CSC108H/A08H Introductory Lab

Congratulations on changing your password. Now it's time for you to do some programming. To earn your lab marks, you must actively participate in the lab and make significant progress.

At the end of the lab, please return this handout to your TA so that we can reuse it and save some trees. We will post the handout on the course website at the end of the week. If you don't finish parts of this lab, you should download the handout and work through it. Come to office hours if you're feeling lost!

1 Objectives

- Learn how to program in pairs as *driver* and *navigator*.
- Become more familiar with Python by tracing and writing code for image manipulation.
- Practice driving and navigating.

2 Driver and Navigator

Throughout the term, we will use the terms *driver* and *navigator*. Here are the definitions of the two roles:

driver: The person typing at the keyboard.

navigator: The person watching for mistakes, and thinking ahead.

Here is the most important rule for this and all future labs:

> **The navigator must not touch the keyboard or mouse.** The driver's role is to work with the computer, and the navigator's is to think about language issues and upcoming issues related to the problem being solved. If the navigator interferes with the driver, the group loses the view of what is coming up *and* may make the problem harder to solve *and* makes it harder for the driver to learn the material.[1]

In every lab handout, we'll call you two `s1` and `s2`, and `s1` will be the first driver.

3 Image Manipulation

For many of you, this will be the first time that you have programmed, which is exciting, but it may also be a bit overwhelming. Don't hesitate to ask your TA or your partner (or your other labmates!) for help. You may also refer to your lecture notes and textbook.

To begin, visit the Labs page of the course website (`http://www.cdf.toronto.edu/~csc108h/winter`) and download `image_functions.py`, `trace.py`, and any image files to your home directory, `/../../c2______`. Your TA will also give you a handout with the program `trace.py` on it.

You will recall from lecture that one can trace the execution of a program, step by step, and examine the values of the variables as the program executes. We will start this week's lab by having you do this on the program `trace.py`.

Complete the following tasks:

1. Double-click the Wing icon. It should open several panes; on top is the editor/tracer, and at the bottom is the Python shell. The shell looks something like this:

[1] Analogously, if you're driving with a friend and looking at the map, do you think it's wise to reach over and turn the steering wheel when it's time to turn the corner? Especially if they're just learning to drive?

```
Python 2.7.2 (v2.7.2:8527427914a2, Jun 11 2011, 14:13:39)
[GCC 4.0.1 (Apple Inc. build 5493)]
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2. Open file `trace.py`, which you just downloaded. Do this by pulling down the **File** menu in the upper left-hand corner of Wing and selecting **Open**, which puts the text of the file in the editor window.
   - Click the **Step Into** button. This will highlight the first line of the file in the editor window and it will start the step-by-step tracing or debugging process.
   - In the lower left-hand area of Wing, click on the **Stack Data** tab. This is where the values of the variables are displayed. For now, you can ignore names that are surrounded by underscores (`__`). You will notice that variables `x` and `y` are not currently listed. Why not? (If you don't know, ask your partner, and if your partner doesn't know, ask the TA.)
   - The handout "Intro Lab Tracing Table" has the code and boxes in which to write the values of the variables. Discuss with your partner what values the variables will have as the program executes. If you can, write down their values. Some may be unpredictable. Why?
   - Show your guesses to your TA.

     Switch roles: `s2` drives and `s1` navigates.
   - For the rest of this exercise, you will use the **Step Over** button (instead of **Step Into**, as above). Each time after pressing this button, check the contents of the **Stack Data** window.
   - As you keep stepping over, record the values for the variables in your boxes and see if they match your predictions. If they don't, discuss with your partner to determine why not.

   Switch roles: `s1` drives and `s2` navigates.
3. Open file `image_functions.py`, which you just downloaded, and fill in the body of function `get_picture` so that it does what the docstring says it should.
4. In the shell, call function `get_picture()` and store the value returned in a variable called `pic`. To display the picture, call `media.show(pic)`.
5. Call function `maximize_red`, passing `pic` as an argument, and store the value returned in a variable called `new_pic`. You should probably choose a small picture so that it won't take too long to run. We have provided this picture for you to work with:

   `http://www.cdf.toronto.edu/~csc108h/winter/labs/w2/BahenSmall.jpg`

   Now call `media.show(new_pic)` to see the modified picture.

   Switch roles: `s2` drives and `s1` navigates.
6. Add to `image_functions.py`: define a function called `remove_blue` that takes a picture as a parameter. It should create a copy of the picture and, working with the copy, set each pixel's blue value to 0.
7. In the Python shell, call `remove_blue` with `pic` as an argument, and store the value returned in a variable called `new_pic`. If you have not clicked the Run button since defining `remove_blue`, then you will not be able to execute it: the shell will not know that it exists. If that is the case, click Run. You'll notice that the shell is reset, which means you'll have to start over and reopen the picture before calling `remove_blue`. After you call `remove_blue`, call `media.show(new_pic)` to display the picture.
8. Add to `image_functions.py`: define a function called `halve_green` that takes a picture as a parameter. This function should create a copy of the picture and, working with the copy, set each pixel's green value to half of the old green value.
9. In the shell, call the function on your picture and then call `show` to see the result.

   Switch roles: `s1` drives, and `s2` navigates.
10. Add to `image_functions.py`: define a function called `swap_red_blue` that takes a picture as a parameter. This function should create a copy of the picture and, working with the copy, swap each pixel's red and blue values. For example, if a pixel's red and blue values are 117 and 38, respectively, then the red and blue values should be set to 38 and 117, respectively.
11. In the shell, call the function on your picture and then call `media.show` to see the result.
12. If time permits, try calling the functions that you have written on your picture in different combinations to see how you can modify the image.