CMSC 201 Section 60
Homework 5
Fully Faithful Functions

Due Date: Monday, October 23, , 2023 by 11:59:59 PM
Value: 40 points

This assignment falls under the standard cmsc201 academic integrity policy. This means you should not discuss/show/copy/distribute your solutions, your code or your main ideas for the solutions to any other student. Also, you should not post these problems on any forum, internet solutions website, etc.

Make sure that you have a complete file header comment at the top of each file, and that all of the information is correctly filled out.

```
"""
File:    FILENAME.py
Author:  YOUR NAME
Date:    THE DATE
Section: YOUR DISCUSSION SECTION NUMBER
E-mail:  YOUR_EMAIL@umbc.edu
Description:
  DESCRIPTION OF WHAT THE PROGRAM DOES
"""
```

Creating Your HW5 Directory

```
linux3[1]% cd cmsc201/Homeworks
linux3[2]% mkdir hw5
linux3[3]% cd hw5
linux3[4]% █
```

Submission Details

Submit the files under the following titles:
(These are case sensitive as usual.)
submit cmsc201 HW5 {files go here}

Problem 1 - Lock and Key	lock_and_key.py
Problem 2 - Pascal's Triangle	pascal.py
Problem 3 - Checkerboard	checkerboard.py
Problem 4 - Minor Key	minor_key.py
Problem 5 - Quasi-Palindrome	quasi_palindrome.py

For example you would do:

```
linux1[4]% submit cmsc201 HW5 lock_and_key.py pascal.py
checkerboard.py minor_key.py quasi_palindrome.py
Submitting lock_and_key.py...OK
Submitting pascal.py...OK
Submitting checkerboard.py...OK
Submitting minor_key.py...OK
Submitting quasi_palindrome.py...OK
linux1[5]% █
```

Coding Standards

NOTE: YOU WILL LOSE POINTS FOR NOT FOLLOWING CODING STANDARDS STARTING WITH THIS HOMEWORK!!!

Coding standards can be found here.

We will be looking for:

1. At least one inline comment per program explaining something about your code.
2. Constants above your function definitions, outside of the "if __name__ == '__main__':" block.
   a. A magic value is a string which is outside of a print or input statement, but is used to check a variable, so for instance:
      i. `print(first_creature_name, 'has died in the fight. ')` does not involve magic values.
      ii. However, `if my_string == 'EXIT':` exit is a magic value since it's being used to compare against variables within your code, so it should be:

```
EXIT_STRING = 'EXIT'
...
if my_string == EXIT_STRING:
```

   b. A number is a magic value when it is not 0, 1, and if it is not 2 being used to test parity (even/odd).
   c. A number is magic if it is a position in an array, like my_array[23], where we know that at the 23rd position, there is some special data. Instead it should be
      USERNAME_INDEX = 23
      my_array[USERNAME_INDEX]
   d. Constants in mathematical formulas can either be made into official constants or kept in a formula.
3. Previously checked coding standards involving:

a. snake_case_variable_names
b. CAPITAL_SNAKE_CASE_CONSTANT_NAMES
c. Use of whitespace (2 before and after a function, 1 for readability.)

Allowed Built-ins/Methods/etc

- Declaring and assigning variables, ints, floats, bools, strings, lists, dicts.
- Using +, -, *, /, //, %, **; +=, -=, *=, /=, //=, %=, **= where appropriate
- Comparisons ==, <=, >=, >, <, !=, in
- Logical and, or, not
- if/elif/else, nested if statements
- Casting int(x), str(x), float(x), (technically bool(x))
- For loops, both *for i* and *for each* type.
- While loops
  - sentinel values, boolean flags to terminate while loops
- Lists, list(), indexing, i.e. my_list[i] or my_list[3]
  - 2d-lists if you want them/need them my_2d[i][j]
  - Append, remove
  - **list slicing**
- String operations, concatenation +, +=, split(), strip(), join(), upper(), lower(), isupper(), islower()
  - **string slicing**
- Print, with string formatting, with end= or sep=:
  - '{}'.format(var), '%d' % some_int, f-strings
  - Really the point is that we don't care how you format strings in Python
  - Ord, chr, but you won't need them this time.
- Input, again with string formatting in the prompt, casting the returned value.

