Final Exam A

Comp 123

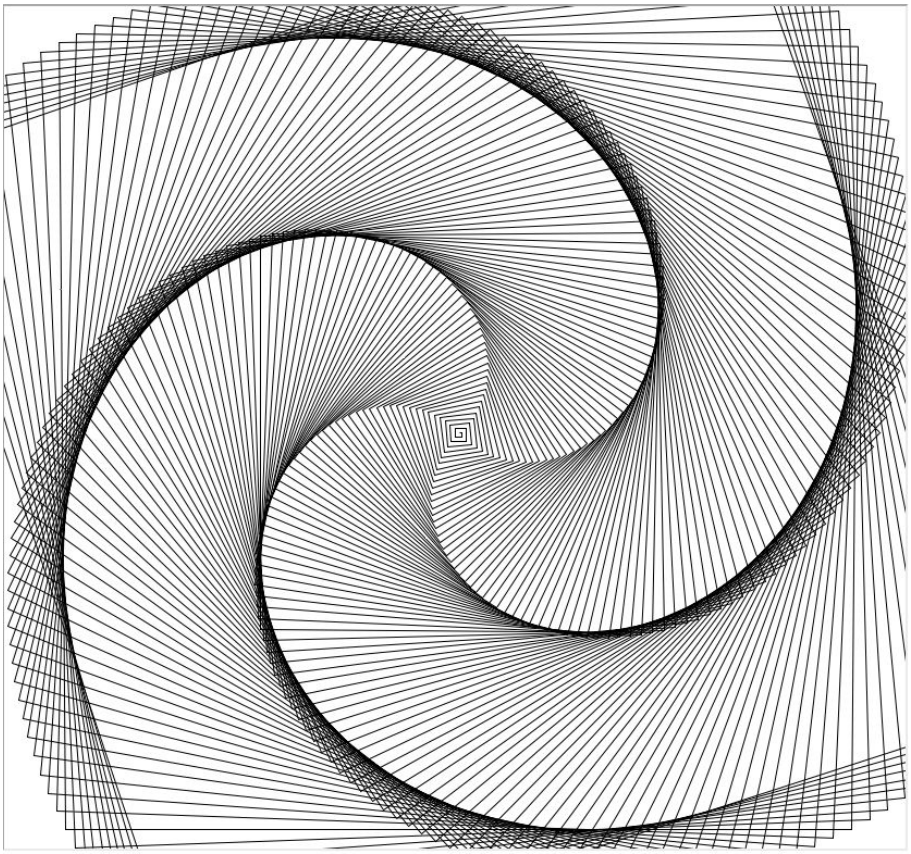
- You may use the computer, and any resources on Moodle, Piazza, the Interactive Python textbook, your online journal, or the Python.org documentation pages. You may also use any paper resources you like.
- Put all work in the file listed for each problem.
- You **may and should** ask me for clarifications, or hints.
- You have 2 hours to complete this test.

1. (10 pts) In the file `finalCode.py`, create a function in Python called `containsI`. This function should take in a list of strings as its input. The function should construct and return a new list that has only those strings from the input that contain the characters `'i'` or `'I'`. See the example below.

```
>>> containsI(['camel', 'chimp', 'lion', 'bear', 'penguin',
'moose'])
['chimp', 'lion', 'penguin']
>>> containsI(['I', 'XV', 'VII', 'mmxv', 'iii'])
['I', 'VII', 'iii']
>>> containsI(['apple', 'peach', 'lemon', 'cherry'])
['apple', 'peach', 'lemon', 'cherry']
```

2. (10 pts) In the file `finalCode.py`, create a function called `coolSpiral1`. This will be a turtle graphics function. It should take in three inputs: a Turtle object to do the actual drawing, the initial line length (also used for the increment size), and a number of repetitions. The turtle will draw a straight line of the initial line length, and then will turn exactly 89.5 degrees to the right. It continues drawing a longer line each time; the length of the line goes up by the initial line length. The number of repetitions is the number of straight lines it draws. Below is a sample call, along with what the function should produce for that call.

```
s = turtle.Screen()
tess = turtle.Turtle()
tess.speed(0)
coolSpiral1(tess, 2, 400)
s.exitonclick()
```



The `coolSpiral1` function should repeat as many times as the input number of repetitions says. Each time through, the turtle will move forward, then turn the specified angle (89.5 degrees). The distance should increase each time, increasing by the second input value, the input line length, each time.

