Homework 5
Fully Faithful Functions

Due Date: Monday, October 21, 2024 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 - Fast Food Order	fast_food.py
Problem 2 - Pascal's Triangle	pascal.py
Problem 3 - Connect Four	connect_four.py
Problem 4 - Twitter Ats	twitter_ats.py
Problem 5 - Quasi-Palindrome	quasi_palindrome.py

For example you would do:

```
linux1[4]% submit cmsc201 HW5 fast_food.py pascal.py
connect_four.py twitter_ats.py quasi_palindrome.py
Submitting fast_food.py...OK
Submitting pascal.py...OK
Submitting connect_four.py...OK
Submitting twitter_ats.py...OK
Submitting quasi_palindrome.py...OK
linux1[5]%
```

Coding Standards

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 - Fast Food Order

Create a file called fast_food.py.

Inside this file create a function called:

```
def fast_food_receipt(order):
```

You may expect that order is a list containing strings.

The function should take a list as a parameter and add the following:

type of item	how to tell which thing it is	price
burger	'burger' or 'sandwich' in the string	$5.00
fries	'fries' in the string	$3.00
drinks	'coke', 'sprite', or 'mountain dew'	$2.50
combo	a burger/sandwich, fries, and drink	$8.50
anything else	anything else	$4.00

When you order a burger, fries, and a drink then that counts as a combo rather than the individual elements and should only cost $8.50.

Write a function that **returns** the total cost of the fast food order. Do not print the value inside your function, make sure you return it.

If you test your function, do not input anything inside the function itself, do that in your if __name__ == '__main__' block. Your function should only return the price.

Sample Output

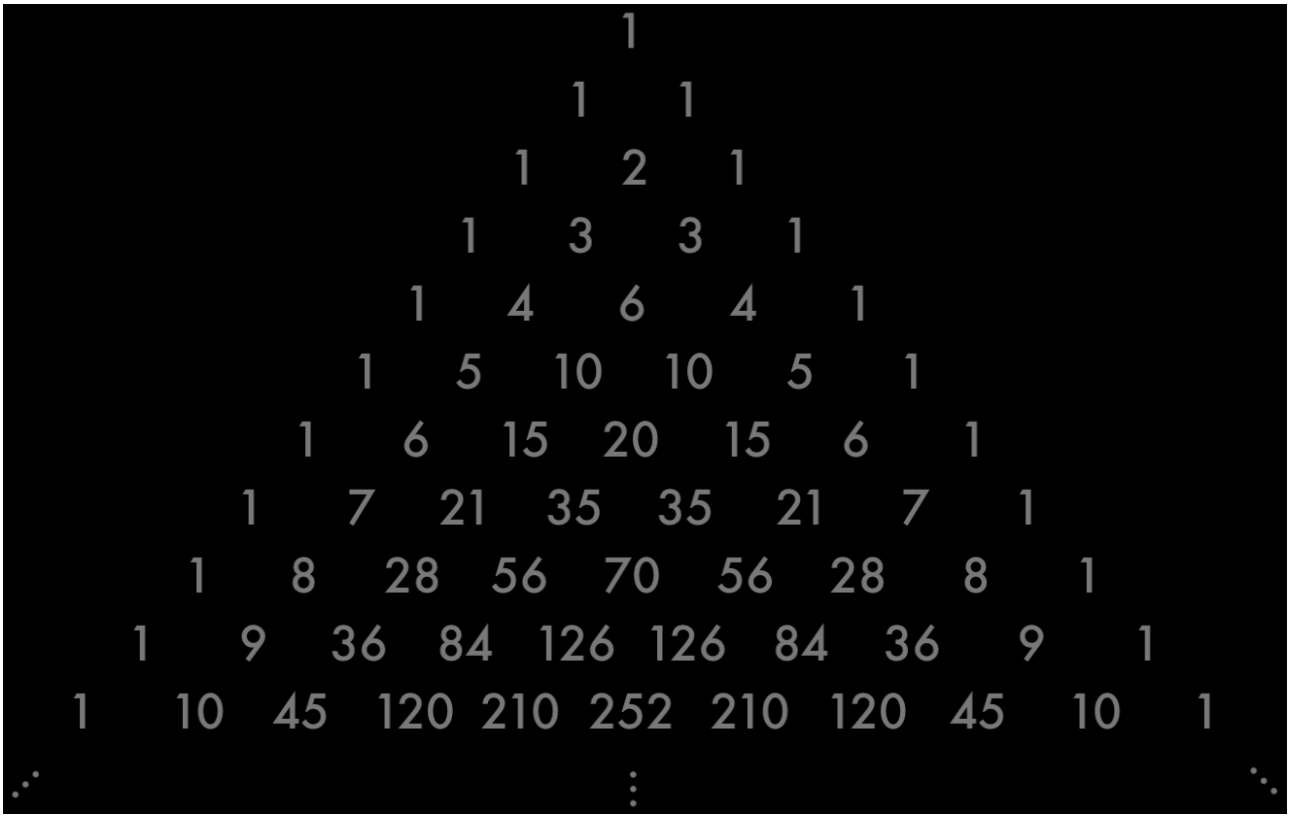
```
linux5[201]% python3 fast_food.py
What would you like to order? pizza
What would you like to order? cheeseburger
What would you like to order? fries
What would you like to order? coke
What would you like to order? apple pie
What would you like to order? hamburger
What would you like to order? double hamburger
What would you like to order? mountain dew
What would you like to order? fries
What would you like to order? fries
What would you like to order? fries
What would you like to order? hamburger
What would you like to order? place order
The total bill is 41.0

linux5[203]% python3 fast_food.py
What would you like to order? pizza
What would you like to order? pasta
What would you like to order? coke
What would you like to order? place order
The total bill is 10.5

linux5[203]% python3 fast_food.py
What would you like to order? burger
What would you like to order? burger
What would you like to order? fries
What would you like to order? coke
What would you like to order? mountain dew
What would you like to order? place order
The total bill is 16.0
```

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.



