CS Scholars - Programming Final Evaluation
Evaluation Period: Thursday 07/27 1:30pm-2:45pm EST

Name:

AndrewID:

This final evaluation will help you evaluate the knowledge you have accumulated over the course. You should work on this evaluation during the class period on Thursday 07/27 (from 1:30pm - 2:45pm EST) and submit it to Gradescope by 3pm EST on the same day. It's okay if you can't finish all the problems - just do what you can!

This final evaluation is open-note, but closed to collaboration. You are welcome to consult the course notes, personal notes, and your past homework assignments during the evaluation, but you should not communicate with anyone outside of the course staff during the period of the evaluation. In particular, do not collaborate with other students.

The final evaluation consists of both a written portion and a programming portion. The written problems should be completed in this starter file; the programming problems should be completed in the programming starter file.

Written Problems

#1 - Data and Variables

Consider this block of code:

```
a = 7
b = "Go" + "Team"
a = a + 1
c = 15 / 2
d = (5 + 5) < 10
#e = True
f = min(a, c)
print(b, "=", f)
```

In the table, list all the **variables** that are defined in the code and the **values** they hold at the end of the code run. You may not need to use all the lines of the table.

Variable	Value

#2 - Conditionals

Consider the following code:

```
if x > 50:
    if (y % 2 == 0) and (y > 20):
        print("A")
    elif (y % 2 == 0) or (y > 20):
        print("B")
    else:
        print("C")
elif x < 50:
    if y == 42:
        print("D")
    else:
        print("E")
```

Choose specific values for x and y that would lead to the code printing each of the given results. Fill out your answers in the table below. If one of the variables could be assigned to any value to achieve the given result, write the word **anything** instead of a value. If there is no possible combination of values that would lead to the given result, write the word **nothing** instead of a value.

Note: the last row should be taken literally; which values would lead to nothing being printed?

	x value	**y value**
prints A		
prints B		
prints C		
prints D		
prints E		
Nothing is printed		

#3 - Strings and Lists

Consider the following list:

```
lst = [ "you", "can", "code", "if", "you", "practice", "each", "day" ]
```

Write a snippet of code that accesses the value `"code"` from the list.

Write a snippet of code that accesses the last three values from the list, all together as one sublist. Use **slicing** to do this.

Write a snippet of code that uses a **method** to find the number of times the value `"you"` occurs in the list. Store the answer in a variable called `result`. (You should not use a loop!)

Write a snippet of code that uses a **loop** to find the number of times the value `"you"` occurs in the list. Store the answer in a variable called `result`. (You should not use any methods!)

Programming Problems

#4 - Function Definitions and Calls

Write the function `divideCost(totalCost, tipPercent, numPeople)`. This function takes **three parameters**: the total cost of a meal (a float), the percent you want to tip (a float), and the number of people who are splitting the bill (an integer). It calculates the total post-tip, divides that total between the given number of people, and **returns** a float: the amount each person owes, rounded to two significant digits.

For example, `divideCost(46.7, 0.18, 3)` would calculate the post-tip cost as `46.7 + 46.7 * 0.18 = 55.106`. Dividing that across three people would yield a per-person cost of `18.3686666667`. Rounded to two digits, that would be `18.37`, which is returned.

Note: for this problem you need to write the function header yourself. You should put it under the comment `''' #4 - Function Definitions and Calls '''` in the starter file.

#5 - Loops

Write a function `makeNumberString(num)` which takes an integer, `num`, and returns a string that follows a certain pattern up to the number `num`.

The pattern starts with a single 1, then two 2s, then a dash. Then there are three 3s, then four 4s, then another dash. This continues until the number `num` is reached; then the pattern ends.

For example, if we called `makeNumberString(7)`, the function would return:

```
"122-3334444-55555666666-7777777"
```

On the other hand, if we called `makeNumberString(4)` the function would return `"122-3334444-"`.

Hint: start by just generating the string of numbers. Once that's working, try to add in the dashes at the proper places. Pay close attention to the properties of the numbers the dashes come after…

#6 - Algorithmic Thinking

Your task is to translate an algorithm for the function `possibleWords(letters, wordList)` which takes `letters`, a list of single-character strings, and `wordList`, a list of words (strings), and returns a new list containing only the words from `wordList` that can be written using only the letters that occur in `letters`.

For example, given the list of letters `["a", "c", "d", "e", "f", "o"]` and the word list `["code", "dart", "face", "feed" ]`, the function would return `["code", "face"]`. Note that `"dart"` cannot be written because the letters list does not contain the letter `"t"`, and `"feed"` cannot be written because the letters list only contains one `"e"`.

You do not need to write an algorithm for this problem from scratch - we're providing one for you! You just need to **translate** this algorithm from plain language into Python code. Here is the algorithm:

1. Create a empty list that will hold the chosen words
2. Loop over each of the words in the word list
    a. Create a copy of the word in a new variable
    b. Loop over each of the letters in the letters list
        i. If the letter is in the copied-word
            1. Find the index where the letter occurs in the copied-word
            2. Update the copied-word variable so that it no longer contains the letter at that index.
    c. If the copied-word is an empty string
        i. Add the original word to the chosen words list
3. Return the chosen words list

Hint: when updating the copied-word variable to remove a letter, use **string slicing** to create a new version containing the letters up to that index, then the letters after that index.

#7 - User Interaction

Using our interaction framework, write an interactive program where the user can manipulate the size, position, and color of a rectangle using their mouse and keyboard. Specifically, the user should be able to do the following:

- **The rectangle grows and shrinks vertically in response to key presses:** If the user presses the up key, the rectangle should grow in height by 10 pixels (the top should move 5 pixels further up, and the bottom move 5 pixels further down). Likewise, if the user presses the down key, the rectangle should shrink in size by 10 pixels (the top moves 5 pixels down and the bottom moves 5 pixels up). The user should not be able to shrink the rectangle below 10 pixels in size.
- **The rectangle moves horizontally in response to mouse clicks:** If the user clicks to the left or right of the rectangle, outside of the bounds of the rectangle, the rectangle moves so that it is horizontally centered in the clicked location. However, the vertical center should remain the same; the rectangle only moves horizontally, not vertically.
- **The rectangle changes color in response to key presses:** if the user presses the 'r' key, the color changes to red; when the user presses 'g', it becomes green; when the user presses 'b', it becomes blue.
- **The rectangle changes color in response to mouse clicks:** if the user clicks inside the rectangle (inside both vertical and horizontal bounds), the rectangle color changes to a randomly-chosen color (orange, yellow, or purple).

Outside of these constraints, you can design the program however you'd like.