Lab 05 (Optional) Specification – A Hand-on Exercise to practice Recursion
50 points

Due by: 04/28/2021 8:00 AM

Lab Goals

- Learn to develop Recursive Implementation and Tree.
- Do a simple exercise to practice Recursive Implementation.

Learning Assignment

If you have not done so already, please read all of the relevant "GitHub Guides", available at the following website:
`https://guides.github.com/`
that explains how to use many of the features that GitHub provides. This reading will help you to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, you should also read:

- **GT chapter 05, 5.2,5.3,5.4**

Assignment Details

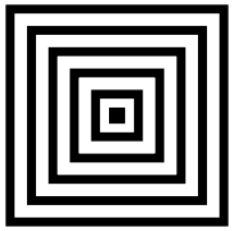
Now that we have discussed some basics of Recursion together in the last few lectures, it is your turn. In this lab, you will practice a variety of functionality to retain the knowledge of Recursive Coding that had been covered so far. This includes modifying one or more code files to implement a series of functionalities. At any duration during and/or after the lab, students are recommended to team up with the Professor and the TL(s) to clarify if there is any confusion related to the lab and/or class materials.

Students are recommended to get started with this part in the laboratory session, by discussing ideas and clarifying with the Professor and the Technical Leader(s). It is acceptable to discuss high-level ideas with your peers, while all the work should be done individually. Late submission is accepted for the part(s) in this section, based on the late policy outlined in the course syllabus.

It is required for all students to follow the honor code. Some important points from the class honor code are outlined below for your reference:

1. Students are not allowed to share code files and/or other implementation details. It is acceptable to have a healthy discussion with your peers. However, this discussion should be limited to sharing ideas only.
2. Submitting a copy of the other's program(s) is strictly not allowed. Please note that all work done during lab sessions will be an opportunity for students to learn, practice, and master the materials taught in this course. By doing the work individually, students maximize the learning and increase the chances to do well in other assessments such as skill test, exams, etc · · ·

Part-1 Developing a Recursive Tree and Implementation (50 points)



As summarized in wiki, the Tower of Hanoi (also called the Tower of Brahma or Lucas' Tower and sometimes pluralized as Towers, or simply pyramid puzzle) is a mathematical game or puzzle. It consists of three rods and a number of disks of different diameters, which can slide onto any rod. The puzzle starts with the disks stacked on one rod in order of decreasing size, the smallest at the top, thus approximating a conical shape. The objective of the puzzle is to move the entire stack to the last rod, obeying the following simple rules:

1. Only one disk may be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or an empty rod.
3. No disk may be placed on top of a disk that is smaller than it.

We already implemented the Tower of Hanoi problem in Python and had hand-drawn the recursive tree for this problem. Reve's puzzle is a variation to the towers of Hanoi problem. Instead of 3 rods we have 4 rods in the Reve's puzzle. Otherwise the problem is the same. The Frame-Stewart algorithm for the Reve's puzzle use the following solution to the towers of Hanoi:

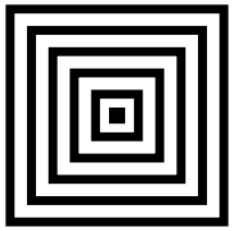
Suppose you need to move n disks from pole 1 to pole 4, with poles 2 and 3 as extras. If there is a single disk, Frame-Stewart will just move the disk, from pole 1 to pole 4. If there is more than one disk the algorithm has 3 steps.

1. First, recursively move the top n–k disks from pole 1 to pole 2.
2. Second, move the remaining k disks from pole 1 to 4, using the towers of Hanoi algorithm on just poles 1, 3, and 4.
3. Last, recursively move the n–k disks from pole 2 to 4.

This is similar in structure to the towers of Hanoi, except that in the middle step more than one disk is moved. Draw the recursive to figure out the solution and also it is very imortant to what is the value for n and k in the Frame-Stewart algorithm provided above. Once you figure out the logic, modify the starter code in the file named `reves.py`. You are required to fill out the code in the recursive method **reve**. The expected solution should be able to move around all the disks (assume number of disks = 5) from the left pole to right pole. The two poles in the middle are auxilary (supportive).

You can either do Part-1 or Part-2. Full points will be awarded if you complete atleast one part correctly. PS next page for Part-2 question.

Part-2 Developing a Recursive Tree and Implementation (50 points)



Modify the starter code to compute the sum of the values of all the elements in a given list. A starter code in `rsum.py` file is provided in the lab repository. The starter code generate a list automatically with a series of randomly generated values. You are required to fill out the **sum** method that is expected to return the sum of all the values in the list. I am not expecting you to use a for/while loop (iterative) logic. You are not allowed to use any built-in functions such as sum to calculate the sum of values in the list. I am only expecting to use a recursive logic to compute the sum and return this as an output of this method. This is similar to how we compute the minimum element in the list recursively in class?

Part 03 - Honor Code

Make sure to **Sign** the following statement in the `honor-code.txt` file in your repository. To sign your name, simply replace Student Name with your name. The lab work will not be graded unless the honor code file is signed by you.

This work is mine unless otherwise cited - Student Name

Submission Details

For this assignment, please submit the following to your GitHub repository by using the link shared to you by the Professor:

1. `reves.py` and/or `rsum.py` file.
2. A signed honor code file, named `honor-code.txt`.
3. To reiterate, it is highly important, for you to meet the honor code standards provided by the college. The honor code policy can be accessed through the course syllabus.

Grading Rubric

1. This lab is optional. If you complete this lab, the lab score will be used to swap with any of your lowest lab score. An opportunity to boost your lab scores.
2. There will be full points awarded for the lab if all the requirements in the lab specification are correctly implemented. Partial credits will be awarded if deemed appropriate.
3. Failure to upload the lab assignment code to your git repo will lead you to receive no points given for the lab submission. In this case, there is no solid base to grade the work.
4. There will be no partial credit awarded if your code doesn't compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before the due date and make sure to follow the honor code policy described in the syllabus. If it is a late submission, then it is the student's responsibility to let the professor know about it after the final submission in GitHub. In this way, an updated version of the student's submission will be used for grading. If the student did not communicate about the late submission, then automatically, the most updated version before the submission deadline will be used for grading purposes. If the student had not submitted any code, then, in this case, there are no points awarded to the student.

5. If you need any clarification on your lab grade, talk to the Professor. The lab grade may be changed if deemed appropriate.