Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2018

Lab 7 - Developing a Simple User Interface for a Photo Editor

Objective

To incorporate some of the filters presented in lectures into a photo-editing program with a text-based user interface.

Demo/Grading

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 can demonstrate a partial implementation of the editor; that is, one in which a subset of the commands work. At a minimum, you should be able to load an image, call a couple of filters, and quit the editor, using the user interface described in the exercises.

After you finish all the exercises, a TA will review your solutions, ask you to demonstrate some of them, and assign a grade. For those who don't finish early, a TA will grade the work you've completed, starting about 30 minutes before the end of the lab period. **Any unfinished exercises should be treated as "homework"; complete these on your own time, before your next lab.**

Overall Design

The photo-editing program will be structured as three modules: Cimpl.py, lab_7_filters.py (which contains many of the filters that were presented in lectures) and lab_7_ui.py (which contains the user-interface code that you'll develop during this lab).

Image Manipulation Filters: All of your filter functions must be in lab_7_filters.py. This module should not contain any code other than these functions. Don't change the interface of these functions (i.e., function names, parameter lists). None of the filters should prompt the user to select an image file or display the modified image.

User Interface: All the user-interface code must be in lab_7_ui.py. No image manipulation functions should be stored in this module. A very incomplete implementation of lab_7_ui.py has been provided on cuLearn. A complete implementation of the `lab_7_ui` module should require no more than a page or two of code.

You are going to develop the `lab_7_ui` module using a software engineering process known as *iterative*, *incremental* development. Each exercise will guide you through the process of designing, coding and testing one aspect the user interface. As you work through the exercises, you will incrementally add features to the user interface. **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 Unsatisfactory.**

Getting Started

Step 1: Create a new folder named Lab 7. (If you're using a lab computer, you can create the folder on the desktop or in your account's Documents folder, or in the M: drive on the network server.)

Step 2: Download lab_7_filters.py and lab_7_ui.py from the *Lab 7* folder on cuLearn. Move these files to your Lab 7 folder.

Step 3: Download Cimpl.py and the image (JPEG) files from the *Lab Materials* section of cuLearn. Move these files to your Lab 7 folder.

Step 4: Launch Wing IDE 101 and open lab_7_ui.py in an editor window. This module contains the definition of a function named `get_image`, which prompts the user to interactively select an image file, then loads and returns the image for editing.

The script in this module is simple: it calls `get_image`, then displays the loaded 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 images for editing.

When working on Exercises 1-8, feel free to define one or more additional functions in lab_7_ui.py. For example, you could incrementally define a function that just displays a menu of the commands provided by the photo editor. You could define another function that calls the display-menu function, prompts the user to enter a command and returns the command. These are just suggestions - we're leaving it up to you to decide how many functions your program should have and what each of them should do.

Exercise 1

You're going to modify the script so that it displays a menu containing two commands, *L)oad Image* and *Q)uit*. The menu will look like this:[1]

```
L)oad image
Q)uit
```

The script will then display a prompt (a colon followed by a space; i.e., `": "`). This prompts the user to enter a single-letter command:

- Type the letter L, then press the Enter key, 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 the user will be prompted to enter another command.

[1] 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.

- Type the letter Q, then press the Enter key, to quit the program.

The following transcript 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 should 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. Currently, we have
# only two commands

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.
```

To display the menu of commands, call Python's `print` function one or more times. To display the command prompt symbol and read input from the keyboard, call Python's `input` function.

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.

Exercise 2

Modify your program so that the menu contains the *N)egative* command. The menu will now look like this:

```
L)oad image
N)egative
Q)uit
```

When you type the letter N after the prompt, the image you are editing will be passed to the negative filter, then the modified image will be displayed.

Here is a transcript 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    # A negative copy of the loaded image is created 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 the menu contains the *G)rayscale* command. The menu will now look like this:

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

When you type the letter G after the prompt, the image you are editing will be passed to the grayscale filter, then the modified image will be displayed.

Important: the image manipulations must be cumulative. For example, 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 a grayscale version of 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 the menu contains the *2)-tone* command. The menu will now look like this:

```
L)oad image
N)egative   G)rayscale   2)-tone
Q)uit
```

When you type the number 2 after the prompt, the image you are editing will be passed to the black-and-white filter, then the modified image will be displayed.

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

Exercise 5

Modify your program so that the menu contains the *3)-tone* command. The menu will now look like this:

```
L)oad image
N)egative   G)rayscale   2)-tone   3)-tone
Q)uit
```

When you type the number 3 after the prompt, the image you are editing will be passed to the black-and-white-and-gray filter, then the modified image will be displayed.

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

Exercise 6

Modify your program so that the menu contains the *S)olarize* command. The menu will now look like this:

```
L)oad image
N)egative   G)rayscale   S)olarize   2)-tone   3)-tone
Q)uit
```

When you type the letter S after the prompt, the program will prompt the user to enter the value of the threshold argument for the solarizing filter (an integer between 0 and 256), like this:

```
: S
Threshold? (0 - 256) : 128
```

The image you are editing and the threshold will be passed to the solarizing filter, then the modified image will be displayed.

Your program does not have to check if the threshold values entered by the user are in the range 0 through 256, inclusive. (See the "Extra Practice" exercise at the end of this handout.)

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, S, 2 and 3 commands) before you've loaded an image into the editor. Here is an example:

```
L)oad image
N)egative   G)rayscale   S)olarize   2)-tone   3)-tone
Q)uit

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

: S
No image loaded

L)oad image
N)egative   G)rayscale   S)olarize   2)-tone   3)-tone
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`, to indicate that no image has been loaded. This variable will be assigned `True` the first time an image is loaded. The two Boolean values are `True` and `False` (spelled exactly as shown here). These values are <u>not</u> strings; i.e., the strings `"True"` and `"False"` are <u>not</u> 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, Go, 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 <u>before</u> an image has been loaded, only "No such command" should be displayed; that is, "No image loaded" should <u>not</u> be displayed at the same time.

Here is an example:

```
L)oad image
N)egative   G)rayscale   S)olarize   2)-tone   3)-tone
Q)uit

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

L)oad image
N)egative   G)rayscale   S)olarize   2)-tone   3)-tone
Q)uit

: L   # Choose and display an image

L)oad image
N)egative   G)rayscale   S)olarize   2)-tone   3)-tone
Q)uit

: B
No such command
```

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

```
    cmd in ["L", "Q", "N", "G", "S", "2", "3"]
```

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; e.g.,

```
    if cmd in ["L", "Q", "N", "G", "S", "2", "3"]:
```

Test this feature and fix any problems.

You should also verify that adding this feature to the user interface didn't "break" a feature that previously worked. Retest all of the commands, as well as the "no image loaded" scenario, before moving on to Exercise 9.

Exercise 9

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. Try different combinations of the N, G, S, 2 and 3 commands, or use the same set of commands, but change the order in which they are invoked.

Wrap-up

1. Remember to have a TA review your solutions to Exercises 1-8, assign a grade (Satisfactory, Marginal or Unsatisfactory) and have you initial the grading/sign-out sheet.
2. Remember to backup your Lab 7 folder before you leave the lab; for example, copy it to a flash drive and/or a cloud-based file storage service.
3. Any unfinished exercises should be treated as "homework"; complete these on your own time, before Lab 8.

Extra Practice

Modify your solution to Exercise 6 so that the program checks if the solarizing filter's threshold values are in the range 0 through 256, inclusive. If the user enters an out-of-range threshold, the program should display an error message and prompt the user to enter another threshold value. This should be repeated until a valid threshold has been entered.