15-110 Hw2 - Written Portion

Name:

AndrewID:

#1 - Debugging with Variable Tables - 5pts

The following program is supposed to count the number of unique prime factors between 2 and 7 that a given number `x` has. For example, `countPrimeFactors(7)` should return 1 [for 7], `countPrimeFactors(4)` should return 1 [for 2], and `countPrimeFactors(15)` should return 2 [for 3 and 5]. Unfortunately, the program has a bug and it isn't outputting the correct results.

Trace the code and fill out the debugging table on the next page for inputs 5, 9, and 28. In each row, enter the current value of `count` at the time when the program reaches the if clause (not if body) associated with the letter. Then select the most likely cause of the bug from the choices below.

```
def countPrimeFactors(x):
    count = 0
    # A
    if x % 2 == 0:
        count = count + 1
    # B
    elif x % 3 == 0:
        count = count + 1
    # C
    elif x % 5 == 0:
        count = count + 1
    # D
    elif x % 7 == 0:
        count = count + 1
    # E
    return count
```

Value of count at...	Input = 5	Input = 9	Input = 28
# A	0	0	0
# B			
# C			
# D			
# E			

What do you think caused the bug?

- ☐ The mod operation causes an error
- ☐ The `count` variable is added too many times
- ☐ The `count` variable is only added to once
- ☐ The missing `else` statement causes an error

#2 - Expression to Truth Table and Circuit - 15pts

Given the Boolean expression shown below, fill out the truth table to perform the same operation as the expression. You may not need to use all the given rows. Then create a circuit that performs the same operation as the expression.

`(x and y) or (not (y xor z))`

x value	y value	z value	output value

For the circuit, you may use logic.ly, a different circuit simulator tool, or you may draw the circuit by hand. You can click on the box on the next page to upload an image into it; if that does not work, use a PDF editing tool (like Preview or smallpdf.com/edit-pdf) to insert the image manually. Make sure to delete the blank page if you do this.

If you have trouble getting your image into the PDF, contact the course staff for help.

Click here to add your image

#3 - Loop Control Variable Values - 9pts

Each of the following problem prompts could be implemented using a while loop. Identify the start value, continuing condition, and update action for the loop control variable you would use in that while loop. Assume that the loop control variable will be outputted at the beginning of the loop, and no conditional will be used. We've given an example of what this looks like in the first line

Ex) Output the numbers from 1 to 10, inclusive.
A) Output all even numbers between 2 and 20, inclusive of 2 and exclusive of 20.
B) Output the numbers from 10 to 1, inclusive.
C) Output the numbers 3, 9, 15, 21.

Prompt	Start Value	Continuing Condition	Update Action
Ex	1	x <= 10	x = x + 1
A			
B			
C			

#4 - Code Tracing with For Loops - 6pts

For each of the following range expressions, list all the values the loop variable will be set to over the course of the range. For example, `range(1, 5)` produces `1`, `2`, `3`, `4`.

Range Expression	Numbers Produced
`range(3)`	
`range(4, 8)`	
`range(1, 10, 3)`	

#5 - Code Tracing with Strings - 10pts

Assuming that the following two lines of code have been run:

```
s1 = "15-110 is cool"
s2 = "CMU rocks!"
```

What will each of the following expressions evaluate to? Don't just run the code in the editor- try to figure out the answer on your own.

Expression	**Value**
`s1[5] + s2[7]`	
`s1[len(s1)-1] + s2[1]`	
`s1[4:8]`	
`s2[2:len(s2)-2]`	
`s1[::4]`	

15-110 Hw2 - Programming Portion

Each of these problems should be solved in the starter file available on the course website. Submit your code to the Gradescope assignment Hw2 - Programming for autograding.

All programming problems may also be checked by running the starter file, which calls the function `testAll()` to run test cases on all programs.

Note: this is the first assignment where you will need to do a substantial amount of coding. We encourage everyone to make good use of Piazza, office hours, and small group sessions to get help. In particular, if you attend a small group session in Week 4, your TA will include Problem #5 (`drawIllusion`) as one of the practice problems and will provide more help in solving the problem than is usually available at office hours.

#1 - `pythagoreanChecker(a, b, c)` - 5pts

Write the function `pythagoreanChecker(a, b, c)` which takes three integers, `a`, `b`, and `c`, and checks whether they are a Pythagorean triple. Three numbers (x, y, z) are a triple when x squared plus y squared is equal to z squared. Note that the numbers `a`, `b`, and `c` may be given in any order.

#2 - `printPrimeFactors(x)` - 5pts

Write the function `printPrimeFactors(x)` which takes a positive integer `x` and prints all of its prime factors.

A prime factor is a number that is both prime and evenly divides the original number (with no remainder). So the prime factors of 70 are 2, 5, and 7, because 2 * 5 * 7 = 70. Note that 10 is not a prime factor because it is not prime, and 3 is not a prime factor because it is not a factor of 70.

