15-110 Hw2 - Written Portion

Name:

AndrewID:

#1 - Expression to Truth Table and Circuit - 12pts

Given the boolean expression shown below, fill out the truth table below 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.

Click here to add your image

#2 - Loop Variable Values - 6pts

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.

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

Prompt	Start Value	Continuing Condition	Update Action
A			
B			
C			

#3 - Writing Test Cases - 5pts

Assume we have written a program `isPositiveEvenInt(value)`, which returns `True` if the given value is a positive **and** even **and** an integer, and `False` otherwise. Write a set of test case assertions for this function that fulfill the following test case types.

Test Case Type	Code
Average Case	
Edge Case	
Special Case	
Varying Result	
Large Input Case	

#4 - Debugging with Variable Tables - 8pts

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. Trace the code and fill out the debugging table for inputs 5, 9, and 28; 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 to too many times
- ☐ The count variable is only added to once
- ☐ The missing else statement causes an error

#5 - 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)`	

#6 - Code Tracing with For-Each Loops - 4pts

In twenty words or less, describe what this function returns on a general input string, s.

```
def mystery(s):
    t = ""
    for c in s:
        if "A" <= c and c <= "Z":
            t = t + c
    return t
```

Answer:

#7 - Linear Search Debugging - 9pts

The following three functions all attempt to implement linear search, as we discussed in lecture. Only one is correct. Identify which of the three functions is correct, then explain what is wrong with the other two and how they can be fixed.

```
def linearSearchA(s, c):
    i = 0
    for char in s:
        i = i + 1
        if char == c:
            return i
    return -1

def linearSearchB(s, c):
    i = 0
    while i < len(s):
        if s[i] == c:
            return i
        i = i + 1
    return -1

def linearSearchC(s, c):
    for i in range(len(s)+1):
        if s[i] == c:
            return i
    return -1
```

Which implementation is correct?

- ☐ linearSearchA
- ☐ linearSearchB
- ☐ linearSearchC

Why are the other two incorrect, and how can they be fixed?

#8 - 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.

#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 (see next page):

```
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 n divides x evenly
        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 - Debugging and Testing - 10pts

There are three functions in the starter file that have been commented out with a triple-quote. Each function has exactly one bug. Use the debugging tactics we discussed in class to identify the bugs and fix them. **Do not attempt to write the functions from scratch**; try to change as little as possible while fixing them.

The first function, `isEven(x)`, is supposed to return `True` if `x` is even, and `False` otherwise. For example, `isEven(12)` should return `True`.

The second function, `battleTextGenerator(person1, person2)`, is supposed to return a string with a battle announcement between the two listed people. For example, `battleTextGenerator("The Rock", "Hulk Hogan")` would return `"TONIGHT: The Rock VS Hulk Hogan"`.

The third function, `makeAdditionString(x, y)`, is supposed to take two integers and return a string showing the addition equation between the two. For example, `makeAdditionString(3, 4)` would return `"3 + 4 = 7"`.

#4 - `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.**

#5 - `printTriangle(n)` - 5pts

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:

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

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

Hint: consider using the `*` string operator to make this problem easier.

#6 - `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