CSC108H/A08 Booleans Lab

1 Objectives

- Write `if` statements to manipulate images
- Practice submitting an assignment
- Use type `bool`

2 `if` statements and image manipulation

In last week's lab, you wrote several functions to manipulate images. Those functions had something in common: they performed the same action on *every* pixel in the image. This week you will manipulate images, but you will not necessarily perform the same action (or any action) on each pixel. Hints:

- You will need to use `if` statements to complete this section.
- In the last two functions, use `media` functions `get_color` and `set_color` when appropriate.

Download `booleans.py` from the course website, and add the following functions to it:

`reduce_blue` `(Picture) -> Picture`	Return a new picture that is a copy of the given picture, in which each pixel with a blue component greater than 100 has its blue component set to zero. Hint: see function `maximize_red` from `booleans.py`.
`black_to_red` `(Picture) -> Picture`	Return a new picture that is a copy of the given picture, in which each pixel that is currently black is set to red.
`odd_y_to_purple` `(Picture) -> Picture`	Return a new picture that is a copy of the given picture, in which each pixel that has an odd Y coordinate is set to purple. Note: use `help` to learn about `media.get_y`.
`grey_posterize` `(Picture) -> Picture`	Return a new picture that is a copy of the given picture, in which each pixel is set to black if its intensity is less than 128, and to white otherwise. Note: The intensity of a pixel is the average of its red, green, and blue color components.

`booleans.py` includes a function `make_test_pic` that creates a *very small* picture for you to use during debugging. Don't try to debug a larger picture; you'll have to step through your code for a lot of pixels!

Once you have completed all four functions, demonstrate them to your TA (if he or she isn't swamped with questions) and move on to the next section.

3   Practice submitting an assignment with a partner

As you know, exercises and assignments in this course will be submitted electronically using the MarkUs submission system. Visit the Submitting Coursework page on the course website to find a link to MarkUs.

To help you learn how to submit **with a partner**, you are going to declare your partnership to MarkUs and then submit your file `booleans.py` as if it were an assignment. Here's what to do:

1. **Log in:** One of you should go to the MarkUs website and log in, using your cdf account name and password.

2. **Go to the assignment:** Once you log in, you will see a list of the course assignments. So far, that includes only the fake assignment called LabPractice. Click on LabPractice to see a page of information about the assignment. It includes the fact that you don't have a "group" (in this case just a partner) yet.

3. **Invite your partner:** Under Group Information, click Create to indicate that you want to create a group rather than work alone. Then click Invite. In the box that pops up, type your partner's cdf username. You have now invited them to be your official partner (just for this LabPractice exercise).

4. **Switch users:** Now log out (look in the upper-right corner of the window) and let your partner log in.

5. **Accept the invitation:** Now that you, the second student in the pair, are logged in, click on assignment LabPractice and your personal version of the assignment information will appear. It includes the fact that you have been invited to be a partner. Under Group Information, click Join to accept the invitation.

6. **Submit work:** Now the two of you are ready to submit your work. Click on the submissions tab (either partner can do this). You'll see that you are "required" to submit booleans.py. Click Add a New File, click Browse to choose it, and then click Submit. You're done!

You can check that you submitted the right file by clicking on a file name in MarkUs and looking at its contents.

This is the only time that you will need to submit your lab work electronically and what you submit will not be marked; this is just for practice submitting. Also, the partnership that you declare will not carry over to the real assignments. If you want to work with the same person again you can, but you will declare that separately for each assignment.

Switch roles: `s1` drives and `s2` navigates: `s1` should also practice submitting `booleans.py`.

4 Type `bool` and the Poisonous Potions Problem

Boolean logic employs two values, `True` and `False`, and three basic Boolean operators, `and`, `or`, and `not`. If `a` and `b` are Boolean variables:

`a and b`	Evaluates to `True` if <u>**both**</u> `a` and `b` are `True`, and `False` otherwise.
`a or b`	Evaluates to `True` if <u>**at least one of**</u> `a` and `b` are `True`, and `False` otherwise.
`not a`	Evaluates to the negated value of `a`; `True` if `a` is `False`, and `False` if `a` is `True`.

Similarly, "`a and b and c`" will evaluate to `True` only if all three variables have the value `True` and "`a or b or c`" will be `True` if any one or more of the three variables is `True`.

This part of your lab is about a logic puzzle involving four potions. We use four boolean variables, `p1` to `p4`, to represent the potions; a `True` value indicates that the potion is poisonous, and a `False` value indicates that it is not.

One of the potions is poisonous. We won't tell you which, but here are five hints:

1. At least one of the four potions is poisonous.
2. `p2` is not poisonous.
3. At least one of `p1` and `p3` is poisonous.
4. At least one of `p3` and `p4` is not poisonous.
5. Exactly one of the potions `p1`, `p3`, and `p4` is poisonous.

The potions puzzle can be represented in Python. Download `puzzle.py` from the Labs page of the course website. The purpose of this program is to test the user's hypothesis about the answer to the puzzle. Look at the main block. For each potion, it asks the user whether they think it is poisonous or not. Then it calls function `solution` to find out if the user is correct, that is, to find out if their hypothesis satisfies all 5 hints.

Function `solution` uses five "helper" functions, `hint1` to `hint5`. Each one checks whether the hypothesis satisfies that one hint. For example, if you call `hint1(True, True, False, True)`, you are testing the guess that `p1`, `p2`, and `p4` are poisonous, and `p3` is not. This satisfies hint 1, so function `hint1` will return `True`.

Function `hint1` has already been written, but the others have not. (As a result the program will crash if you run it.) Write the other four functions according to the descriptions in the table below, using `hint1` as an example of what the code should look like. Add the functions to `puzzle.py`, **switching driver and navigator after each one**.

`hint1(p1, p2, p3, p4)`	Return `True` if one or more of the potions is poisonous and `False` otherwise.
`hint2(p2)`	Return `True` if potion `p2` is not poisonous, and `False` otherwise.
`hint3(p1, p3)`	Return `True` if least one of potions `p1` and `p3` is poisonous, and `False` otherwise.
`hint4(p3, p4)`	Return `True` if at least one of potions `p3` and `p4` is not poisonous, and `False` otherwise.
`hint5(p1, p3, p4)`	Return `True` if exactly one of the given potions is poisonous, and `False` otherwise.

Once you have added the rest of the hint functions, run the program and try to solve the puzzle. If the program gives you the "sorry" message, either your solution really is incorrect or one of the `hint` functions is wrong! Run the program again, but use the debugger this time to verify that each `hint` function works as you expect. Hint: `solution` has been written so that the result of each `hint` is saved in a variable that the debugger will let you inspect.

Once you have found the solution, show your TA your working puzzle code. See you in class!