SFU SIMON FRASER UNIVERSITY

CMPT 120-D300 Assignment 1

Total points: 100, Weighting: 7.5%

Due: Sunday, February 5, 11:59 pm

Assignment 1 includes four programming problems. Read the following instructions before you start working on it:

- Assignments should be submitted to an electronic marking tool called *Gradescope*. If you have not registered in Gradescope yet follow the instructions in this document. You can submit your assignment as many times as you like until the deadline. If necessary, we will review and grade submissions individually to ensure that you receive due credit for your work.
- Follow the requested format for the file name, program inputs, and program outputs. Otherwise, your submission will not be graded. Use Python to write all the programs in this assignment.
- Assignments should be submitted on the due date in order to receive full credit. If you submit your assessment after the due date a penalty of 5% of the total mark allocated for the assignment is deducted per day for the first 3 days after which the submission is not accepted.
- You are encouraged to ask your conceptual questions regarding the assignment on the course *Piazza*.
- You should NOT post any code segment about the assignment problems to Piazza or share your code with other students until February 8, 11:59 PM, the assignment extended deadline. Direct questions regarding your code and solution should be asked privately through Piazza.
- Gradescope is sensitive to user input prompts, as expected. Use the EXACT user input prompts as specified in the assignment questions, otherwise, some test cases may fail.
- In the problem descriptions we use *red single quotes*, ', to show the message that should be printed by your program, excluding ' . For example, if the assignment asks you to display the message 'Enter a positive integer:', you should EXACTLY use the string between red single quotes as argument of the *input* function. For instance, if the assignment problem asked you to interact with the users via a nonsense message '‡ÉªÉ¢Êœ', then, you should simply add the following line to your program `input('‡ÉªÉ¢Êœ\n')` without ANY modification 😎 😄.
- Note that some messages you are asked to display to users end with \n to insert a new line and get user input in that line. You should use it Whenever you are asked to do it.
- Commonalities between students' code will be identified using a Code Similarity detection tool which might be considered plagiarism.
- Course assignments are individual tasks. Please respect Academic Integrity and follow the university's code of conduct.

Question 1: Personality-aware Chatbot [100 marks]

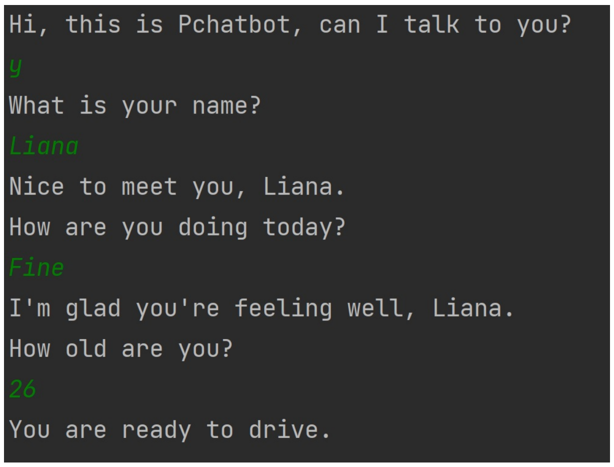
Have you ever wondered how smart are chatbots nowadays in interacting with users? Do you know they use conversational AI to automatically perceive and respond based on what they receive? In this part of the assignment, you are going to implement a personality-aware chatbot, called Pchatbot, which can have limited interaction with the user but is interesting enough to motivate you to think about developing a real personality-aware chatbot in the close future.
Your implemented chatbot should ask the following questions from the user to complete a conversation:

1. 'Hi, this is Pchatbot, can I talk to you?\n'
2. 'What is your name?\n'
3. 'How are you doing today?\n'
4. 'How old are you?\n'

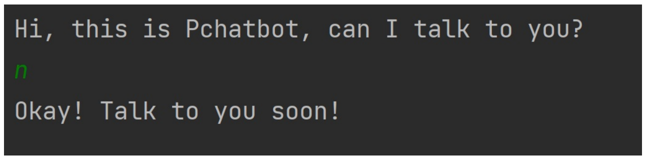
Your program should have the following features:

- For the first question it should be able to accept these inputs from the user: 'Y', 'y', 'N', and 'n'.
  - In the case of **Y** and **y** Pchatbot should continue the conversation.
  - In the case of **N** and **n** Pchatbot should terminate the conversation with this message: 'Okay! Talk to you soon!'.
- For the second question, Pchatbot first should take the user name and print the message: 'Nice to meet you, *Name*.' and *Name* refers to the name that Pchatbot took from the user.
- For the third question, Pchatbot should be prepared to accept two categories of answers from the user and react accordingly:
  - **Positive response:** if the user feels good and the answer is 'Good', 'I'm great', 'I'm good' or 'Fine', then, your chatbot should respond 'I'm glad you're feeling well, *Name*.'. Note that *Name* refers to the user name.
  - **Negative response:** if the user doesn't feel good and the answer is 'Bad', 'Not Okay', or 'I'm not feeling good', then, the chatbot should respond 'Have some time to yourself to recharge!'.
  - **Other response:** Chatbot should say 'I see!' for any other user's answer.
- For the fourth question, if the user is older than 18 years (age > 18) and his response to the third question is *only* in the Positive response category Pchatbot should reply 'You are ready to drive.' Otherwise, Pchatbot should answer 'Still taking the bus!', and this is going to be the end of the user and Pchatbot interaction.

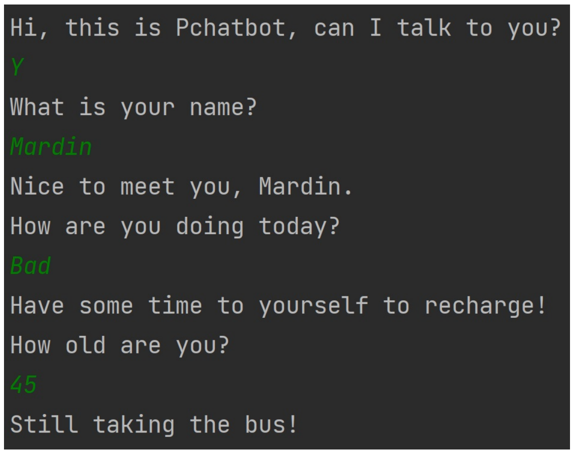
Figure 1 show some sample interactions between Pchatbot and the user. For this question You **must** name your code file **q1.py** and upload it to the *HW1Q1: Personality-aware Chatbot* section on Gradescope.



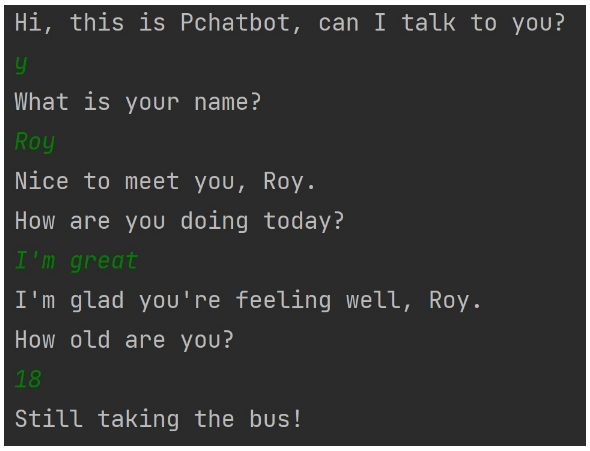
```
Hi, this is Pchatbot, can I talk to you?
y
What is your name?
Liana
Nice to meet you, Liana.
How are you doing today?
Fine
I'm glad you're feeling well, Liana.
How old are you?
26
You are ready to drive.
```



```
Hi, this is Pchatbot, can I talk to you?
n
Okay! Talk to you soon!
```



```
Hi, this is Pchatbot, can I talk to you?
Y
What is your name?
Mardin
Nice to meet you, Mardin.
How are you doing today?
Bad
Have some time to yourself to recharge!
How old are you?
45
Still taking the bus!
```



```
Hi, this is Pchatbot, can I talk to you?
y
What is your name?
Roy
Nice to meet you, Roy.
How are you doing today?
I'm great
I'm glad you're feeling well, Roy.
How old are you?
18
Still taking the bus!
```