- Using the functions provided to you in the starter code.
- Using import with libraries and specific functions **as allowed** by the project/homework.

Forbidden Built-ins/Methods/etc

This is not a complete listing, but it includes:

- break, continue
- **Dictionaries**
    - creation using dict(), or {}, copying using dict(other_dict)
    - .get(value, not_found_value) method
    - accessing, inserting elements, removing elements.
    - This won't be forbidden much longer
- **Creating your own Classes**
    - Neither will this, be patient.
- methods outside those permitted within allowed types
    - for instance str.endswith
    - list.index, list.count, etc.
- Keywords you definitely don't need: await, as, assert, async, class, except, finally, global, lambda, nonlocal, raise, try, yield
- The *is* keyword is forbidden, not because it's necessarily bad, but because it doesn't behave as you might expect (it's not the same as ==).
- the following built in functions/keywords: any, all, breakpoint, callable, classmethod, compile, exec, delattr, divmod, enumerate, filter, map, max, min, isinstance, issubclass, iter, locals, oct, next, memoryview, property, repr, reversed, round, set, setattr, slice, sorted, staticmethod, sum, super, type, vars, zip
- exit() or quit()
- If something is not on the allowed list, not on this list, then it is probably forbidden.
- The forbidden list can always be overridden by a particular problem, so if a problem allows something on this list, then it is allowed for that problem.

Problem 1 - Lock and Key

Create a file called lock_and_key.py.

Inside this file create a function with the following definition (there are no spaces in the variable/function names– those are underscores):

```
def lock_and_key(key_cuts, lock_pinning, minimum):
```

When you have a key and a lock, the key cuts and the lock pinning must add together to 6. However, some keys and some locks are a bit imprecise and may not be machined perfectly. Therefore we will allow a minimum difference between the perfect value of 6 and the actual values.

For instance, if one of the key pins is a 2.1 and the key itself is a 4.1, and our minimum difference is 0.25 then this will be allowed because 2.1 + 4.1 < 6 + minimum = 6.25.

Similarly, if you have a pin which is 3.9 and a key cut which is 1.9 then with the same minimum of 0.25 we see that 5.75 < 3.9 + 1.9 = 5.8 < 6.25, so that is also in the proper range.

Basically the equation to match should be:

$$6 - m \leq c + p \leq 6 + m$$

or alternatively:

$$|c + p - 6| < m$$

This is where c is the key cut, p is the pin size, and m is the minimum. You will be passed two lists of cuts and pins, and then the minimum. Ensure that they key has the same number of cuts as the lock has pins, and then test to see whether that key opens the lock. If you wish you can use the absolute value function abs for this purpose.

Sample Output

Given this code that runs the function:

```
if lock_and_key([2.1, 3.5, 2.7], [4.1, 2.5, 3.2],
0.25):
   print('Unlocked')
else:
   print('Still Locked')

if lock_and_key([2.1, 3.5, 2.7, 1.7], [4.1, 2.5,
3.2], 0.25):
   print('Unlocked')
else:
   print('Still Locked')

if lock_and_key([2.1, 3.5, 2.7, 1.7], [4.1, 2.5, 3.2,
3.2], 0.25):
   print('Unlocked')
else:
   print('Still Locked')
```

