Homework 2

Due 4:00pm, Wednesday, February 13, 2019

Silly Premise

NASA is hiring one additional astronaut for their DISCOVER mission to Pluto. Who will it be? The top two candidates are Ginger the Golden Retriever puppy and Billy the human baby. To display their astronaut skills, Ginger and Billy need to show NASA how much they know about loops. Loops are very important in space! Satellites move in loops and so do planets! Your mission is to help both Ginger and Billy show off their loop skills (because secretly you are both team puppy and team baby?). Mysterious!!

Installation and Handin

Homework Setup. Copy the support files to your home directory by using the cs4_install command in a Brown CS Terminal window. For this homework type

```
cs4_install hw02
```

There should now be a `hw02` folder within your homeworks directory. Using Terminal, you can move into the `hw02` folder with the cd command:

```
cd ~/course/cs0040/homeworks/hw02
```

Homework hand-in. Be sure to turn in all the files requested and that they are named exactly as specified, including spelling and case. When you're ready to submit the files, run:

```
cs4_handin hw02
```

from a Brown CS Terminal window from your `~/course/cs0040/homeworks/hw02` directory. The entire contents of your `~/course/cs0040/homeworks/hw02` directory will be handed in. Check for a confirmation email to ensure that your assignment was correctly submitted using the `cs4_handin` command. You can re-submit this assignment using the `cs4_handin` command at any time and only your most recent submission with be graded.

Part I: Understanding Loops (10)

Ginger is a little behind on his understanding of loops because he wasn't quite paying attention in class:(That's ok though, you are here to help!
Put your answers for this problem in the `hw02_1.py` file.

Problem 2.1a) Tracing Mystery

Refer to the `mystery()` function in the `hw02_1.py` file, which uses an *index-based* for loop.

Trace the execution of the following call to this function:

```
mystery([4, 7, 3, 5, 8, 1])
```

In particular, you should complete the table below the code to illustrate how the expression at the top of each column changes during the execution of the loop.The table begins with a row for the initial value of the variable `count`. The remaining rows (including ones that you should add as needed) should show what happens for each value of the loop variable `i`. **State the return value** of this function call.

Problem 2.1b) Nested Loops

Consider the `nested_loops()` function in the `hw02_1.py` file, which uses a nested loop. Complete the table below the function (adding rows as needed) to show how the variables change over time and the values that are printed.

We completed similar traces in lecture, so those materials may be helpful for this exercise.

Problem 2.1c) While Loops

Consider the `while_loops()` function in the `hw02_1.py` file, which uses a while loop. Complete the table below the function (adding rows as needed) to show how the variables change over time and the values that are printed:

Part II: Processing Sequences with Loops (30)

Now that you've helped Ginger, it's time to help Billy the baby. Billy is pretty confident about loops. He likes loops a lot and did some extra research on them. He wants to show NASA that he is amazing! Please help him show off his cooliosis osmosis tricks!!!

Put your answers for this problem in the file named `hw02_2.py`.

Remember to design your code using tests. This means writing some test cases *first*, before you start writing your function, so that you can see whether it works as you work on it. The examples in this document can be a good starting point for your tests, so feel free to use some of them, but they are NOT extensive, so remember to write your own!

Problem 2.2a) Double String Redux

Write a function `double(s)` that takes an arbitrary string s as input, and that ***uses a loop*** to construct and return the string formed by doubling each character in the string. Here are three examples:a

```
>>> double('hello')
'hheelllloo'
>>> double('python')
'ppyytthhoonn'
>>> double('')
''
```

Hint: This function performs a *cumulative computation* that gradually builds up a string. We discussed a similar function (a loop-based `remove_vowels`) in lecture.

Problem 2.2b) Weave Redux

Write a function `weave(s1, s2)` that takes as inputs two strings `s1` and `s2` and uses a loop to construct and return a new string that is formed by “weaving” together the characters in the strings `s1` and `s2` to create a single string. In other words, the new string should alternate characters from the two strings: the first character from `s1`, followed by the first character from `s2`, followed by the second character from `s1`, followed by the second character from `s2`, etc. If one of the strings is longer than the other, its “extra” characters – the ones with no counterparts in the shorter string – should appear immediately after the “woven” characters (if any) in the new string. For example:

```
>>> weave('aaaaaa', 'bbbbbb')
'abababababab'
>>> weave('abcde', 'VWXYZ')
'aVbWcXdYeZ'
>>> weave('aaaaaa', 'bb')    # four extra 'a' characters at the end
'ababaaaa'
>>> weave('aaaa', 'bbbbbb')  # two extra 'b' characters at the end
'ababababbb'
>>> weave('aaaa', '')        # all of the 'a' characters are extra!
'aaaa'
>>> weave('', 'bbbb')        # all of the 'b' characters are extra!
'bbbb'
>>> weave('', '')
''
```

Note: You will need to use an *index-based loop* so that you can access the corresponding characters from both `s1` and `s2` at the same time. See the template on the stencil code.

