15-110 Hw2 - Written Portion

Name:

AndrewID:

Complete the following problems in the fillable PDF, or print out the PDF, write your answers by hand, and scan the results. Also complete the programming problems in the starter file hw2.py from the course website.

When you are finished, upload your hw2.pdf to **Hw2 - Written** on Gradescope, and upload your hw2.py file to **Hw2 - Programming** on Gradescope. Make sure to check the autograder feedback after you submit!

Written Problems

#1 - Python Error Identification - 5pts

Can attempt after Booleans, Conditionals, and Errors lecture

For each of the following lines of code, select whether it causes a **Syntax Error, Runtime Error**, or **No Error** (choose only one answer each). You are guaranteed that no code has a logical error and that no variables are defined before the code runs.

```
print("Hello World"
```

- ☐ Syntax Error
- ☐ Runtime Error
- ☐ No Error

```
print(Test)
```

- ☐ Syntax Error
- ☐ Runtime Error
- ☐ No Error

```
print("2+2=" + 4)
```

- ☐ Syntax Error
- ☐ Runtime Error
- ☐ No Error

```
x - y = 5
```

- ☐ Syntax Error
- ☐ Runtime Error
- ☐ No Error

```
x = 1 == 2
```

- ☐ Syntax Error
- ☐ Runtime Error
- ☐ No Error

#2 - Boolean Expression to Truth Table/Circuit - 7pts

Can attempt after Circuits and Gates lecture

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 add the image in the correct location and delete the blank page if you do this so that Gradescope will find your image correctly.

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

Click here to add your image

If that doesn't work, see instructions on the previous page.

#3 - Full Adder Facts - 6pts

Can attempt after Circuits and Gates lecture

In class and in the lecture slides, we showed how to put together a Full Adder circuit. For each of the following questions, choose the **best** answer as relates to that circuit.

What are X and Y?

- ☐ The two whole numbers being added
- ☐ Single binary digits of the two numbers being added
- ☐ Two binary digits of the first number being added

What is C_{in}?

- ☐ The third whole number being added
- ☐ A single binary digit of the third number being added
- ☐ The number carried in from the previous addition
- ☐ The remainder of the current addition

Why do we need two output values?

- ☐ To manage the large number of gates
- ☐ To account for both of the inputs
- ☐ To hold both the result and the original number
- ☐ To hold both the result and the number that will be carried over

#4 - Code Tracing While Loops - 6pts

Can attempt after While Loops lecture

Given the following block of code, fill out a variable table that shows the values of the variables at the **end** of each iteration of the loop. You may not need to fill out values for every listed iteration.

```
x = 0
y = 10
z = 0
while x <= y:
    x = x + 3
    y = y + 1
    z = (x + y) - z
    print(x, y, z)
```

	x value	**y value**	**z value**
Pre-loop	0	10	0
Iter 1			
Iter 2			
Iter 3			
Iter 4			
Iter 5			
Iter 6			
Iter 7			
Iter 8			

#5 - For Loop Control Variable Values - 6pts

Can attempt after For Loops lecture

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 Indexing and Slicing - 10pts

