NYU Tandon School of Engineering

CS-UY 1114 Spring 2023

Homework 03

Due: 11:59pm, Thursday, February 23rd, 2023

Submission instructions

1. You should submit your homework on **Gradescope**.
2. For this assignment you should turn in 5 separate `.py` files named according to the following pattern: `hw3_q1.py`, `hw3_q2.py`, etc.
3. Each Python file you submit should contain a header comment block as follows:

```
"""
Author: [Your name here]
Assignment / Part: HW3 - Q1 (etc.)
Date due: 2023-02-23, 11:59pm
I pledge that I have completed this assignment without
collaborating with anyone else, in conformance with the
NYU School of Engineering Policies and Procedures on
Academic Misconduct.
"""
```

No late submissions will be accepted.

REMINDER: *Do not use any Python structures that we have not learned in class.*

For this specific assignment, you may use everything we have learned up to, **and including**, variables, types, mathematical and boolean expressions, user IO (i.e. `print()` and `input()`), number systems, and the `math` / `random` modules, selection statements (i.e. `if`, `elif`, `else`), and `for`- and `while`-loops. Please reach out to us if you're at all unsure about any instruction or whether a Python structure is or is not allowed.

Do **not** use, for example, user-defined functions (*except* for `main()` if your instructor has covered it during lecture), string methods, file i/o, exception handling, dictionaries, lists, tuples, and/or object-oriented programming.

Failure to abide by any of these instructions will make your submission subject to point deductions.

Problems

1. **Why, This Car Could Be Systematic, Programmatic, Quadratic! (`hw3_q1.py`)**
2. **(Odd and Even) Baby Steps (`hw3_q2.py`)**
3. **Box Tattooed On Her Arm (`hw3_q3.py`)**
4. **Mod Culture (`hw3_q4.py`)**
5. **2ne1 (`hw3_q5.py`)**

Question 1: *Why, This Car Could Be Systematic, Programmatic, Quadratic!*

Write a program that asks the user to input three floating-point numbers: `a`, `b`, and `c` (just this once, **only these three** single-letter variables will be permitted). These are the parameters of the **quadratic equation**. Classify the equation as one of the following:

- **Infinite number of solutions**: For example, `a = 0`, `b = 0`, `c = 0` has an infinite number of solutions.
- **No solution**: For example, `a = 0`, `b = 0`, `c = 4` has no solution.
- **No real solution**: For example, `a = 1`, `b = 0`, `c = 4` has no real solutions.
- **One real solution**: In cases there is a solutions, please print the solutions.
- **Two real solutions**: In cases there are two real solutions, please print the solutions.

Hint: If a ≠ 0 and there are real solutions to the equation, you can get these solutions using the following formula:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Figure 4: The **quadratic formula**.

The number of solutions depends on whether the discriminant (b^2 - 4ac) is positive, zero, or negative.

For example, an execution *could* look like:

```
Please enter value of a: 1
Please enter value of b: 4
Please enter value of c: 4
This equation has 1 solution: x = -2.0
```

"What is this? A math class?" Yeah, we've heard it a million times before.

Problem 2: *(Odd and Even) Baby Steps*

Write a program that reads a positive integer (say, *n*), and prints the first *n* odd numbers (don't use *n* as a variable name). Write two versions in the file, one using a `for`-loop, and one using a `while`-loop. For example, one execution *could* look like this:

```
Please enter a positive integer: 5
Executing while-loop...
1
3
5
7
```

```
9

Executing for-loop...
1
3
5
7
9
```

Both of these implementations must be included in the same file.

Problem 3: *Box Tattooed On Her Arm*

Write a program that asks the user to input a positive integer, and print a triangle of numbers aligned to the left, where the first line contains the number `1`. The second line contains the numbers `2`, `1`. The third line contains `3`, `2`, `1`. And so on. Surrounding your triangle should be a "frame" composed of the plus `'+'`, minus `'-'`, and pipe `'|'` characters. For example:

```
Please enter a positive integer: 5
+-----+
|1    |
|21   |
|321  |
|4321 |
|54321|
+-----+
```

```
Please enter a positive integer: 7
+-------+
|1      |
|21     |
|321    |
|4321   |
|54321  |
|654321 |
|7654321|
+-------+
```

You may **not** use the `end` parameter of the `print()` function in this problem. Also, note that for input values higher than 9, it will start looking not-so-nice. That's perfectly okay; as long as your output looks good for numbers between 0 and 9, you're all set.

