CSE101: Introduction to Programming

Assignment 1 (Deadline: Thur, 26th Sep 2024, 11:59:59 pm)

Instructions:

- Assignments are to be done individually by each student. You should use VS code for developing the code - for writing functions you can take help from Copilot.
- ***This assignment is based on concepts covered till last week. You cannot use lists, dictionaries, sets etc. In some questions, lists are used - only for looping over them item by item. (You cannot use operations on list or functions like len on lists, index in a list, etc). You can only use libraries where these are allowed explicitly in the question.***
- Code for each question should be put in one file named <RollNo>_Q1.py, <RollNo>_Q2.py, …. Your code should have a function test() (or _test()) in which you will have all the test cases you write for the functions in your program. You will zip the files for all the questions and submit the **<RollNo>.zip** file on GC.
- TAs will run plag check on the programs - if plagiarism is detected, institute rules will apply. To avoid this, discuss the problems with each other, but must write the code yourself and not copy. (Using ChatGPT can also get you in trouble, as it is likely to generate similar code for all.)
- TAs may ask you to explain the code you have written (with help of CoPilot). If you cannot explain, it may be treated as copying (or getting the full program generated).
- For grading, TAs will run the python programs and test using some test cases. They will look at the code to ensure that you have followed the instructions (e.g. of writing functions and dividing code, writing suitable test cases).
- Code for each question will be given 0, 1, or 2 marks for incorrect, partially correct, and fully correct answers. The test cases written for each problem will carry 0, 0.5, or 1 mark. (The total marks will be converted to out of 10).
- Solve bonus questions only after you have done all (or at least 80%) of the regular problems correctly. Bonus questions will count only if you have done at least 80% of the regular questions. They together carry only 1 mark.
- The assignment is not yet complete - more questions may be added (it is being released soon, so you can start working on them.)

1. (Simple, ~20 LOC) Integration through computation. Suppose the velocity of a rocket at a time t is given by:

$$f(t) = 2000ln[\frac{140000}{140000 - 2100t}] - 9.8t$$

Take as input starting time (a) and ending time (b), and find the distance covered by a rocket between time a and b. For computationally determining the distance, work with time increments (delta) of 0.25 seconds. You can use the math module of python for this problem.

Hint: Compute the velocity at time t and t+delta, take the average, and then compute the distance traveled in this delta time duration. Start from a and keep computing in increments of delta till b.

2. (Simple, ˜25 LOC). A person is standing and is looking at a pole in front of him. Given the angle of view to the top (in degrees – should be between 0 and 90) and the horizontal distance from the person to the base of the pole (in meters), you have to find the height of the pole and the length of the line from the person to the top of the pole.

Your program has to take as input (from the user on the terminal) the angle (in degrees) and another input for distance to the base of the pole. Then it computes and prints the height of the pole and the distance to the tip of the pole.

Note. For this question, you must write functions to compute sin(), cos(), … using the series for them and ***cannot use*** the python provided functions (e.g., in the math module). If the value of pi is needed, you can use the standard value (3.14). First, write these functions and test them (have tests in tests() for this using asserts). Then write the main program to take inputs for angle and distance, and call the functions to compute the height and distance.

3. (Simple, ˜50 LOC) Write a program to take as input a number between 0 and 99, and print the text equivalent of the number, i.e., for inputs 5, 13, 43, and 85, outputs are "five", "thirteen", "forty three", and "eighty five". (Hint: Have a function to write the text for tens digit, and another to write the text for the unit digit - use of elif construct will be helpful in these. In main, take the input, get the decimal and units digits of the number and print their text. The code is simple though it may be long as you have to print for different cases. You may first write a function to print the text for a units digit and test it, and then build the rest)

Bonus Question. Expand the code above to write in text any number lesser than 100 crore. (FYI we have: units, tens, hundred, thousand, ten-thousand, lac, ten-lac, crore, ten-crore).

4. (Medium, ˜40LOC, 1 function; nested for loop) The current population of the world is combined together into groups that are growing at different rates, and the population of these groups is given (in millions) in a list. The population of the first group (is Africa) is currently growing at a rate **pgrowth** 2.5% per year, and each subsequent group is growing at 0.4% lesser, i.e. the next group is growing at 2.1%. (note: the growth rate can be negative also). For each group, every year, the growth rate reduces by 0.1. With this, eventually, the population of the world will peak out and after that start declining. Write a program that prints: (i) the current total population of the world, (ii) the years after which the maximum population will be reached, and the value of the maximum population.

You must write a function to compute the population of a group after n years, given the initial population and the current rate of growth. Write a few assertions for testing this function (in the test() function)

In the main program, you can loop increasing the years (Y) from 1 onwards, and for each Y, you will need to compute the value of each group after Y years using the function and compute the total population of the world as a sum of these. To check if the population is declining, save the population for the previous Y, and every year check if the population has declined – if it has declined, the previous year's population was the highest (otherwise, this year's population will become the previous year's for next iteration).

