Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2013

Lab 7 - Developing a Photo Editor

Objectives

To incorporate the filters you developed in Labs 4, 5 and 6 into a photo editor with a text-based user interface.

Attendance/Demo

To receive credit for this lab, you must demonstrate your work. I don't expect every student to finish the photo editor before the end of the lab period. You will receive a SAT grade for this lab if, at demo time, you have finished a reasonable number of the exercises and can demonstrate a partial implementation of the editor; that is, one in which a subset of the commands work.

If you don't finish the photo editor today, finish it on your own time, as a post-lab exercise. Make sure you finish the program before the start of your next lab period.

References

Make sure you understand the three versions of the interactive math quiz program, which are posted on cuLearn in the Week 7 section, before you start this lab.

Overall Design

The program will be structured as three modules: Cimpl.py, filters.py (the module you developed during the previous three labs), and photo_editor.py (which you'll work on during this lab).

Image Manipulation Filters: All of your filter functions must be in filters.py. This module should not contain any code other than these functions. None of the filters should prompt the user to select an image file or display the modified image. The image that a filter will manipulate must be passed to the filter when it is called..

User Interface: All the user interface code must be in photo_editor.py. No image manipulation functions should be stored in this module. A very incomplete implementation of photo_editor.py has been provided on cuLearn. This module contains the definition of function `get_image`, which your code will call to interactively select and load image files for editing.

A complete implementation of the photo_editor.py module should require no more than a page or two of code.

The completed photo editor will provide seven single-letter commands (L, Q, N, G, P, S and E), which are displayed in a menu (by `print` statements) when the editor is ready to accept a new command. All commands will be typed on the keyboard, and will be read by calling Python's `raw_input` function.

The complete user interface will be developed incrementally as you work through the exercises. **Do not**

change the user interface from the specifications that are provided in each exercise. If you change the user interface, your program will be graded UNS (Unsatisfactory).

Getting Started
Step 1: You can use your Lab 5 folder from previous two labs. This folder should contain Cimpl.py, the three image (JPEG) files, and the filters.py module you've been developing over the past few weeks.

Step 2: Download photo_editor.py from cuLearn to this folder.

Step 3: Launch Wing IDE 101 and open photo_editor.py.

Step 4: The script in photo_editor.py prompts the user to interactively select an image file, then loads and displays the image. Read the code in the module and run the script. Make sure you understand how `get_image` works - your code will call this function to load an image for processing.

Exercise 1
You're going to modify the script so that it has two commands: one to load and display an image, and another to quit the program. When the script starts, a menu of commands will be displayed. You will then be prompted to enter a single-letter command. Type the letter L to select and load an image file (call function `get_image` to do this), then display the image. After the command is processed, the menu of commands will be redisplayed and you will be prompted to enter another command. Type the letter Q to exit the program.

The following example provides the detailed specification for the first iteration of the user interface. User input is shown in **boldface** to distinguish it from the output displayed by the program. The *italicized comments* are intended to clarify things, and will not be displayed by your program. **Remember, in Exercises 1 through 8, you must implement the user interface (the menu of commands, command prompt, and error messages) exactly as they are specified by the examples.**

 # A menu of commands is displayed when the program starts.

L)oad image
Q)uit

 # The command prompt is a colon followed by a space; i.e., `': '`

 # Typing the letter L after the prompt allows the user to interactively select an
 # image file using a chooser dialogue box. After the image is loaded, it is displayed.

: **L**

 # The menu of commands and the command prompt are redisplayed.

L)oad image

Q)uit

: **L** *# Choose and display another image*

 # The menu of commands and the command prompt are redisplayed.

L)oad image
Q)uit

: **Q** *# Typing the letter Q causes the program to finish.*

Your program does not have to handle invalid commands; e.g., typing a command other than L or Q at the ":" prompt. (You'll take care of this later.)

Test your program, and fix any problems before moving on to Exercise 2.

Historical note: this user interface is loosely based on the one used by the UCSD p-system (http://en.wikipedia.org/wiki/UCSD_Pascal) in the early days of personal computers. Similar interfaces were used by early versions of some of the most popular programming IDEs for the IBM PC; e.g., Borland's Turbo Pascal.

Exercise 2
Modify your program so that you can type the N command to process an image using your negative filter. Change the menu of commands to look like this:

L)oad image
N)egative
Q)uit

When you type the letter N at the ":" prompt, the image that you are editing will be modified by the negative filter, then the modified image will be displayed.

Here is an example that illustrates an editing session after this exercise has been completed:

L)oad image
N)egative
Q)uit

: **L** *# Choose, load and display an image.*

L)oad image
N)egative
Q)uit

: **N** *# The loaded image is converted to a negative image and displayed.*

L)oad image
N)egative
Q)uit

: **Q** *# The program finishes.*

Your program doesn't have to do anything reasonable if the user attempts to create a negative image before loading an image; i.e. types the letter N before typing L. (You'll take care of this later.)

Test your program, and fix any problems before moving on to Exercise 3.

Exercise 3
Modify your program so that you can type the G command to process an image using your weighted grayscale filter (the one you developed in Lab 6). Change the menu of commands to look like this:

L)oad image
N)egative G)rayscale
Q)uit

Note that the image manipulations should be cumulative. If you load an image and type the letter N, a negative image is created and displayed. If you then type the letter G, the program creates and displays a grayscale version of the negative image, not the image that was originally loaded.