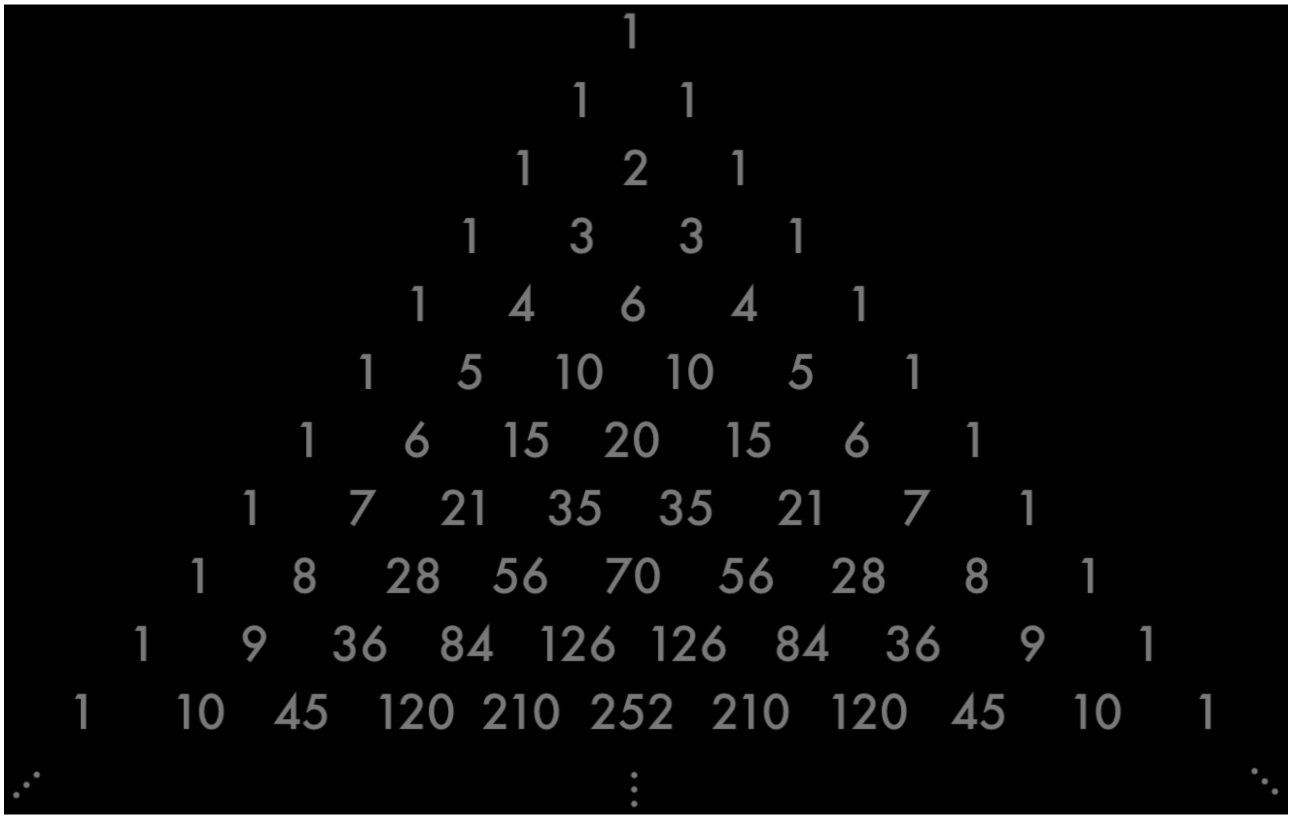
```
linux5[201]% python3 lock_and_key.py
Unlocked
Still Locked
Still Locked
```

Problem 2 - Pascals Triangle

For this problem you're going to take a list, and compute the next level of Pascal's triangle based on the previous level (if the previous list isn't in Pascal's triangle, do it anyway).

Create a function `def next_level(level):` where level is a list of ints containing a level of the triangle.

To compute the next level in Pascal's triangle, it's important to know what Pascal's Triangle is.



```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5  10  10   5   1
        1   6  15  20  15   6   1
      1   7  21  35  35  21   7   1
    1   8  28  56  70  56  28   8   1
  1   9  36  84 126 126  84  36   9   1
1  10  45 120 210 252 210 120  45  10   1
⋰                   ⋮                   ⋱
```

Note here that if you look at any entry, and then at the two entries at a diagonal above it, you'll see that the entry is always equal to the sum of the two diagonal entries above it.

So for instance given the list [1, 4, 6, 4, 1] your objective is to return the list [1, 5, 10, 10, 5, 1].