3. (10 pts) In the file `finalCode.py`, you will find a function called `findPosition`. This function takes a string keyword and a filename. It reads the text from the file, looking for the first occurrence of the keyword. When it finds it, it reports the line number (counted starting at one) and the position within the line, where the keyword string occurred. If it does not find the keyword, it returns (-1, -1) to indicate that the word did not occur. Below are some sample calls and their correct results.

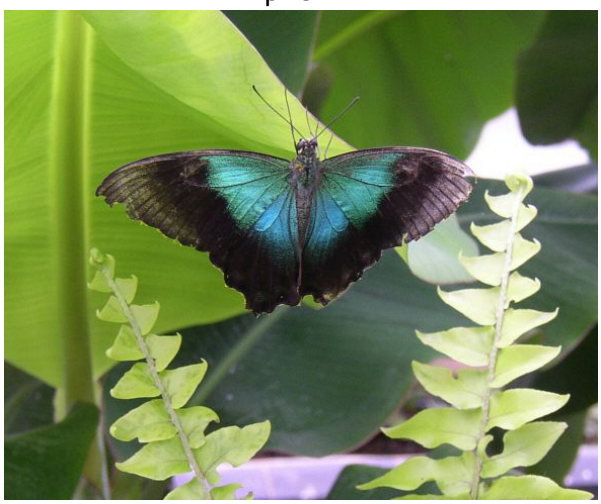
```
>>> findPosition("Anne", "TextFiles/persuasion.txt")
(34, 14)
>>> findPosition("Ishmael", "TextFiles/mobydick.txt")
(507, 8)
>>> findPosition("bank", "TextFiles/alice.txt")
(2, 0)
```

```
>>> findPosition("Frederick", "TextFiles/alice.txt")
(-1, -1)
```

Your task is to find and correct the **two** bugs in this function that prevent it from working correctly. Find them, mark the lines they are on with a description of the bug, and fix the bug!

4. (10 pts) In the file `finalImages.py`, create a function, `quarterPic`, that takes in a Picture object as its only input. It should create a new, blank picture the same size as the original. In this picture, it should place four half-sized copies of the original picture. See below for some examples, the original images, and the correct results.

```
pic1 = makePicture("butterfly2.jpg")
pic2 = makePicture("daisies.jpg")
pic3 = makePicture("horse.jpg")
res1 = quarterPic(pic1)
res2 = quarterPic(pic2)
res3 = quarterPic(pic3)
```

pic1	res1
	
pic2	res2

pic3

res3

This function involves two things we looked at this term: scaling down by half, and copying. There are multiple ways to solve this, but take it step by step: first make the new picture, then figure out how to scale down the original, and then how to copy it four times into the new picture.

5. (10 pts) In the file `finalGUI.py`, you will find the start of a `tkinter` GUI program. This program's purpose is to act like a "color-picker." The program will keep track of specific

red, green, and blue channel values, and will display the corresponding color in the background of the window. Buttons will allow the user to increase or decrease each channel's value, and the background color should automatically update. The pictures below show what the program should display for two different RGB values:

I have created the outline of functions for this program: you will complete them:

- In `colorPickGUI` I have set up the window, and given examples of how to create the buttons and labels. Complete the remaining five buttons and the remaining two labels following my model, and laying them out so they appear like the picture above.
- Each button needs its own callback function. I have provided one, `incrRed`, as a model. This is the callback for the button I created, as well. Define five more callback functions, bound to the other five buttons, so that they work similarly. Use my helper functions where needed.

6. (10 pts) In the file `finalCode.py`, you will find the function definition: `keepPositives`. This function uses recursion to look through a list of numbers, building a new list that keeps only the positive numbers from the list. This function has three bugs in it. Find the bugs, put a comment on the line where the bug occurs describing the bug, and fix the bugs. Below are some sample calls, along with the correct results.

```
>>> keepPositives([1, -2, 3, -4, 5])
[1, 3, 5]
>>> keepPositives([-2, -4, -6])
```

```
[]
>>> keepPositives([5832])
[5832]
```