Prime factors can be repeated when the same factor divides the original number multiple times; for example, the prime factors of 12 are 2, 2, and 3, because 2 and 3 are both prime and 2 * 2 * 3 = 12. The prime factors of 16 are 2, 2, 2, and 2, because 2 * 2 * 2 * 2 = 16.

Here's a high-level algorithm to solve this problem. When we find factors by hand, we repeatedly **divide the number** by the smallest possible factor until the number becomes 1. Our algorithm looks something like this:

1. Set a number n (the factor) to be 2
2. Repeat the following procedure until the number x becomes 1
    a. If the current number x is divisible by n
        i. Print the number n
        ii. Set x to x divided by n
    b. If it doesn't
        i. Set n to be n plus 1

#3 - `factorial(x)` - 5pts

Write the function `factorial(x)` which takes a non-negative integer, `x`, and returns x!. Recall that x! = x*(x-1)*(x-2)*...*3*2*1. **You may not use the built-in function `math.factorial()`; that would make this too easy.**

#4 - `printTriangle(n)` - 10pts

Write a function `printTriangle(n)` which prints an ascii art triangle out of asterisks based on the integer `n`. For example, `printTriangle(5)` would print the following:

```
*
**
***
**
*
```

Note that the triangle is five lines long, with the top and bottom line each having only one asterisk, the second and second-from bottom lines each having two asterisks, etc. So `printTriangle(9)` would look like:

```
*
**
***
****
*****
****
***
**
*
```

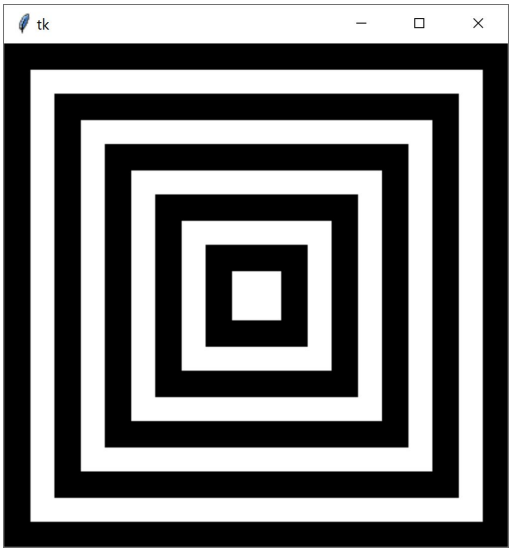
You'll want to create a loop where each iteration prints a single line of the triangle. To draw multiple stars on a single line, consider using a nested loop or the * operator.

Note: `n` is guaranteed to be positive and odd.

#5 - `drawIllusion(canvas)` - 10pts

Write the function `drawIllusion(canvas)` which takes a Tkinter canvas and draws the illusion shown below. You must use a **loop** to do this; don't hardcode a large number of rectangles.

Hint: it's easiest to make this illusion by drawing **overlapping squares**. Start with the largest black square, then draw the next-largest white square, etc. You'll need to draw 10 squares total. The canvas is 400px wide, so each square will be 40 pixels smaller than the previous one in size.



#6 - `getSecretMessage(s, key)` - 10pts

You can hide a secret message in a piece of text by setting a specific character as a key. Place the key before every letter in the message, then fill in extra (non-key) letters between key-letter pairs to hide the message in noise.

For example, to hide the message "computer" with the key "q", you would start with "computer", turn it into "q**c**q**o**q**m**q**p**q**u**q**t**q**e**q**r**", and then add extra letters as noise, perhaps resulting in "orup**qc**rzyp**qo**m**qm**hcy**qp**whh**qu**t**qt**xt**qe**ye**qr**pa". To get the original message back out, copy every letter that occurs directly after the key, ignoring the rest.

Write a function `getSecretMessage(s, key)` that takes a piece of text holding a secret message and the key to that message and returns the secret message itself. For example, if we called the function on the long string above and "q", it would return "computer". You are guaranteed that the key does not occur in the secret message.

Hint: loop over every character in the string. If the character you're on is the key, add the **next** character in the string to a result string.

#7 - `getMiddleSentence(s)` - 10pts

Write the function `getMiddleSentence(s)` that takes a string `s`, checks whether it has exactly three sentences, and if it does, returns the middle sentence. We define a sentence to be a consecutive string of one or more non-whitespace characters that ends in one of the following characters: **`. ! ?`**

For example, given the following string:
`"You've got to ask yourself a question. Do I feel lucky? Well, do ya, punk!"`

The function should return `"Do I feel lucky"`. Note that we remove the punctuation at the end of the returned sentence for simplicity.

If the inputted string does not have exactly three sentences, you should instead return `"Improper structure"`. Note that the test cases are guaranteed to not use ., !, or ? inside a sentence, and that each sentence will only end in one punctuation mark.

To solve this problem, you should use **string operations and methods**. Specifically:

- `s.replace()` can help manage multiple punctuation types
- `s.count()` can detect if there are exactly three sentences or not
- `s.find()` and slicing can help find the beginning and end of the middle sentence
- `s.strip()` can remove any unnecessary whitespace in the returned value