At the end, make sure you remember to add whatever that last element is to the list again.

When the code here is run:

```
level = [1]
for i in range(10):
    for x in level:
        print(x, end='\t')
    print()
    level = next_level(level)
```

Here is the output.

```
1
1	1
1	2	1
1	3	3	1
1	4	6	4	1
1	5	10	10	5	1
1	6	15	20	15	6	1
1	7	21	35	35	21	7	1
1	8	28	56	70	56	28	8	1
1	9	36	84	126	126	84	36	9	1
```

Alternatively, you can run code like this:

```
in_string = input('What values do you want to run
next_level on? ')
while in_string != '':
    values = []
    for x in in_string.split():
        values.append(int(x))
    print(next_level(values))
    in_string = input('What values do you want to run
next_level on? ')
```

Sample Output

```
linux5[109]% python3 pascal.py
What values do you want to run next_level on? 1 2 3
[1, 3, 5, 1]
What values do you want to run next_level on? 1 4 1
[1, 5, 5, 1]
What values do you want to run next_level on? 1 9 8 4 2 1
[1, 10, 17, 12, 6, 3, 1]
What values do you want to run next_level on? 1 3 3 1
[1, 4, 6, 4, 1]
What values do you want to run next_level on? 1 4 6 4 1
[1, 5, 10, 10, 5, 1]
What values do you want to run next_level on? 1 -1 1
[1, 0, 0, 1]
What values do you want to run next_level on? 1 0 0 1
[1, 1, 0, 1, 1]
What values do you want to run next_level on? 1 1 0 1 1
[1, 2, 1, 1, 2, 1]
```

Problem 3 - Checkerboard

Inside of a file called checkerboard.py create a function

```
def checkerboard(size, symbol_1, symbol_2):
```

Your goal will be to draw a checkerboard pattern where you alternate symbols that are passed as arguments. For instance if the size is 5 and the symbols are 'a' and 'b':

```
ababa
babab
ababa
babab
ababa
```

Notice that each line starts with the opposite symbol and then alternates.

In order to calculate which symbol you need, think about modulus division.

Sample Output

You should write some code so that you can input the lines, generally.
Remember blue is user input.

```
linux5[109]% python3 checkerboard.py
What size do you want? 3
What symbols do you want? : ~
:~:
~:~
:~:

linux5[110]% python3 checkerboard.py
What size do you want? 7
What symbols do you want? E H
EHEHEHE
HEHEHEH
EHEHEHE
HEHEHEH
EHEHEHE
HEHEHEH
EHEHEHE

linux5[110]% python3 checkerboard.py
What size do you want? 10
What symbols do you want? . /
./././././
/./././././
./././././
/./././././
./././././
/./././././
./././././
/./././././
./././././
/./././././
```

Problem 4 - Minor Key

Here is a list of musical notes, written with flats (rather than sharps):
C, D♭, D, E♭, E, F, G♭, G, A♭, A, B♭, B, C (in the next octave)

Observe that not all notes have flats, i.e. C flat is actually B which is not a black key on the piano for instance.

The harmonic minor scale is composed of the following sequence of steps:

[1 Step] [½ Step] [1 Step] [1 Step] [½ Step] [**1½ Step**] [½ Step]

For instance, the C harmonic minor scale is C, D, E♭, F, G, A♭, B, C.

Each musical "half step" is a single step through the list, the name half-step predates computers so you have to forgive musicians a little.

The E♭ harmonic minor scale is E♭ F G♭ A♭ B♭ B D E♭
The G harmonic minor scale is G A B♭ C D E♭ G♭ G

Define a list:

```
MUSICAL_NOTES = ['C', 'D\u266d', 'D', 'E\u266d',
'E', 'F', 'G\u266d', 'G', 'A\u266d', 'A', 'B\u266d',
'B']
```

Note that '\u266d' is the unicode for the flat symbol.

Allow the user to enter starting notes like: "E" or "B flat" or "G flat"

You will have to calculate the starting note from this, or if it's not a note, for instance if they enter "R"

Then display the harmonic minor scale starting at that note, or an error message indicating that it is not a note.