The population can be set as an assignment:

pop = [50, 1450, 1400, 1700, 1500, 600, 1200]

In the program, you can loop over this list, but you cannot use any list functions (and you need not index into the list).

Bonus Problem. Change the program to determine the number of years it will take to reach the maximum population for different pgrowth rates of: 2.0%, 2.25%, 2.5%, 2.75%, and 3% (you can keep these values in a list and then loop over it to get values - but cannot use any list operations). Note for this, the computation of years and maximum population will have to be converted into a function (which will return two values).

5. (Medium, ~40LOC) The demand for an item in the market (in terms of the number of items) decreases as the price (p) of the items increases. On the other hand, the supply of the item increases as the price increases (as producers expect to make more profit). Suppose the demand and supply changes as follows with price:

Demand function, D is: ln D(p) = a - b*p (i.e. D(p) is e to the power of rhs)
Supply function, S is: ln S(p) = c + d*p

An equilibrium is achieved when demand and supply match (approximately) - this is a basic economics theory. Determine at what price an equilibrium will be reached, and output the equilibrium price and the number of items produced/bought at this equilibrium.
Given a minimum price, find at what price an equilibrium is found, if it exists. (If no solution is possible - print that.) Iterate by starting with the minimum price and increasing the price by 5% every time (so your equilibrium will not be precise), to find the equilibrium. Your program should print the equilibrium price, and the demand and supply at that price. For this take a, b, c, d as 10, 1.05, 1, 1.06. You can assume the minimum price as 1.0.

Hint: Write a function to compute demand and another to compute supply (for given coefficients and a given price). In main, you start with the initial price, call the function repeatedly with increased price to find where the demand and supply finally cross.

6. (Medium, ~30LOC) Write a program to computationally find a root of a given polynomial in x. Your program should have a function to compute the value of the polynomial for a given value of x (you can assign the coefficients in the first few statements and then compute and return the value). You should have another function that takes as parameters the function to compute the polynomial and an initial value of x (x0), which is used to find the root. For computing the root of the polynomial, use the newton-Raphson method (look it up) - it will start with assuming the initial root as x0, and if the value of the polynomial is not 0 at x0, determine the slope at x0 to find x1 - the point where the straight line with this slope will intersect the x-axis. Then use this x1 and repeat the process. Till the value of the polynomial reaches close to 0 (you can assume that if the polynomial is within +/-0.2, it is a root).

   Your main program should ask for the value of x0 and then compute the root. Try with different values of x0 and see what root it finds – you will see that starting point can lead to different roots. As newton Raphson may not converge, or the polynomial may not have a root, after trying for some number iterations (say 100), your program should print a suitable message and quit.

   For the problem, use the polynomial: x**3 - 10.5*x**2 + 34.5*x - 35 (FYI, this one has 3 different roots)

   Bonus problem: Given that the polynomial has n roots and a range x1 and x2 within which all the roots are, expand the root finding program to find all the roots between x1 and x2.

   For the problem, use the polynomial: x**3 - 10.5*x**2 + 34.5*x - 35 (FYI, this one has 3 different roots)

7. (Medium, ~40 LOC) Visibility Problem. In a 2-D plane, suppose you are sitting at (0,0). The task is to determine how many points (i,j) will be visible to you in a square of size NxN. Visible point: Which you can see, i.e. not hidden by any other point, i.e. while (1,1) is visible, (2,2) or (3,3) are not visible. It is known that a point (a,b) is visible if GCD(a,b) is 1. Write a function to determine the number of visible points for different values of N starting from (0,0). The density of visible points is no of visible points divided by N*N. It is known to converge to 6/pi**2 . Find the value of N for which density comes within some % of this (say 20%).

8. (Simple, ~20 LOC) Solving a differential equation dy/dx = f(x,y). Suppose dy/dx is (x+y), and y(0) is 1. Write a program to computationally determine the value of y for a given x (as float input). Note to compute take step size h=0.1 (or some such number), then compute the value of y from x=0 by incrementing x by h, using the approximation that the differential equation can be viewed as (y2-y1)/((x+h) - x) = f(x,y) .

9. (Simple, ~30 LOC). Random walk. Suppose you are sitting at (0,0) in a 2-D plane. And you randomly walk for a random number of units in x direction or y direction. Your goal is to reach a designated destination (n,m). If at any time you have already reached n or m, then no further walk in that direction is done. Write a program to determine the average number of steps that will be needed to reach a given destination - to be taken as input. For determining the average,

you must do the random walk for as many times as needed, till the change in average is less than 10%. For generating a random number, you can use these:

```
import random # have this at the top of the program
x = random.random()
```

Note that the direction to move (i.e. right or up) is randomly selected, and the distance traveled (an integer) in one walk is a random integer between (0,5).