Test your program, and fix any problems before moving on to Exercise 4.

Exercise 4
Modify your program so that you can type the P command to process an image using your posterizing filter. Change the menu of commands to look like this:

L)oad image
N)egative G)rayscale P)osterize
Q)uit

Test your program, and fix any problems before moving on to Exercise 5.

Exercise 5
Modify your program so that you can type the S command to process an image using your sepia-tinting filter. Change the menu of commands to look like this:

L)oad image
N)egative G)rayscale P)osterize S)epia tint
Q)uit

Test your program, and fix any problems before moving on to Exercise 6.

Exercise 6
Modify your program so that you can type the E command to process an image using your improved edge-detection filter (function `detect_edges_better`). Change the menu of commands to look like this:

```
L)oad image
N)egative   G)rayscale   P)osterize   S)epia tint   E)dge detect
Q)uit
```

You will also need to prompt the user to enter the value of the threshold argument for the `detect_edges_better` function:

```
: E
Threshold? : 10.0
```

Test your program, and fix any problems before moving on to Exercise 7.

Exercise 7
Modify your program to display the error message "No image loaded" if you attempt to use one of the filters (the N, G, P, S and E commands) before you've loaded an image into the editor. Here is an example:

```
L)oad image
N)egative   G)rayscale   P)osterize   S)epia tint   E)dge detect
Q)uit

# Here, the user attempts to posterize an image before an image file
# has been loaded, so an error message is displayed.

: P
No image loaded

L)oad image
N)egative   G)rayscale   P)osterize   S)epia tint   E)dge detect
Q)uit

:
```

There are different ways to keep track of whether an image has been loaded. Python has a built-in Boolean type, `bool`, so you could use a Boolean variable which is initialized to `False`, but is changed to `True` the first time an image is loaded. The two Boolean values are `True` and `False` (spelled exactly as shown here). These values are not strings; i.e., the strings `"True"` and `"False"` are not

Boolean values.

Don't use integer variables (with values of 0 and 1) to represent Boolean variables. This is a hack that is often seen in ancient Basic and C code, but should be avoided when using programming languages that provide a Boolean type (Python, Java, modern versions of C and C++, etc.).

Test your program, and fix any problems before moving on to Exercise 8.

Exercise 8
Modify your program to display the error message "No such command" if an invalid command is typed. Note that if an invalid command is entered before an image has been loaded, only "No such command" should be displayed; that is, "No image loaded" should not be displayed at the same time.

Here is an example:

```
L)oad image
N)egative   G)rayscale   P)osterize   S)epia tint   E)dge detect
Q)uit

: A   # There's no command corresponding to the letter "A"
No such command

L)oad image
N)egative   G)rayscale   P)osterize   S)epia tint   E)dge detect
Q)uit

: L   # Choose and display an image

L)oad image
N)egative   G)rayscale   P)osterize   S)epia tint   E)dge detect
Q)uit

: B
No such command
```

You could use an `if-elif-else` statement to check if a command is valid, but if you have several commands, the statement will be long (several `elif`'s). There's an easier way. The expression:

```
cmd in ["L", "Q", "N", "G", "P", "S", "E"]
```

evaluates to `True` if the string bound to `cmd` matches one of the character strings in the list of strings enclosed by `[]`; otherwise it evaluates to `False`. This expression can be used as the condition in an `if` statement or a `while` statement.

Wrap-up

1. Remember to have a TA review your solutions to the exercises and give you a grade (Satisfactory, Marginal or Unsatisfactory) before you leave the lab.

2. You'll need your `filters` and `photo_editor` modules later in the term, so remember to save a copy of your filters.py and photo_editor.py files (copy them to a flash drive, or email copies to yourself, or store them on your M: drive - remember, files left on your desktop or in your Documents folder are not guaranteed to be there the next time you log in).

3. Log in to cuLearn, click the Submit Lab 7 link and submit photo_editor.py. (You don't need to submit filters.py) After you click the Add submission button, drag the file to the File submissions box. After the icon for the file appears in the box, click the Save changes button. At this point, the submission status for your files is "Draft (not submitted)". You can resubmit the file by clicking the Edit my submission button. After you've finished uploading the file, remember to click the Submit assignment button to change the submission status to "Submitted for grading".

Extra Practice
After you have demonstrated your lab work and uploaded your code to cuLearn, feel free to try the following exercises.

Extra Exercise 1 (Easy)
Interesting effects can be achieved when you apply multiple filters, one after the other, so that the modified image produced by one filter is further modified by another filter. For example, load an image file, then select the N, G, P, S and E commands in that order.

Extra Exercise 2 (Easy)
Modify your program so that you can call the other filters you built in earlier labs; e.g., the solarize, black-and-white, black-white-and-gray, extreme contrast, simplify, blur and flip filters.

Extra Exercise 3 (Moderate)
Add a record/playback feature to your program. You will need three new commands: one to start recording, another to stop recording, and a third to playback the recorded commands in the order in which they were recorded..

Hint: one of the easiest ways to do this is to save the recorded commands as a list of strings. (We'll explore Python lists after Break Week.)

Extra Exercise 4 (Difficult)
Both of the textbooks listed in the course outline contain an introduction to building graphical user interfaces (GUIs) with Python's tkinter module. Additional tutorial information about tkinter can be found on the Web. Replace your photo editor's text-based user interface with a GUI.

Posted: Tuesday, October 22, 2013