Can attempt after String Indexing, Slicing, and Looping lecture

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

```
s1 = "coding 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[7] + s2[6]`	
`s1[1] + s2[len(s2)-1]`	
`s1[7:11]`	
`s2[2:len(s2)-2]`	
`s1[::4]`	

Programming Problems

For each of these problems (unless otherwise specified), write the needed code directly in the Python file, in the corresponding function definition.

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

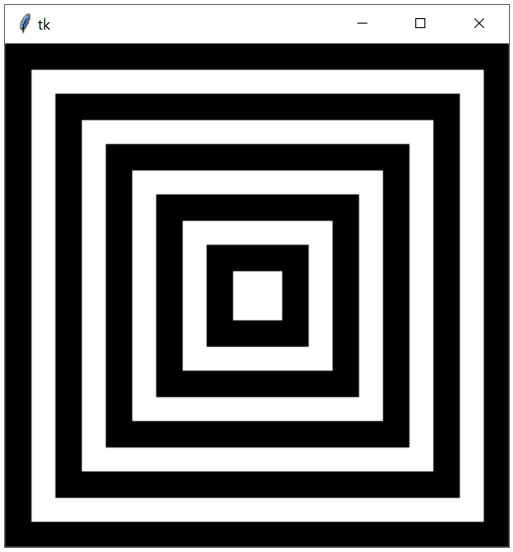
Note: Hw2 is the first assignment where you will need to do a substantial amount of coding. We encourage everyone to make good use of Piazza, office hours, and small group sessions to get help.

In particular, if you attend a small group session in Week 4, your TA will include Problem #1 (`drawIllusion`) as one of the practice problems and will provide more help in solving the problem than is usually available at office hours.

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

Can attempt after For Loops lecture

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. (You can use either a while loop or a for loop, whichever you prefer).



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 should be 20 pixels smaller on each side than the previous one (with the last square being exactly 40 pixels wide).

Another Hint: start by considering what the loop control variable should be. Which values need to change as you move to the next square? How do those values relate to the loop control variable? Consider our approach to drawing a grid in lecture as well.

#2 - `partialProduct(n, x)` - 10pts

Can attempt after For Loops lecture

Write the function `partialProduct(n, x)` which takes a non-negative integer `x` and a non-negative integer `n` and returns the product n * (n+1) * (n+2) * … * (x-1) * x. You are guaranteed that `n` will always be smaller than or equal to `x`.

You may not use any built-in functions in the `math` library. Instead, you must use a **for loop** to solve this problem.

Hint: consider the sum-1-to-10 problem we went over in lecture. You can use a very similar approach to solve this problem.

#3 - `printDiamond(n)` - 10pts

Can attempt after For Loops lecture

Write a function `printDiamond(n)` which prints ascii art of a diamond with a size based on the positive integer `n`. For example, `printDiamond(4)` would print:

```
   11
  2**2
 3****3
4******4
 3****3
  2**2
   11
```

Whereas `printDiamond(3)` would print:

```
  11
 2**2
3****3
 2**2
  11
```

You'll want to create a loop where each iteration prints a single line of the ascii art. To draw multiple spaces and multiple asterisks on a single line, consider using the * operator, which can be used to repeat a string an integer number of times.

Hint 1: Every line is composed of three parts: outer spaces, inner asterisks, and two numbers on the outside of the diamond. For example, the second line of the size=3 diamond has one space, then the number 2, then two asterisks, then the number 2 again. Consider each of these parts individually, note how they change between iterations, then determine how to map the loop control variable to each part separately.

Hint 2: if you're feeling overwhelmed, simplify the problem by **breaking it down into parts**! Start by just making the top half of the diamond. First, get the numbers to appear correctly; second, add in the asterisks. Finally, add the leading spaces.

Hint 3: how can the program switch from the increasing top half to the decreasing bottom half? Consider using **two separate loops** (one going up, one going down), or a single loop with a conditional that changes how the loop control variable is used.

#4 - `printPrimeFactors(x)` - 10pts

Can attempt after For Loops lecture

Using the algorithm shown below, write the function `printPrimeFactors(x)` which takes a positive integer `x` and prints all of its prime factors in a nice format.

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. We'll display repeated factors on a single line as a power expression; for example, 16 would display 2 ** 4, because 2 is repeated four times.

Here's a high-level algorithm to solve this problem. Start by iterating through all possible factors. When you find a viable factor, repeatedly **divide the number** by that factor until it no longer evenly divides the number. Our algorithm looks something like this:

1. Repeat the following procedure over all possible factors (2 to x, inclusive)
    a. If x is evenly divisible by the possible factor
        i. Set a number count to be 0
        ii. Repeat the following procedure until x is not divisible by the possible factor
            1. Set count to be count plus 1
            2. Set x to x divided by the factor
        iii. If the number count is exactly 1
            1. Print the factor by itself
        iv. If the number count is greater than 1
            1. Print "f ** c", where f is the factor and c is the count

As an example, if you call `printPrimeFactors(600)`, it should print

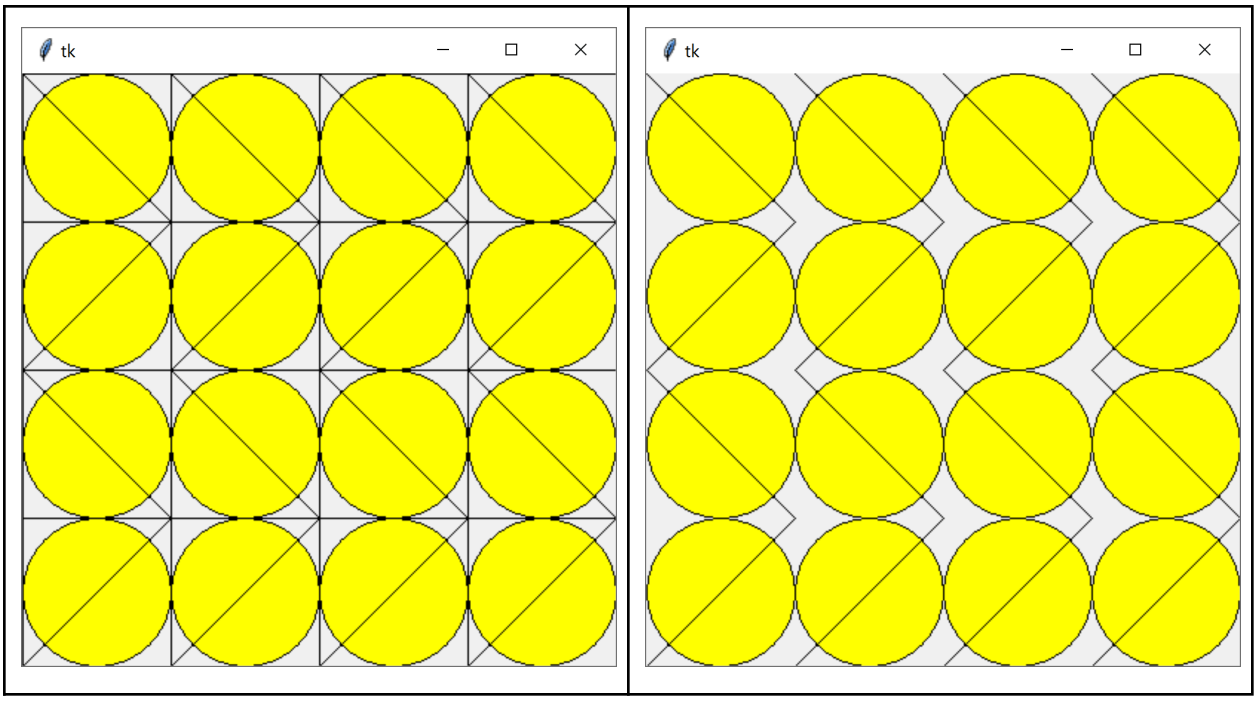
```
2 ** 3
3
5 ** 2
```

#5 - Repeating Pattern - 10pts

Can attempt after For Loops lecture

Write a function `repeatingPattern(canvas, numCells, cellSize, showGrid)` which takes a Tkinter canvas, two integers, and a Boolean, and draws a repeating pattern in a grid of squares, all on the given canvas. Further constraints are listed on the next page, after a short example.

Here is a simple example of a repeating pattern on a grid that meets the constraints, with `numCells` set to `4`, `cellSize` set to `100`, and `showGrid` set to `True` on the left and `False` on the right. The pattern contains two calls (one to create a circle, one to create a line) and a keyword argument (the circle color) and alternates over the rows of the grid (even rows have the diagonal from top-left to bottom-right, odd rows have the diagonal from top-right to bottom-left).



(continued on next page)

You can design whatever pattern you like, but your program must meet a few constraints:

- The program must create a `numCells` x `numCells` grid of squares of the given `cellSize` on the canvas based on the given parameters. The grid should only be drawn if `showGrid` is `True`.
- Each cell of the grid must contain its own instance of a pattern. (You are welcome to have more than one pattern, but they must repeat in some way).
- The pattern must use **at least two different Tkinter function calls** and **at least one keyword argument**. For example, you could draw a circle with a smaller square inside it, and set the square to have a specific color.
    - You **cannot** use the grid-square as one of the two Tkinter function calls; the two calls must be distinctly visible when the grid is on.
- There must be at least two **alternate versions** of the pattern included in the grid. The alteration can be small - perhaps you can change the color used to draw the pattern, or the pattern's orientation, or some other way of varying what is drawn.
- The alternate patterns must alternate in a **systematic fashion**. This could mean alternating over the rows, or the columns, or in diagonals, or like a checkerboard - whatever works for you!
- To get full credit, your solution **must use loops**. We went over how to draw grids with loops in lecture - check the slides for an example!

Hint: a good way to approach this problem is to write a **helper function** `drawPattern(canvas, left, top, size)` (potentially with other parameters too) that draws the pattern on the canvas at the given pixel location in the given size. You can then call this helper function repeatedly inside a **loop** in `repeatingPattern`.

Fun fact: repeating patterns with less systematic alternation were used in some of the most famous art by Pittsburgh-native artist **Andy Warhol**, like the *Shot Marilyns* (https://en.wikipedia.org/wiki/Shot_Marilyns).

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

Can attempt after String Indexing, Slicing, and Looping lecture

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.