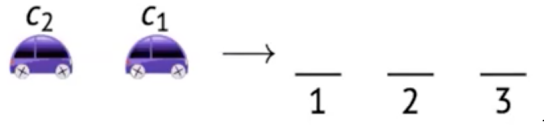

Programming Problem Set 2

Directions: Download the template files I have provided on Blackboard. Then open Spyder, load these template files, and write the following program. Submit your source code via Gradescope in .py format. READ THE INSTRUCTIONS on submitting your work in the Course Documents section of Blackboard.

Be sure to read the SPECIFICATIONS carefully! And write comments!

1) parking.py



A one-way street has three parking spots, numbered 1, 2, and 3. Two cars enter the street, one at a time. Each car has a preferred spot and will park according to the following rules:

- Car 1 always gets to park first. Car 2 will go next.
- If the car's preferred spot is available, it will attempt to park there.
- If the car's preferred spot has been taken, it will attempt to park in the next available spot it sees. (No backing up on the one-way street! For example, if a car's preference is spot 2 and spot 2 is taken, it will attempt to park in spot 3. If a car's first preference is spot 3, and spot 3 has been taken, the car will not park.)

Write a program asking for two cars' preferred spots and give the resulting parking configuration, assuming the vehicles follow the rules above. If one of the cars cannot park, your program should output this information (as specified in the sample run below).

A sample run might look like this (where the user enters the first 3 and the first 1, and everything else is output by the program):

```
Car 1 preferred spot: 3
Car 2 preferred spot: 1
Car 1 parks in spot 3
Car 2 parks in spot 1
```

(since car 1's preferred spot is not taken and then car 2's preferred spot is not taken), or (where the user enters the first and second 3's):

```
Car 1 preferred spot: 3
Car 2 preferred spot: 3
Car 1 parks in spot 3
Car 2 cannot park
```

(since car 1's preferred spot is not taken, car 2 has no *next* available place to park).

Hint: there are nine possible sequences of inputs to this program. One way to do this problem is by writing one case for each sequence – a little tedious, but not too bad.

Note: This problem has six visible test cases on Gradescope. You will see if your code passes these 6 test cases. In addition, it has three invisible test cases, which you will not be able to see until after grades are released (you will not see if your code passed or did not pass these). So make sure you run additional tests on your own to cover all the possible inputs described! (You will need to determine by hand what the correct output should be in your test cases.)

Specifications: your program must

- ask the user to input the preferences of each of the two cars: 1, 2, or 3. Use the input prompt format as specified in the sample runs above.
- print out the parking configuration, given that the cars will park according to the rules specified above. (Use the output format as in the sample runs above.)

¹from Pamela Harris's talk <https://www.youtube.com/watch?v=WRHQRVXIJR8>

- NOT use lists or tuples (or other structures we have not yet discussed)!

Interested in knowing more about what this parking business is about?

The idea for this assignment was inspired by a talk at a recent virtual conference I attended (ask me about it!). Believe it or not, the idea of *parking functions* is an area of active mathematics research. (Things get a lot more complicated when we generalize the problem to have n cars and n spots, and there are other generalizations too!)

I encourage you to read more in the articles linked below or watch the talk from the conference also linked below.

<http://www.girlsangle.org/page/bulletin-archive/GABv14n02E.pdf>

<http://www.girlsangle.org/page/bulletin-archive/GABv14n03E.pdf>

<https://www.youtube.com/watch?v=WRHQRVXljR8>

2) quadratic.py

Write a program that prompts the user to input the three coefficients a , b , and c of a quadratic equation; i.e., an equation of the form $ax^2 + bx + c = 0$. The program should display the solutions of this equation (also known as *zeros*, or *roots* of the function $f(x) = ax^2 + bx + c$) in the following manner:

1. If the equation has one solution, display `ONE SOLUTION:` followed by the solution, displayed with *4 digits printed out after the decimal place*, as shown in the examples below.
2. If the equation has two real solutions, display `TWO REAL SOLUTIONS:` followed by the two solutions, each displayed with *4 digits printed out after the decimal place*, as shown in the examples below. The *smaller* of the two should be displayed first.
3. If the equation has solutions that are not real numbers, display `COMPLEX SOLUTIONS:` followed by two solutions displayed in the form $\alpha + \beta i$, where each α and β should be displayed with *4 digits printed out after the decimal place*, as shown in the examples below. The solution with the "minus" i term should be displayed first in your output (so, for example, $1.0000 - 2.5000i$ should be displayed before $1.0000 + 2.5000i$).