Note that on the stencil code, we determine the length of the shorter string before beginning the loop, because the loop should only consider the index values that are present in both loops.

After the loop, you will need to handle any “extra” characters from the longer string (for cases in which the strings don’t have the same length). One way to do that is to use conditional execution (e.g., an `if-else` statement), although other approaches may also be possible.

Problem 2.2c) Find Index Redux

Write a function `index(elem, seq)` that takes as inputs an element `elem` and a sequence `seq`, and that *uses a loop* to find and return the index of the first occurrence of `elem` in `seq`. The sequence `seq` can be either a list or string. If `seq` is a string, `elem` will be a single character. `seq` can also be a list containing elements of any datatype (e.g., int, string, etc.), in which case `elem` will also be of the same datatype. Don't forget that the index of the first element in a sequence is 0.

Notes:

- If `elem` is not an element of `seq`, the function should return -1.
- You may *not* use the `in` operator in this function to test for the presence of `elem` in `seq`. (However, you *are* welcome to use `in` as part of the header of a for loop.)
- You may *not* use any built-in *methods* (i.e., functions like `find` or `index` that are found inside of string or list objects). However, you *may* use built-in *functions* like `len` and `range`.

Here are some examples:

```
>>> index(5, [4, 10, 8, 5, 3, 5])
3
>>> index('hi', ['well', 'hi', 'there'])
1
>>> index('b', 'banana')
0
>>> index('a', 'banana')
1
>>> print(index('n', 'banana'))
2
>>> index('i', 'team')
-1
>>> index(8, [4, 10, 5, 3])
-1
```

Hint: One way to solve this problem is to have two return statements: one inside the loop, and one after the loop has completed.

Part III: Image Processing with Loops (30)

Ginger overheard that Billy was doing some cool tricks with loops, so he wants to do something even COOLER! He decides that image processing is the answer, but he doesn't know much about this topic. Please help poor Ginger because Ginger is the best.

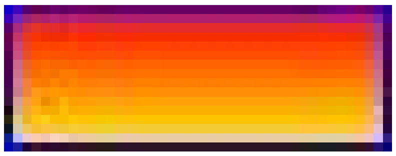
Put your answers for this problem in the file named `hw02_3.py`.

In lecture, we considered a module that is able to process images stored in PNG format. We saw that images are composed of pixels, and that the color of each pixel can be represented by a list of RGB values–three integers between 0 and 255 that represent the amount of red, green, and blue that is present in that pixel.

We have provided some PNG images that you can use when testing your functions. They include the following:

`Ginger.png`:

`test.png`:



`spam.png`:

Important: Two of these images have a one-pixel border: `test.png` has a blue border, and `spam.png` has a red one. When you create a new image that is based on one of these images, these borders should help you to ensure that you are transforming the entire image.

In your file `hw02_3.py`, we've given you the following example of a function that processes a PNG image:

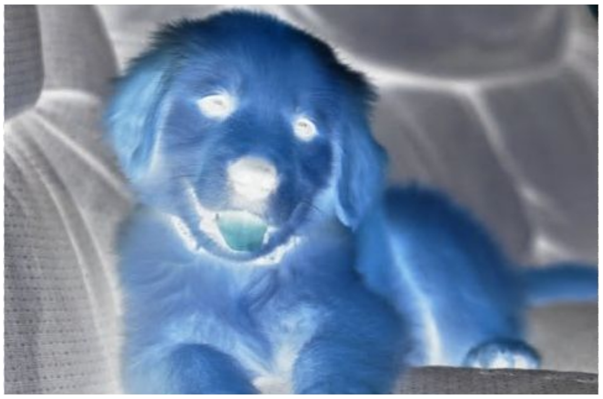
```
def invert(filename):
    """ loads a PNG image from the file with the specified filename,
        and creates a new image in which the colors of the pixels are
        inverted.
    """
    ...

invert('ginger.png') # Run the examples
invert('test.png')
```

To test it, run `hw02_3.py`. If all goes well, you should see the following message:

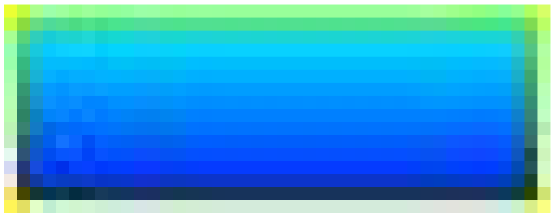
```
invert_ginger.png saved
invert_test.png saved.
```