10. (Long, ~50-60 LOC). Julius Caesar needs help building a version of the Julian Calendar. It is exactly like our modern-day calendar. You need to create a Python program like this (as a sequence of rows and columns).

```
Mon Tue Wed Thu Fri Sat Sun
                      1   2
 3   4   5   6   7   8   9
10  11  12  13  14  15  16
17  18  19  20  21  22  23
24  25  26  27  28  29  30
```

The program should have the following features:

1. **Display the Month**: Print a calendar for any given month in the Julian Calendar. For example, if the user inputs "March 44 BC," the program should display the entire month of March 44 BC.
2. **Navigate Through Months**: Allow the user to type "next" or "previous" to display the next or previous month, respectively. The program should continue running until the user types "exit".

== BONUS PROBLEMS - For bonus marks do at least two (+ Bonus problems given above) =====

These are some bonus problems. Others who are not able to submit it for bonus by the due date can treat these as practice problems.

11. (Medium; ~20 LOC) The Egyptian architects are trying to plan the blueprint of the Pyramids of Giza. They struggle with visualizing how the pyramids will look from different views (top and side). Your task is to write a Python program to help them visualize these views by entering the number of steps in the pyramid.

Given the number of steps, you must initially print the top view and the menu below:

A. Switch view

B. Exit

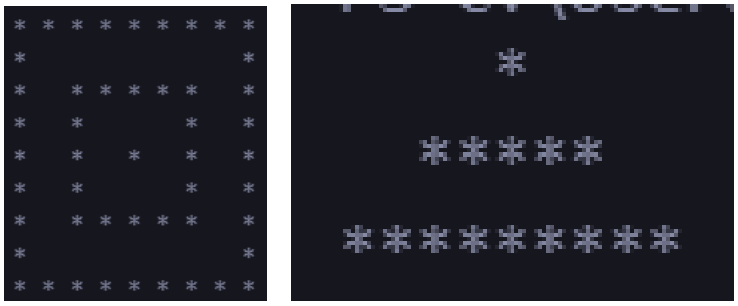
You can Implement the menu above using an infinite loop. If the current view is the top view, then the switch prints the side view. If the current view is a side view, then the switch prints the top view.

The input given to the program will be an integer 1 to 10.

EXAMPLES

Input: 2

Output:



Explanation: There is a top of the pyramid with 1 '*' in and then two steps below.

Input: 3

Output:

Explanation: There is a top of the pyramid with 1 '*' and then three steps below.

12. (Medium, ˜30 LOC) Two variable optimizations. Many problems require optimization of some function – minimizing (e.g., loss) or maximizing (e.g., profit) given some function and some constraints. This is an example of this problem.

A furniture company manufactures dining room tables and chairs. The relevant manufacturing data are given in the table.

Department	Labor-Hours per Table	Labor-hrs per Chair	Maximum Labor-Hours Available/day
Assembly	8	2	400
Finishing	2	1	120
Profit per Unit	$90 for first M units, $100 after that	$25 for first M units, $30 after that	

Write a program to determine how many tables and chairs should be manufactured each day to realize a maximum profit. Print the number of chairs, tables, and the maximum profit. Initially have M = 10. Run the program again for M = 0, 20 and see how the output changes. Keep the code to compute this simple (even if it is inefficient), and you do not have to optimize the code.

Hint: Let x1 represent the number of tables and x2 represent the number of chairs. Determine the equations for total profit P, and the constraints on the two types of labor hours. Write functions to compute P and each constraint (these will be very simple functions). Have a statement in the main program to set M and pass it to function for computing profit - you can change the value of M for different runs of the program. Write the main program that ranges over the values of x1 and x2 (from min possible value to max possible value) to find at which values max is reached.

13. (Medium, ˜50LOC) Find my lucky number. Here are steps to find a lucky number: Given a number n and an integer k, calculate the sum of digits in the odd and even positions from left to right. If the sum of digits in the odd positions is lesser or equal to the sum of the digits in the even position, then it is a lucky number - print it. If the sum of digits in the odd positions is greater than the sum of the digits in the even position (then it is not a lucky number), shift each digit k times to the left (if you reach the first digit position then the digit moves to the end on the right) - this will be a lucky number - print it. (When we shift by an odd number k, the odd digits become even and here we have my lucky number.) **You can not use strings/list or any placeholders.** [Hint : Use modulus % operator.]

Write a test function that generates a number n and k. K must be odd.

Input: An integer n with any number of digits

An integer k indicating the starting position of the rotation

Constraints : 9<n< 10**9 , k is odd and smaller than count of digits |n|.

Output:

Odd_sum, even_sum

The rotated number based on the condition.

Example input : 123456789 3

Example output: 25 20

456789123

Explanation 1) x=123456789, k =3, sum of odd positioned = 1+3+5+7+9=25, sum of even positioned = 2+4+6+8=20

Since 25 > 20, we rotate digits starting from 3rd position to the end and place them at the beginning. The output is 456789123