Figure 1: Pchatbot sample input and output

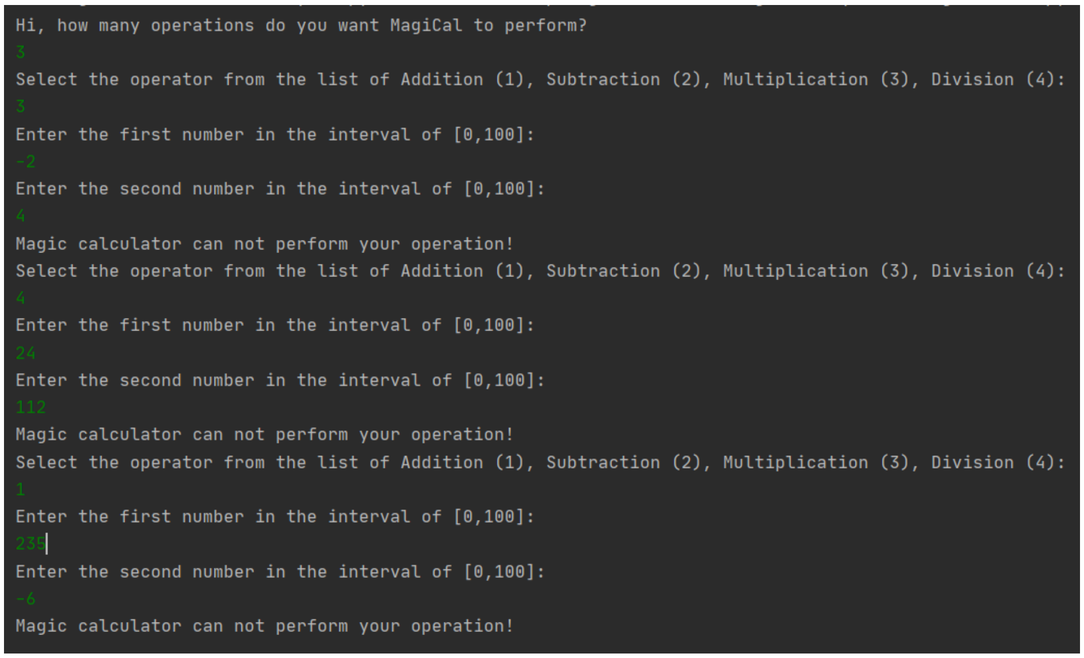
Question 2: Magic Calculator [100 marks]

Have you thought about creating your own calculator? You might, but how difficult is it to create a calculator that includes all functions? Does it need a lot of programming? Well, Windows 10 calculator contains over 35000 lines of code and that should give you an idea. The good news is that in this assignment you will implement a magic calculator, called MagiCal, which familiarize you with the basics of developing such a program. Who knows, you may develop a future version of Windows calculator in a few years :-)!

Magic calculator or MagiCal is not a regular calculator! MagiCal has special features as follows:

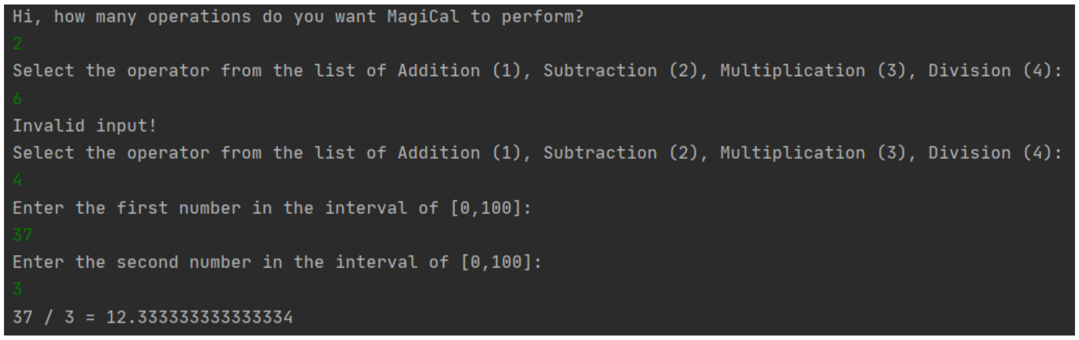
- It only accept four operators: *Addition*, *Subtractoin*, *Multiplication* and *Division*.
- It only accepts *integer* numbers between 0 (inclusive) and 100 (inclusive) as input.
- MagiCal should know how many operations it will perform. It takes this information from the user displaying the message 'Hi, how many operations do you want MagiCal to perform?\n'. Let's assume the user-provided number is k. Now, MagiCal asks the user for k times to provide the operator and two integer numbers as described below:
  - It prints a message 'Select the operator from the list of Addition (1), Subtraction (2), Multiplication (3), Division (4):\n'. Based on the user-provided input there should be two types of responses from MagiCal:
    - If the user enters a valid input, 1, 2 3 or 4, then, the user will be asked to enter two numbers as the operands of the provided operator, as described below.
    - For any invalid output the magic calculator should notify the user with the message 'Invalid input!'.
  - To take the first integer number MagiCal displays the message 'Enter the first number in the interval of [0,100]:\n'.
  - After entering the first number, MagiCal will ask for the second integer number using the message 'Enter the second number in the interval of [0,100]:\n'.
  - If one or both of the provided numbers are not in the specified interval MagiCal will show 'Magic calculator can not perform your operation!', and proceeds for the next operation. If there is not any operation remaining, the program terminates.
  - Finally, if two numbers are in the specified interval MagiCal applies the operator received in the first step and shows the result in the format of 'first_number operator second_number = result'. Note that if the user-provided operator is *division* and the second number is 0, MagiCal should display the message 'The denominator cannot be 0.'.

Figure 2 shows sample interactions between MagiCal and the user. For this question You **must** name your code file **q2.py** and upload it to the *HW1Q2: Magic Calculator* section on Gradescope.



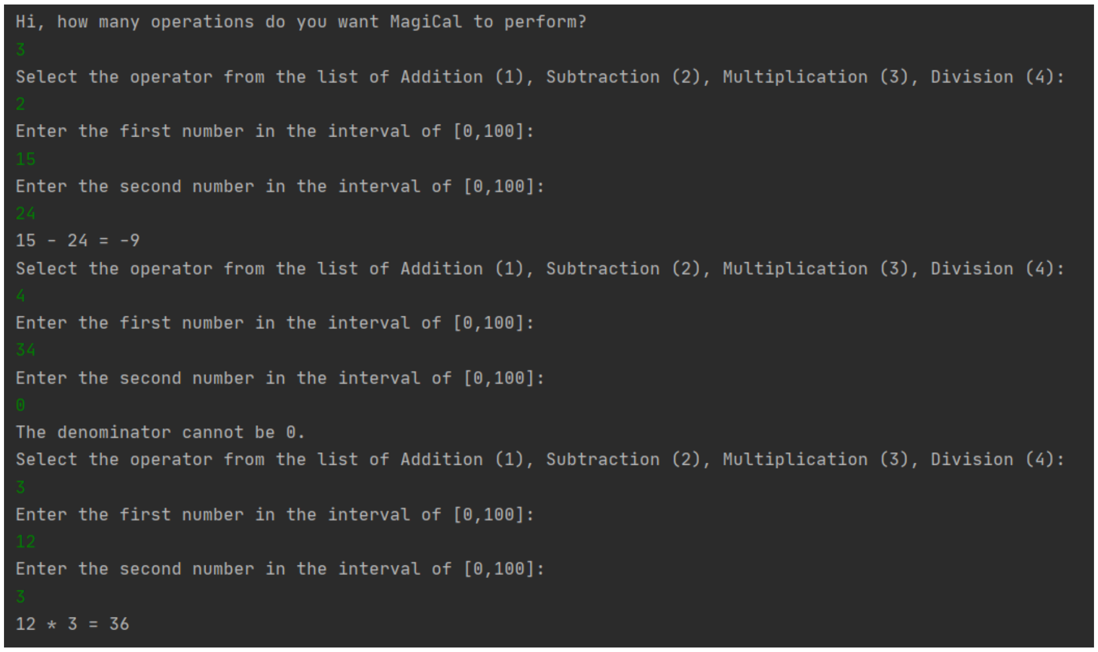
```
Hi, how many operations do you want MagiCal to perform?
3
Select the operator from the list of Addition (1), Subtraction (2), Multiplication (3), Division (4):
3
Enter the first number in the interval of [0,100]:
-2
Enter the second number in the interval of [0,100]:
4
Magic calculator can not perform your operation!
Select the operator from the list of Addition (1), Subtraction (2), Multiplication (3), Division (4):
4
Enter the first number in the interval of [0,100]:
24
Enter the second number in the interval of [0,100]:
112
Magic calculator can not perform your operation!
Select the operator from the list of Addition (1), Subtraction (2), Multiplication (3), Division (4):
1
Enter the first number in the interval of [0,100]:
235
Enter the second number in the interval of [0,100]:
-6
Magic calculator can not perform your operation!
```