Now if you look in your `hw02` folder, you should now see files named `invert_ginger.png` and `invert_test.png` (although the .png extension may or may not be visible). Double-clicking on those files should show you the following images:

`Invert_ginger.png`:



`invert_test.png`:



Support Functions and Image Methods

The `invert()` function illustrates most of the key features of the image-processing module that we've given you. The function begins as follows:

```
img = load_image(filename)
```

This line uses the `load_image()` function, which takes a string representing the name of a PNG image file, loads that image from disk, and creates a representation of the image that we can manipulate in our code. After this line of code executes, the variable `img` represents the whole image. More precisely, the variable `img` represents an image *object*.

As discussed in lecture and section, one important property of objects is that they carry around functions inside themselves that we can call to obtain information about the object or to modify its contents. Functions that belong to an object are called *methods*, and we call them using the dot . operator.

Here is a reminder of the key methods that are available inside every image object:

method name	what it does
`img.get_width()`	returns the width of the image represented by the image object img
`img.get_height()`	returns the height of the image represented by the image object img
`img.get_pixel(r, c)`	returns the list of RBG values for the pixel at row r and column c of the image represented by the image object img
`img.set_pixel(r, c, rgb)`	changes the RBG values values for the pixel at row r and column c of the image represented by the image object img to the list of RBG values given by rgb
`img.save(filename)`	saves the image represented by the image object img in a file with the specified filename

These methods will be helpful for you to find and manipulate information about the images you're going to modify with your functions.

Testing

Testing that the output of your functions is correct is tricky, so we've provided you with helper functions for testing your implementations in your `hw02_3.py` file. The function `test_image` generates sample images which you can use in your tests, `image_equal` tests the equality of two images, and `test_invert` is an example for what a test function should look like in the context of png images.

Problem 2.3a) Black and White

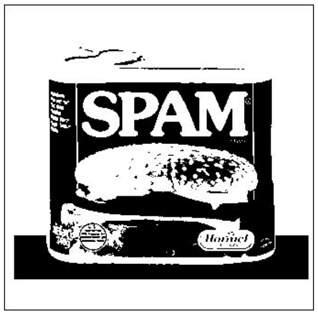
Fill in the function `bw(filename, threshold)` that loads the PNG image file with the specified filename and creates a new image that is a pure black and pure white version of the original image. The second input to the function is an integer threshold between 0 and 255, and it should govern which pixels are turned white and which are turned black.

The brightness of a pixel [r,b,g] can be computed using this formula:

```
.21r + .72g + .7b
```

We've given you a helper function `brightness` that you should use to make this computation. If a pixel's brightness (as determined by the brightness helper function) is greater than the specified threshold, the pixel should be turned white `([255,255,255])`; otherwise, the pixel should be turned black `([0,0,0])`.

For example, here is a black-and-white version of `spam.png` that was created using a threshold of 100:



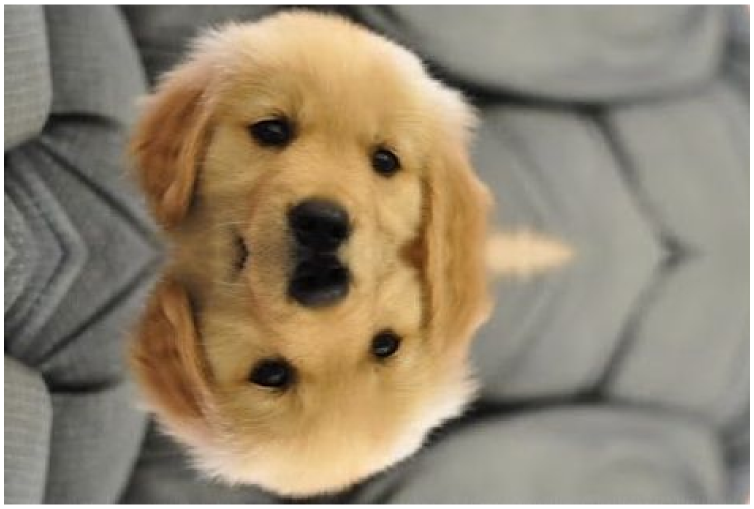
The new image file should have the name `bw_`*`filename`*, where *`filename`* is the name of the original file. For example, when you turn `spam.png` into a black-and-white image, the resulting file should be named `bw_spam.png`.

Notes:

- **Include test cases for your bw function.** Refer to test_invert() for how to do this. **You only need to write test cases for the 1x1, 2x2, and 3x3 cases.**
- Don't forget to take advantage of the brightness function that we've given you.
  Here again, you should include a docstring and any other comments that might be necessary to explain your code.
- More generally, you should aim to make your code easy to read. For example, you should choose descriptive names for your variables. In addition, we encourage you to leave a blank line between logical parts of your function.

