CS Scholars - Programming Hw2 - Written
Due Date: Friday 07/14 EOD

Name:

AndrewID:

For full credit on the assignment, complete either all Review + Core problems (#1-#10) or all Core + Spicy problems (#1-#5, #8-#12).

Bonus problems are related to the Advanced Track content, and are optional.

Core Problems - Written

#1 - Code Tracing Conditionals

Can attempt after Booleans and Conditionals lecture

Given the following block of code, choose specific values for x, y, and z that would lead to the code printing A, B, C, D, or E. If one of the variables could be assigned to any value to achieve the result, write the word **anything** instead of a value. Fill out your answers in the table below.

```python
if x < 10:
    if y > 20:
        if z == "foo":
            print("A")
    else:
        if y % 2 == 0:
            print("B")
        else:
            print("C")
elif x < 100:
    if y < 0 and z == "bar":
        print("D")
    elif y < 0:
        print("E")
```

Printed Result	x value	y value	z value
A			
B			
C			
D			
E			

#2 - Python Error Identification

Can attempt after Errors, Debugging, and Testing lecture

For each of the following lines of code, select whether it causes a **Syntax Error, Runtime Error**, or **No Error**. 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

#3 - Writing Test Cases

Can attempt after Errors, Debugging, and Testing lecture

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 five test case types we discussed in lecture. Your test cases should be runnable in Python (use `assert` statements!).

#4 - Loop Control Variables

Can attempt after While Loops lecture

Each of the following problem prompts could be implemented using a loop. Identify the start value, continuing condition, and update action for the loop control variable you would use in that 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, including 2 but not including 20.

B) Output the numbers from 10 to 1, inclusive on both.

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			

#5 - For Loops

Can attempt after For 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. Don't just copy the code into the editor; trace it yourself! You may not need to fill out values for every listed iteration.

```python
x = 0
y = 0
for z in range(3, 15, 2):
    x = x + z
    if x % 2 == 0:
        y = y + 1
    print(x, y, z)
```

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

Programming Problems

Each of these problems should be solved in the starter file available on the course website. They should be submitted to the Gradescope assignment Hw2 - Programming to be autograded. Make sure to check the autograder feedback after you submit!

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

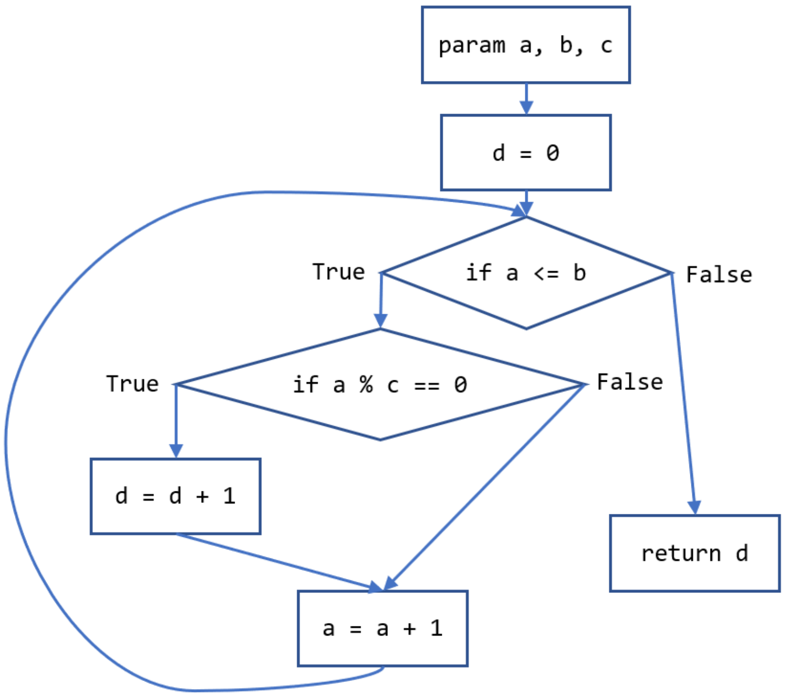
To test your code before submitting, 'Run file as script' and make sure all the test functions pass.

Review Problems - Programming

#6 - Flow Chart to Program

Can attempt after While Loops lecture

Given the control flow chart below, write a function `mysteryFunction(a, b, c)` that implements the control flow chart correctly. You should use a **while loop**.



#7 - `printSquare`

Can attempt after Loops lecture

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

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

Note that the square is five lines long, with each line having five asterisks. As another example, `printSquare(8)` would look like:

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

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

Note: `n` is guaranteed to be positive.