- Also, your displayed solution should NOT have a negative sign in front of a 0.0000 ; for example,

`COMPLEX SOLUTIONS: x = -0.0000 - 1.9634i and x = -0.0000 + 1.9634i`

should be displayed instead as

`COMPLEX SOLUTIONS: x = 0.0000 - 1.9634i and x = 0.0000 + 1.9634i`

Don't bother trying to omit the zeros.

- Also, take care not to display outputs such as

`COMPLEX SOLUTIONS: x = 0.1493 - -1.0112i and x = 0.1493 + -1.0112i`

with the "-" and "+" before the imaginary term. Instead, it should be displayed as

`COMPLEX SOLUTIONS: x = 0.1493 - 1.0112i and x = 0.1493 + 1.0112i`

4. If 0 is entered as input for a , do no calculations and output the message `The equation is not quadratic`

For example, a run might look like this:

```
Enter x^2 coefficient: 0
Enter x^1 coefficient: -2
Enter x^0 coefficient: 1
The equation is not quadratic
```

OR

```
Enter x^2 coefficient: 1
Enter x^1 coefficient: -2
Enter x^0 coefficient: 1
ONE SOLUTION: x = 1.0000
```

OR

```
Enter x^2 coefficient: 3
Enter x^1 coefficient: 5
Enter x^0 coefficient: 1
TWO REAL SOLUTIONS: x = -1.4343 and x = -0.2324
```

OR

```
Enter x^2 coefficient: -1
Enter x^1 coefficient: 2
Enter x^0 coefficient: 7
TWO REAL SOLUTIONS: x = -1.8284 and x = 3.8284
```

OR

```
Enter x^2 coefficient: 2.1
Enter x^1 coefficient: 4
Enter x^0 coefficient: 10
COMPLEX SOLUTIONS: x = -0.9524 - 1.9634i and x = -0.9524 + 1.9634i
```

In the last case, note that the letter that is printed out to represent the square root of -1 is the letter `i`, **not the letter `j`**! Python has some functions that produce complex values, but when you print these values, they display the letter `j` to represent $\sqrt{-1}$. I don't want that – instead, your program should calculate the imaginary coefficient and include code that prints out the string `"i"`. **Do NOT use the `cmath` module, the `complex` function, the `.imag` attribute or any Python functions that perform replacements in strings.** (If you don't know what any of those are, don't worry about it.)

Note: This problem has six visible test cases on Gradescope. You will see if your code passes these six test cases. In addition, it has three invisible test cases, which you will not be able to see until after grades are released (you will not see if your code passed or did not pass these). So make sure you run additional tests on your own to cover all the possible inputs described! (You must work out the correct output in your test cases.)

Specifications: your program must

- ask for and accept coefficients from the user. Use the input prompt format as specified in the sample runs above.
- display the *type* of solutions as above. If there is exactly one solution, it is acceptable if the program occasionally misidentifies it (due to float weirdness, as discussed in class). However, it should work with the example test cases.
- display all solutions with precisely four digits printed after each decimal (for complex solutions, four digits after the decimal for both real and imaginary parts).
- use the letter `i` to represent $\sqrt{-1}$ in output, not the letter `j`.
- NOT use the `cmath` module, the `complex` function, the `.imag` attribute, or any Python functions that perform replacements in strings.
- Remember that to pass the test cases on Gradescope, formatting, including spacing, must match exactly as in the sample runs above.