Problem 4: *Mod Culture*

Ask the user to input a positive integer, say, `n`, and print all the numbers from `1` to `n` that have more even digits than odd digits. For example, the number `134` has two odd digits (`1` and `3`) and one even digit (`4`), therefore it should **not** be printed.

Printing should occur **all on one line, separated by spaces**. For example, if `n = 30`, the program should print:

```
2 4 6 8 20 22 24 26 28
```

You may **not** use string casting (i.e. convert integers to strings).

Hint: The title of this problem.

Problem 5: *2ne1*

For this last problem, we will be programming a simple version of the card banking game **twenty-one** (or *vingt-un* in a superior language). According to Wikipedia:

> The game has a **banker** (the person who deals the cards) and a variable number of **punters** (the people who receive cards from the banker).
>
> - The banker deals two cards, face down, to each punter.
> - The punters, having picked up and examined both cards, announce whether they will stay with the cards they have or receive another card from the banker.
> - The aim is to score exactly twenty-one points or to come as close to twenty-one as possible, based on the card values dealt.
>   - If a punter exceeds twenty-one, they lose.
>   - Anyone who achieves twenty-one is also a win.

You will simulate this exact process as if the banker were a random number generator and the only two punters were you and the computer. Your program should:

1. Ask the user whether they want to play twenty-one. They may only enter the characters `'y'` for "yes" and `'n'` for "no". If the player chooses "no", then the game simply ends. Your program must continue asking the player for input *until* the user enters either `'y'` or `'n'`.
2. If the player chooses to play, your program must "give them" two random cards of values [1, 11].
3. The program must then show the player the value of these two cards.
4. The program will then ask the player whether they want a third card or not to put their score closer to 21. Again the program must continue asking the user for input until they enter `'y'` for "yes" or `'n'` for "no".
   - If the user entered `'y'`, generate another random card of value [1, 11]. The user's total score in this case will be the value of all three cards added together.
   - If the user entered `'n'`, the user's total score in this case will be the value of the first two cards added together.
5. Generate the user's opponent's score by generating a random number in the range of [0, 100]. Your program should re-generate another random number **until this number falls between 3 and 33**. Of course we could simply generate number between 0 and 33 instead, but this adds a bit more entropy to the process.
6. Print a message letting the user know who won.
   - **The user automatically wins** if they score a 21 and the computer doesn't.
   - **The computer automatically wins** if they score a 21 and the user doesn't.
   - **The user also wins** if their score is closer to (but not higher than) 21 than the computer's score.
   - **The computer wins** if their score is closer to (but not higher than) 21 than the user's score.
   - **There is a draw/tie** if both players scored the same score and/or if both of their scores are above 21.

Check out the following sample executions. Your wording doesn't need to be the same, but the program must interact with the user in the same basic way:

```
Would you like to play 'Twenty-One'? [y/n] n
Thank you for playing!
```

```
Would you like to play 'Twenty-One'? [y/n] y
Your cards are worth 7 and 10.
Would you like another card? [y/n] n
Your score is 17!
Your opponent's score is 14!
You win! Your score was 17.
```

```
Would you like to play 'Twenty-One'? [y/n] y
Your cards are worth 9 and 11.
Would you like another card? [y/n] yes
Would you like another card? [y/n] y
Your score is 26!
Your opponent's score is 22!
It's a draw!
```

```
Would you like to play 'Twenty-One'? [y/n] y
Your cards are worth 4 and 6.
Would you like another card? [y/n] y
Your score is 15!
Your opponent's score is 8!
You win! Your score was 15.
```

```
Would you like to play 'Twenty-One'? [y/n] y
Your cards are worth 5 and 11.
Would you like another card? [y/n] y
Your score is 23!
Your opponent's score is 9!
Your opponent wins! Their score was 9.
```