COMP1405B (Fall 2015)

Assignment 4 of 6 — Due: Monday, November 9th by 11:30pm

For each question in this assignment ***you will be submitting three files containing source code written in Python 3, that have been compressed into a "zip" file.*** *The* ***Python sources*** *(i.e., the .py files)* ***should be named "a4q1.py", "a4q2.py", and "a4q3.py"****. The* ***zip file should be named a4.zip****. You will* ***submit your file using cuLearn****.*

A LATE POLICY IS IN EFFECT FOR THIS ASSIGNMENT
LATE ASSIGNMENTS WILL BE ACCEPTED FOR 48 HOURS AFTER THE DEADLINE AT A PENALTY OF 2.0% / HOUR

You are expected to ***demonstrate good programming practices at all times*** *(e.g., choosing descriptive variable names, provide comments in your code, etc.) and* ***your code may be penalized if it is poorly written****. You are also expected to* ***do the necessary preparatory work*** *(i.e., devising an algorithm)* ***before you start coding****.*

PLEASE NOTE: YOU WILL BE ASKED TO PRESENT EITHER PSEUDOCODE OR A FLOWCHART BEFORE YOU WILL RECEIVE ANY ASSISTANCE FROM THE INSTRUCTOR OR A TEACHING ASSISTANT

Question 1 – ASCII Art Generation

For this question you will use SimpleGraphics to create an ASCII art generator – a program that will convert gif images into sequences of ASCII characters that, when printed to the console, approximate the original image. Your program will load an image (specified by the user), get the dimensions of that image, and then use two for loops to loop through the image and consider every pixel. Each pixel should be converted to a grayscale value and then, depending on how light (i.e., how close the gray value is to 255) the pixel is, print out a character of similar brightness (assuming the program is printing white text on a black background). To clarify, the "@" symbol in white, on a black background, appears brighter than the "." symbol, so printing "@" is a good way to draw a white pixel, printing "." is a good way to draw a dark (almost black) gray, etc. An example is included below:

```
..=:..,.,::=:,::=:::=:==::====::===:=~==
.=.,...,:==,=:::==::~:====:::=::::==:===
=~....,,~~~:III==~=~=~:=~=~~:=~~=~~:~:=~
=:....:~=::,?III???~~~:~III=:=::==:=:===
....,:===~.,I7777777777I+II=======:~~=:=
....==:,,,:~777I77I7III~=+?=:=======~==:
..,:=:::,,I7~~7IIIIIIII=+=+=======::=:::
..::::...:I7,,II~,,?III?II?=:=:=:::::,::
:=:,.:,,:=7I7==IIIIIIIII+=::===:,:::,:=
::,.,,:::~?I==??+?I+??II?=::==::::=::===
,.:,:::==~?+?I?+?=?+++??===:=~:::=:=====
.,,:=~==::==I7?????++???I?:~===::====~~:
::~=:==:===?77II=++++?III??+======:===::
:===~===:==II7I7I?I?III?+I??~:::::::::..,
~:=::=:====IIIIIII??I+?+?I?+=::::::=::::
:===:::,::=IIIIIII?III+I++I??+=::=:==:::
:,:,,,:,::=II?III?+?I???+?+++?=====:==~=
~:~=====:~~?I+=?I?=~===?++=?=?====:::~~
====:~.,.,=I7?~~?==~==?=++?==?==~:::====
::,.,,,:::=?I+~~?==~=?+???====~,:,:::~~~
::,::::,:,,?II~:=~::=??===~===:,::===~~~
=:::::::::::?II~:.:,,?I??=~~==~::==~===:~
=:~=:=::==:?I+=:,:,.I7?+==?=,.,:::~:====
===:====:=II??~,,...II??~+.,,:=:==~=::==
===~~==::===?=,,:::??+??:::~============
~=::=~===~====~::~~:+?~:~===~=~~~:~~::::
=:::=====:~==:===:::==:==::=::~::,::~:~~
```

Question 1 – ASCII Art Generation ***(continued)***

Your ASCII art generator must not use global variables and must be organized into functions - in this case, functions for getting user input (with no arguments and returning the loaded image and any other options selected by the user), doing the conversion (with an image as an argument and a two-dimensional list of strings as the return value), and for printing the ASCII art representation of the image to the command prompt.

You may assume that the image selected by the user will never have a width greater than 80, and so the ASCII art you create will never require more than 80 characters per line. You are expected, however, to use at least 10 different characters as colours (recalling that characters like "#" and "@" look white when printed on a black background, and look black when printed on a white background, while the opposite is true for characters like ".").

Since the command prompt equivalent on other operating systems is often black text on a white background (as opposed to the Windows command prompt that has white text on a black background), you must give the user the option to specify whether the background should be black or white. (Remember, if you are trying to create the ASCII art for a white pixel on a black background, you will use a character like "#", but to create the ASCII art for the same pixel on a white background, use a character like ".").

Question 2 – Saving ASCII Art Files

Your solution for this question should be considered an additional component to your solution for Question 1, but each solution must be a separate program so start your solution to Question 2 by copying your solution to Question 1. Write a function that saves an ASCII art image to an external file with a filename specified by the user. The very first line of the file should be the string of characters you use for your different grayscale colours (in order of increasing brightness on a black background - e.g., "`.,-+#`") and then the remaining lines of the file should be the rows of your image.

Question 3 – Manipulating ASCII Art Files

For this question you will create a program that can load the ASCII art files created by your solution to Question 2. This program should ask the user for a filename and will try to load that ASCII art file; loading is a two stage process, because you must first load the string of characters used for the different grayscale colours into a variable, and then you must load the remaining lines as the image itself. Your program must also terminate successfully if the user specifies a filename that doesn't exist (i.e., it must not crash - you are expected to handle the "file not found" exception).

Once you have loaded the file, display it on the command prompt and present the user with a menu of different operations. The operations your program must support should each be programmed in separate functions, and the set of operations must include the ability to:

- flip the image horizontally or vertically,
- rotate the image by 90 degrees clockwise,
- invert the image (i.e., make the ASCII art "negative" of the image), and
- save the new image with the same filename as the original image and then end the program.

After the user selects an operation you must perform the operation and then return to the menu so that the user can apply additional operations.