Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2017

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

Objective

To incorporate some of the filters you developed in Labs 5 and 6 into a photo editor 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.

When you have finished all the exercises, call a TA, who 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, filters.py (the module you developed during the previous two 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. Don't change the interface of these functions (i.e., function names, parameter lists) from what was specified in Labs 5 and 6. 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 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, X, S and E), which are displayed in a menu when the editor is ready to accept a new command. All commands will be typed on the keyboard.

You are going to develop the `photo_editor` 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.

Step 2: Download photo_editor.py, Cimpl.py and the image (JPEG) files from cuLearn to your Lab 7 folder. Copy the filters.py file that contains your solutions to the Lab 5 and 6 exercises to your Lab 7 folder.

Step 3: Launch Wing IDE 101 and open photo_editor.py in an editor window.

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.

When working on Exercises 1-8, feel free to define one or more additional functions in photo_editor.py. For example, as you work on Exercises 1-6, you could incrementally define a function that just displays the menu of commands. 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. 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 quit the program.[1]

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

[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.

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 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 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    # 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. 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 X command to process an image using your extreme-contrast filter. Change the menu of commands to look like this:

```
L)oad image
N)egative   G)rayscale   X)treme contrast
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   X)treme contrast   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   X)treme contrast   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, X, S and E commands) before you've loaded an image into the editor. Here is an example:

```
L)oad image
N)egative   G)rayscale   X)treme contrast   S)epia tint   E)dge detect
Q)uit

# Here, the user attempts to sepia-tint 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   X)treme contrast   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`, 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 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, 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 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   X)treme contrast   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   X)treme contrast   S)epia tint   E)dge detect
Q)uit

: L   # Choose and display an image

L)oad image
N)egative   G)rayscale   X)treme contrast   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 (it will have several `elif` clauses). There's an easier way. The expression:

```
    cmd in ["L", "Q", "N", "G", "X", "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.

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 seven image processing commands, as well as the "no image loaded" scenario.

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, X, S and E 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. You'll be using your `photo_editor` and `filters` modules later in the term, so remember to backup your project folder before you leave the lab; for example, copy it to a flash drive and/or a cloud-based file storage service.