Core Problems - Programming

#8 - Interactive Program

Can attempt after Conditionals lecture

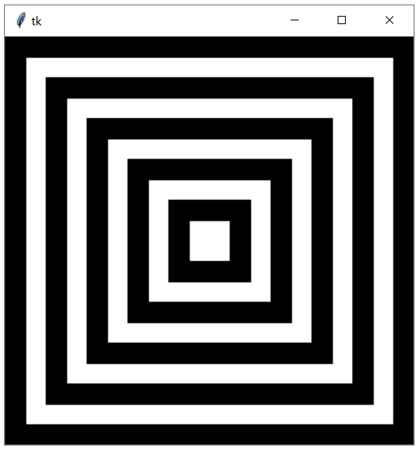
In the function **`interactiveProgram`**, use the `input` function and conditionals to set up a short interactive program of your own design. This could be a very short choose-your-own-adventure story, or a Buzzfeed-style quiz, or whatever else you'd like! The only requirements are:

1. You must use the `input` function to collect information from the user at least three times.
2. The `interactiveProgram` function must take no parameters
3. You must use **conditionals** somewhere in your code. There should be at least two `if` statements and at least one `elif` or `else` statement.
4. All the code for your interactive program must be in the `interactiveProgram` function (or helper functions that `interactiveProgram` calls).

#9 - `drawIllusion`

Can attempt after While Loops lecture

Write the function `drawIllusion(canvas)` which takes a Tkinter canvas and draws the illusion shown below. You must use a **while 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 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?

#10 - `printPrimeFactors`

Can attempt after For Loops lecture

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. To find factors manually, iterate 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)
    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
```

Spicy Problems - Programming

#11 - `isHappyNumber`

Can attempt after While Loops lecture

Write the function `isHappyNumber(n)` which takes a positive integer n and returns `True` if `n` is a happy number and `False` otherwise. We say a number is happy if repeatedly replacing the number with the sum of the squares of its digits eventually leads to the number 1 (which is a steady state, as $1^2 == 1$).

For example, given the number 49, replacing the number with the sum of the squares of its digits gives us the sequence:

49 -> $4^2 + 9^2$ = 97 -> $9^2 + 7^2$ = 130 -> $1^2 + 3^2 + 0^2$ = 10 -> $1^2 + 0^2$ = 1

Therefore, 49 is a happy number.

How can we tell if a number is **not** happy? Any unhappy number will eventually end up in a cycle that includes the number 4:

4 -> 4^2 = 16 = $1^2 + 6^2$ = 37 -> $3^2 + 7^2$ = 58 -> $5^2 + 8^2$ = 89 -> $8^2 + 9^2$ = 145 -> $1^2 + 4^2 + 5^2$ = 42 -> $4^2 + 2^2$ = 20 -> $2^2 + 0^2$ = 4 -> …

So if the sum of the squares of the digits ever becomes 4, you know that the number is not happy.

Hint: the loop control variable for this problem is tricky. Try to replicate the repeated process shown above.

Another hint: you will need **two** while loops to solve this problem, not one!

#12 - `printDiamond`

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.

Bonus Problems

Advanced Programming 1 - Recursion

Assume you want to write a function `recursiveSum` that takes a positive integer, `n`, and **recursively** computes the sum from one to `n`.

For example, the result when calling the function on n=5 is 5+4+3+2+1 = 15.

What condition do you need to check for your **base case**, and what do you return?

What is the recursive call on a smaller problem in the **recursive case**, and how do you use that result to solve the whole problem for n?

Advanced Programming 2 - Recursion

In the programming starter file, write the function `powerSum(n, k)` that takes two non-negative integers `n` and `k` and returns the so-called power sum: $1^k + 2^k + ... + n^k$. You must use **recursion** to solve this problem: for loops, while loops, and the function `sum` are not allowed.

Note that the test function for `powerSum` is commented out; you'll need to uncomment it to test your function.

Advanced Computer Science - Concurrency

You are managing a volunteer organization that makes peanut butter and jelly sandwiches. The steps to make a PB&J sandwich are:

1. [P] Spread peanut butter on one slice of bread (30 seconds)
2. [J] Spread jelly on top of the peanut butter (30 seconds)
3. [S] Complete the sandwich by putting a slice of bread on top (15 seconds)

Originally each worker makes one sandwich at a time, with all three workers working in parallel. Each of the cells in the following table represents 15 seconds, with the whole table representing three minutes of work. Fill in the cells with the letters representing the steps to demonstrate the original system the volunteers used.

Logistical note: If a step spreads across multiple cells, designate this with a dash (-) in the cell(s) following the first one. If no work occurs in a cell, leave it blank.

Worker	00:00	00:15	00:30	00:45	01:00	01:15	01:30	01:45	02:00	02:15	02:30	02:45
A												
B												
C												

How many complete sandwiches could be made by three workers in three minutes with the original system?	

Recent changes have affected the ingredients you have to work with; now you just have one giant tub of peanut butter and one giant tub of jelly. You decide to reorganize the workers to accommodate the changes while still being efficient. Create a new schedule that uses **pipelining** to make sandwiches instead.

Worker	00:00	00:15	00:30	00:45	01:00	01:15	01:30	01:45	02:00	02:15	02:30	02:45
A												
B												
C												

How many complete sandwiches could be made by three workers in three minutes with the new pipeline?	