a



```
Hi, how many operations do you want MagiCal to perform?
2
Select the operator from the list of Addition (1), Subtraction (2), Multiplication (3), Division (4):
6
Invalid input!
Select the operator from the list of Addition (1), Subtraction (2), Multiplication (3), Division (4):
4
Enter the first number in the interval of [0,100]:
37
Enter the second number in the interval of [0,100]:
3
37 / 3 = 12.333333333333334
```

b



```
Hi, how many operations do you want MagiCal to perform?
3
Select the operator from the list of Addition (1), Subtraction (2), Multiplication (3), Division (4):
2
Enter the first number in the interval of [0,100]:
15
Enter the second number in the interval of [0,100]:
24
15 - 24 = -9
Select the operator from the list of Addition (1), Subtraction (2), Multiplication (3), Division (4):
4
Enter the first number in the interval of [0,100]:
34
Enter the second number in the interval of [0,100]:
0
The denominator cannot be 0.
Select the operator from the list of Addition (1), Subtraction (2), Multiplication (3), Division (4):
3
Enter the first number in the interval of [0,100]:
12
Enter the second number in the interval of [0,100]:
3
12 * 3 = 36
```

Figure 2: MagiCal sample input and output

Question 3: Counting Prime Numbers [100 marks]

A prime number is an integer number greater than 1 that is not a product of two smaller integer numbers. In other words, a prime number has only two positive divisors: one and itself. Examples of prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23, and 29. Do you know during history scientists have always been looking for mysterious properties of prime numbers? Do you know one of the *Millennium Prize Problems* is about the distribution of prime numbers? Well, if this introduction motivated you enough, now, it is time to write a program about prime numbers.

In this part of the assignment, you are asked to create a program, called PrimeFinder, to count the number of prime numbers in a sequence of numbers provided by the user. PrimeFinder should work as follows:

- First, PrimeFinder should ask how many numbers the user has decided to pass to PrimeFinder using the message 'How many numbers do you want to check?\n'. Let's call the provided number by user n.
- For n times PrimeFinder should print 'Enter a positive integer:\n' and take a positive integer from the user.
- For each prime number, PrimeFinder prints out 'X is a prime number.' where X is the number provided by the user in that round.
- For each nonprime number PrimeFinder does not print out a message.
- PrimeFinder assumes that its users are smart enough not to enter a float number.
- If the user inserts a negative number PrimeFinder should ignore it and inform the user by the message 'PrimeFinder ignores negative numbers!'. For instance, if n=*2* the user-provided sequence of numbers can be 3, 5 or 2, -4, -7, 5. In other words, PrimeFinder will continue working until it gets exactly positive integers.
- Finally, PrimeFinder prints out the total count of prime numbers received from the user, Y, in the following format: 'Total prime numbers: Y'.

