be done in just a little bit of code so if your solution is getting lengthy take a step back and think about how you can work with strings. Remember that a string is a *list* of

For problem 25 you can actually brute force this and the result will come out pretty quickly. If you're stuck on figuring out the length maybe think about something you could do by casting it to a string again. If you want to, try to implement a faster method that doesn't check every single Fibonacci number in order. There's lots of different things you can do here, and I will give some bonus points for implementations that do this. Just don't forget your comments!

When you're developing your functions, it is totally ok to run them with print statements, force values to see what is going on in a loop, etc. Just be sure to fix all of this before you submit. Since we are building a library of functional tools like a special calculator, each function should run without any printing. (Imagine solving a complex problem on a calculator and having tons of intermediate steps flooding your screen - not great). Any uncommented print statements in your submission **will result in a penalty**.

5 Submission

It is EXTREMELY IMPORTANT to me that you do this correctly. Plz.

You should have a file structure as follows:

```
working_directory
        username_hw03_part1
                problem16.py
                problem20.py
                problem25.py
                testcases.py
                PHGN311TestLib.py
```

The `working_directory` doesn't matter, work wherever fits your organization style. Everything from `username_hw03_part1` matters. **Match this name exactly** but with your username. **NOT** `username_HW03_part1`, **NOT** `username_hw03part1`, **NOT** `username_hw3_part1`...

This attention to specifics is very important in computing, it can help avoid bugs in code, but also prevent accidental damage to systems or sensitive data. This will become clearer the farther you go into computing. Pay attention to these details and you will not have any penalties!

To submit, zip up `username_hw03_part1`. **DO NOT** select all the python files at once and zip those, zip the whole folder. **DO NOT** have extra levels of folders. For example the following structure would result in a penalty:

```
working_directory
        username_hw02_part2.zip
                username_hw02_part2
                        username_hw02_part2
                                problem7.py
                                problem9.py
                                ..........
                                testcases.py
                                PHGN311TestLib.py
```

When you zip up your folder for submission, you can verify that its right. Double click on the zip folder and you can still see what's in it without re extracting the folder. On the first level, you should see a folder with the name following `username_hw03_part1`. If you look inside that, you should see your python files. If for example you see python files immediately after looking inside the first level of the zip folder, it's not correct. If you have to look into the zip folder then a folder then *another* folder, it's wrong. Double check this! Ask your classmates for help, send me an email, or check in office hours. Me checking that your submission structure is right now saves a lot of time later. I don't want to penalize anyone for this, but we have to be able to follow specifications!

Have fun and keep up the great work!