CS Scholars - Programming Final Evaluation
Evaluation Period: Thursday 07/28 10:30am-12:00pm EST

Name:

AndrewID:

This final evaluation will test the knowledge you have accumulated over the course. You should complete this evaluation during the class period on Thursday 07/28 (from 10:30am - 12:00pm EST) and submit it to Gradescope by 12:05pm EST on the same day.

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 - Variables and Functions
- #2 - Loops
- #3 - Debugging and Testing

Programming Problems
- #4 - Function Definitions and Conditionals
- #5 - Nesting and Strings
- #6 - User Interaction

Written Problems

#1 - Variables and Functions

Consider this block of code:

```
1 a = 9
2 b = "Ready" + "Go"
3 a = a + 1
4 c = 23 / 2
5 d = (5 + 5) < 10
6 result = max(a, c)
7 print(b, result)
```

In the first table, list all the **variables** that occur in the code and their **values** and **types** at the end of the code run. You may not need to use all the lines of the table.

Variable	Value	Type

In this table, list all the **functions** that are called in this code, their **arguments**, and their **returned values**. Again, you may not need to use all the lines in the table.

Function	Arguments	Returned Value

#2 - Loops

Consider this block of code:

```
result = 0
for i in range(2, 20, 6):
    result = result + i
    print(i, result)
print("---")
s = ""
while result > 1:
    s = s + str(result) + ","
    result = result // 2
    print(result)
print("---")
print(s)
```

For both the **for loop** and the **while loop**, write in the **loop control variable's** name, start value, continuing condition, and update action.

Loop	Name	Start Value	Continuing Condition	Update Action
For				
While				

In the box below, write what is printed to the interpreter when the code above runs.

#3 - Debugging and Testing

A friend asks you for help with a program they're writing. Their code is supposed to take a list of strings, return True if the strings are all in length order (each string is the same length or longer than the string that came before it), and return False if it's out of order. The code doesn't always work as expected, but it doesn't cause any syntax or runtime errors, and it passes the test set they wrote. Their code and tests are included below:

```
1  def inLengthOrder(lst):
2      if len(lst) <= 1:
3          return True
4      for i in range(1, len(lst)):
5          if len(lst[i-1]) > len(lst[i]):
6              return False
7          else:
8              return True
9
10 def testInLengthOrder():
11     print("Testing inLengthOrder()...", end="")
12     assert(inLengthOrder(["a", "aa", "aaa", "aaaa"]) == True)
13     assert(inLengthOrder(["one", "two", "three"]) == True)
14     assert(inLengthOrder(["test"]) == True)
15     print("... done!")
```

Give an example of a **test case** you would add to make the test set more robust, and explain why you would add it.

Briefly describe a **debugging strategy** you might use to find the problem after adding a new test case.

Programming Problems

#4 - Function Definitions and Conditionals

Write a Python function, `probableRobot`, which takes information about a person filling out a form online and returns a Boolean representing whether or not that person is probably a robot.

The function takes three arguments (in this order): `checkedBox` (a Boolean, whether or not the person checked a box saying they are not a robot); `responseSpeed` (an integer, the number of milliseconds it took for the person to check the box); and `cookieHistory` (a list of strings, past interactions the person has had with the website).

The function should return `True` if either of two conditions is met:

- The user did not check the box (`checkedBox` is `False`)
- The user checked the box too quickly (`responseSpeed` is less than `10` milliseconds) and the user has no history with the website (the `cookieHistory` list is empty)

Otherwise, the function should return `False`.

For example, `probableRobot(False, 8, ["cookie"])` would return `True` because `checkedBox` is `False`. `probableRobot(True, 326, [])` would return `False` because `checkedBox` is `True`, and though `cookieHistory` is an empty list, `responseSpeed` is not less than `10`.

Note that for this problem, a function header is **not** provided; you'll need to write the full function header yourself. You should write this function under the comment
`''' #4 - Function Definitions and Conditionals '''`

#5 - Nesting and Strings

Write a Python function `getPunctuationFrequency(text, punc)` which takes two strings - a piece of text and a single character (a punctuation mark) - and returns the frequency of how often that punctuation mark occurs in the text compared to other punctuation marks. We define punctuation marks as the characters in the library variable `string.punctuation`, which holds the string `"!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"`.

For example, `getPunctuationFrequency("Really?! I don't know, is that true?? I guess I'm excited then!", "?")` would find 3 `"?"` characters in the string and 8 total punctuation marks, for an overall frequency of 3/8 = 0.375, and would return `0.375`.

If the given text contains no punctuation marks, you should return `0`, not `NaN`.

#6 - 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, so 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.