When the user enters "quit" then stop asking for notes and exit the program.

My suggestion is that you break this problem down into parts:

1) Translate the input into the musical notation by creating a 'helper function' which takes as a parameter the user's input and outputs the proper note, meaning it will replace 'flat' with ♭ and remove the space, so that it matches the description in the list of strings.
2) Compute the starting index in the list of musical notes or determine that the starting index doesn't exist because the note isn't a real note. I use another helper function for this purpose.
3) Calculate the scale itself in a slightly bigger function but then return it as a list.
4) Outside of your function, implement a main loop where you will join the answer together and output it.

This is generally how programmers think about problems. We attempt to break each problem into subproblems which we can write small functions to solve.

Then we put those functions together and get our final results.

Sample Output

```
linux5[109]% python3 minor_key.py
Enter a starting note (C, D flat): G
G A B♭ C D E♭ G♭ G
Enter a starting note (C, D flat): A
A B C D E F A♭ A
Enter a starting note (C, D flat): A flat
A♭ B♭ B D♭ E♭ E G A♭
Enter a starting note (C, D flat): R flat
There is no starting note R♭
Enter a starting note (C, D flat): quit

linux5[110]% python3 minor_key.py
Enter a starting note (C, D flat): C
C D E♭ F G A♭ B C
Enter a starting note (C, D flat): D
D E F G A B♭ D♭ D
Enter a starting note (C, D flat): E flat
E♭ F G♭ A♭ B♭ B D E♭
Enter a starting note (C, D flat): G
G A B♭ C D E♭ G♭ G
Enter a starting note (C, D flat): D flat
D♭ E♭ E G♭ A♭ A C D♭
Enter a starting note (C, D flat): quit
```

Problem 5 - Quasi-Palindrome

A palindrome is a word spelled the same forwards and backwards, for instance "racecar" or "tacocat" which of course is a real word.

Most sequences of letters like abcdefg are not palindromes. Let's introduce a nearby concept where we talk about a quasi-palindrome.

A quasi-palindrome of type 1 is a word where it would be a palindrome if we could change one letter for instance: abcdba would be the palindrome abccba or abddba if we could change either one of those letters. However it is not a palindrome itself.

A quasi-palindrome of type 2 is a word where it is 2 away from being a palindrome. For instance abczxa is 2 away from being a palindrome.

Write a function called:

```
def quasi_palindrome(word, errors):
```

Where word is a string and errors is the maximum number of errors allowed. Determine if a word is a quasi-palindrome (of type -N) where N is the number of errors allowed. Your function should return True or False.

Sample Output

```
linux5[281]% python3 quasi_palindrome.py
What word do you want to check? tacocat
How many errors do you want to allow? 0
It was a 0-quasi-palindrome!
What word do you want to check? abcdba
How many errors do you want to allow? 1
It was a 1-quasi-palindrome!
What word do you want to check? abcdeferrba
How many errors do you want to allow? 1
It was not a 1-quasi-palindrome!
What word do you want to check? abcdeferrba
How many errors do you want to allow? 2
It was a 2-quasi-palindrome!
What word do you want to check? alba
How many errors do you want to allow? 1
It was a 1-quasi-palindrome!
What word do you want to check? this can't be a
palindrome
How many errors do you want to allow? 100
It was a 100-quasi-palindrome!
What word do you want to check? quit
```

Note about Data Types

Whenever you have an input statement and try to cast it, if you expect an integer, we'll make sure we always give an integer. Even if you request a list of integers, we'll give a list of integers. On the other hand, we don't guarantee that the integers are in the valid ranges, i.e. if you ask for a number between 1 and 5, we are allowed to give you -17.

Similarly for floating points, we can give you any floating point that we want, but it won't result in a ValueError. Since try-except is still not permitted, we can't require you to validate types in the way that Python intends.

Basically this means:

1) **Don't** worry about validating type.
2) **Do** worry about validating ranges.