Figure 3 shows a sample output of PrimeFinder. You **must** name your code file **q3.py** and upload it to the *HW1Q3: PrimeFinder* section on Gradescope.

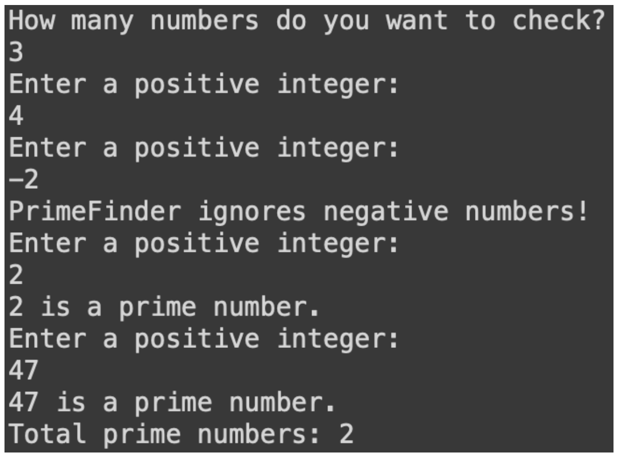

```
How many numbers do you want to check?
3
Enter a positive integer:
4
Enter a positive integer:
-2
PrimeFinder ignores negative numbers!
Enter a positive integer:
2
2 is a prime number.
Enter a positive integer:
47
47 is a prime number.
Total prime numbers: 2
```

Figure 3: PrimeFinder's sample input and output.

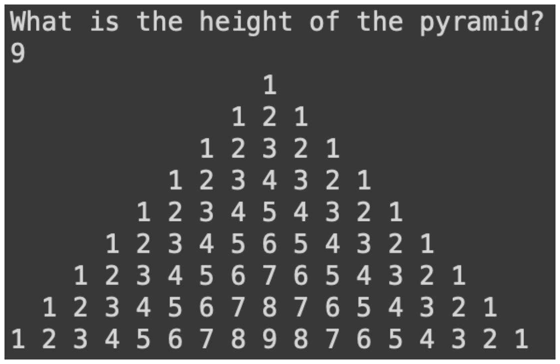
Question 4: Pyramid of Numbers [100 marks]

Why have civilizations in many parts of the world been built as pyramids? What makes pyramids so special? Do you know it is estimated that the Great Pyramid of Giza is built of 2.3 million stone blocks? There is a lot to know about pyramids and this assignment wants to motivate you to do so by writing a program, called PyNum, which draw a pyramid of numbers.

PyNum first ask the user about the pyramid's height using the message 'What is the height of the pyramid?\n'. Let's call the pyramid height h, which should be a positive *integer*. Next, PyNum prints a pyramid of height h with numbers as following:

- PyNum only prints a pyramid of numbers taller than 1 and shorter than 10. In other words, the user-provided input can not be smaller than 2 or greater than 9.
- If the user-provided value is not in the specified range, PyNum will let the user know using the message 'PyNum cannot help you!'
- On the top of the pyramid and in the first line, PyNum will always print 1, as depicted in Figure 4.
- PyNume will print '1 2 3 . . . n n-1 n-2 . . . 1' in the nth line of pyramid. Let's assume $h = 3$, then, the first line of the pyramid will be 1. And second and third lines will be '1 2 1' and '1 2 3 2 1', respectively.
- Note that there is a white space between each pair of consequent numbers.

Figure 4 shows a sample output of PyNum for $h = 9$. You **must** name your code file **q4.py** and upload it to the *HW1Q4: Pyramid of Numbers* section on Gradescope.



```
What is the height of the pyramid?
9
                1
              1 2 1
            1 2 3 2 1
          1 2 3 4 3 2 1
        1 2 3 4 5 4 3 2 1
      1 2 3 4 5 6 5 4 3 2 1
    1 2 3 4 5 6 7 6 5 4 3 2 1
  1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 1
```

Figure 4: PyNum's sample input and output.