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 - Connect Four

Create a function `def connect_four(the_grid, player_symbol):`

inside of a file called connect_four.py.

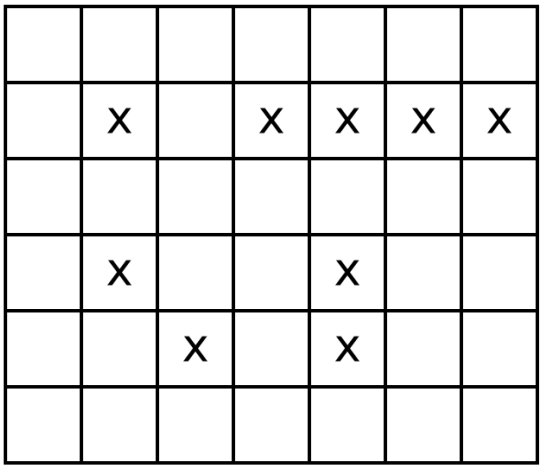
The variable the_grid will be a 2d-list of individual characters, and the player_symbol will be a single character.

In this version of connect four, we'll **only** consider vertical and horizontal matches, **not diagonal.** This means that we are looking for four of the player_symbol in a straight line up and down or left to right.

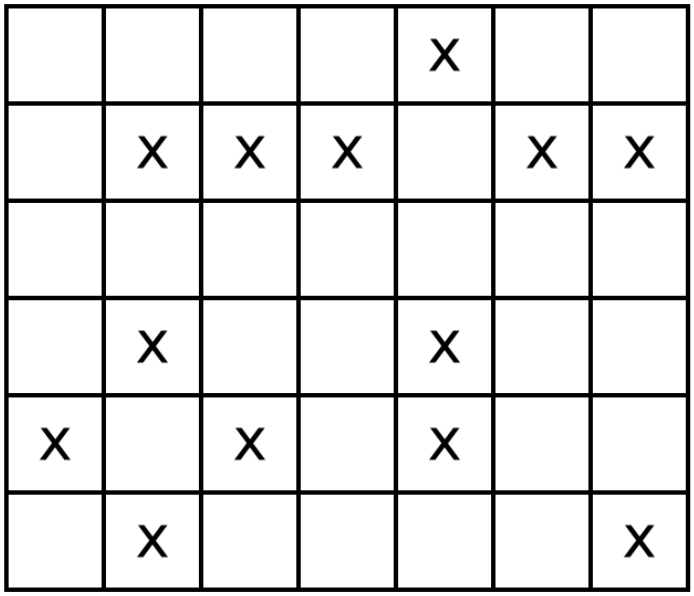
For instance if you receive a grid like this, you should return True because there are four x's in a vertical line. Assume the rest of the characters are not 'x'.

	X			X		
				X		
	X			X		
		X		X		

Here is an example of a horizontal line victory for the game.



Here is a grid made up of some x's but also have no victory, so you would return False.



Sample Output

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

```
linux5[109]% python3 connect_four.py
How many rows will be entered? 5
ooxxoo
xooxoo
oooxoo
oooxoo
ooooox
True

linux5[110]% python3 connect_four.py
How many rows will be entered? 6
ooooxxxo
ooxxoooo
oooooxxo
ooxxoooo
ooooxxxx
oooooooo
True

linux5[110]% python3 connect_four.py
How many rows will be entered? 4
xoox
oxox
oooo
xoox
False
```

Problem 4 - Twitter Ats

The function should have the signature:

```
def twitter_ats(the_tweet):
```

There are two types of things that we want to extract when it comes to tweets, at's denoted by @ and hashtags denoted by #.

For instance if you were to tweet "@ProfessorHamilton 201 is really great #Python #hash_tags_are_weird" then you should return a list like this:

[['ProfessorHamilton'], ['Python', 'hash_tags_are_weird']]

Only return a list of unique usernames, so for instance if someone were to "at" the same person twice, don't add the name twice. Strip off the # tags and @ symbols. If someone tweets with a # symbol without any thing after it, or @ with no username, then ignore it.

For instance "@ @ @ @ # # # #" should produce [[],[]], two empty lists.

The tweet variable itself will be a string. Make sure you return a list of lists.

Sample Output

Blue text indicates user input. Black text is output by the program or console prompt.

```
linux5[109]% python3 twitter_ats.py
What do you want to tweet? Have you ever found that
you really like cheese? @CheeseMan #CheeseDay
The users were: CheeseMan
The hash-tags were: CheeseDay
What do you want to tweet? #Hash #Tags #Are #Cool
The users were:
The hash-tags are: Hash, Tags, Are, Cool
What do you want to tweet? @Eric @Sarah @Jimmy
@Theresa
The users were: Eric, Sarah, Jimmy, Theresa
The hash-tags are:
What do you want to tweet? @Sammy What do you think
about the tweets? #BirdWatching
The users were: Sammy
The hash-tags are: BirdWatching
What do you want to tweet? 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.