Problem 2.3b) Mirror

Fill in the function `mirror_vert(filename)` that loads the PNG image file with the specified filename and creates a new image in which the original image is "mirrored" vertically. In other words, the bottom half of the pixels in each column should be replaced by the reversed top half of the pixels from the same column. For example, here is `ginger.png` mirrored vertically:



The new image file should have the name `mirrorv_`*`filename`*, where *`filename`* is the name of the original file. For example, when `spam.png` is mirrored vertically, the resulting filename should be called `mirrorv_spam.png`.

Notes:

- **You do not need to include test cases for your mirror function. However, you should still see if your code is working by calling your function on our example images, and looking at the images to see if they are properly mirrored.**
- You'll need to adjust the number of iterations performed by at least one of your two nested loops.
- When computing the appropriate coordinates for a flipped pixel, don't forget that valid `(r,c)` coordinates for an image of height h and width w are the following:
  $0 \leq r \leq h - 1$
  $0 \leq c \leq w - 1$
- Given these ranges, where should a pixel from the topmost row of a given column be mirrored on the bottom half? Where should a pixel from the second row of a given column be mirrored? Where should a pixel from the rth row of the original image be mirrored? Think about these questions when implementing your solution for this assignment.

Part IV: TT Securities (30)

For the final test, NASA is giving Ginger and Billy a difficult task--they must help an investment firm called Time Travel Securities, Inc. (also known as TT Securities or TTS). This is because the DISCOVER Pluto mission is actually a secret mission to test some new time travelling technologies. NASA wants them to write a program that will use loops to process a list of stock prices. Show off your loop skills and help Ginger and Billy become astronauts!

As always, put your answers for this problem in the file named `hw02_4.py`. We mentioned this application in lecture and look at an example user-interaction loop. In `hw02_4.py`, we've also given you some starter code that is based on the code from lecture.

You may not use the built-in `sum`, `min`, or `max` functions for this problem. Instead, you should use loops of your own construction to compute the necessary values.

Support Code Overview

The starter code that we've given you includes a function called `tts()` that serves as the "main" function–the one that handles the user interactions. It first calls `display_menu()` to print a menu of options for the user. Then it asks for user input by using the `input` function (see line 50). Depending on what the user inputs, the code will call the corresponding function. Your job is to fill in these corresponding functions. To run the program, you should call the function `tts()`, which we have done for you in line 214.

Required Functionality

Your final program should present the user with the following menu of choices:

```
(0) Input a new list of prices
(1) Print the current prices
(2) Find the latest price
(3) Find the average price
(4) Find the standard deviation
(5) Find the min and its day
(6) Find the max and its day
(7) Test a threshold
(8) Your TT investment plan
(9) Quit

Enter your choice:
```

Our starter code includes support for options 0, 1, 2, and 9. Read over the code that we've given you, and make sure that you understand how the various functions work and fit together.

- When inputting a new list of prices, please input them as numbers separated by a space. For example, enter the following when inputting the price list `[40, 10, 20, 35, 25]`:

```
Enter a new list of prices: 40 10 20 35 25
```

For this exercise, we're asking you to change our code for option 1 and implement options 3 through 8, which will be broken down into sections below. **Please remember to write tests for the functions you write for each option, as well as any helpers you write for those functions.** For your functions and their tests, you can assume that you will only be dealing with non-empty lists of floats.

Problem 2.4a) Option 1: Print the Current Prices

Revise the function `print_prices()` that implements Option 1 so that it uses a loop to print a table of days and prices in the format shown below:

```
Day Price
--- -----
 0  20.00
 1  10.00
 2  30.00
```

Notes:

- Recall that a price's day number is simply its index in the list of prices.
- To print neatly formatted columns, you can use print statements with a formatting string. To understand how this works, we encourage you try the following examples from the Shell:

```
>>> val = 12
>>> print('%7.2f' % val)
  12.00
>>> print('%9.2f' % val)
    12.00
>>> print('%5.0f' % val)
   12
```

- In each case, the first number in the formatting string specifies the width of the column in which val should be printed, and the second number specifies the number of digits that

should appear after the decimal when the number is printed. The printed number is aligned with the right-hand side of the specified column. For instance:
    - In the first example above, the specified column width is 7, which is why there are two spaces before the number. Those two spaces, plus the five characters in 12.00, add up to a total column width of 7.
    - In the second example above, the specified column width is 9, so there are four spaces before the printed number.
    - In the final example above, the column width is 5, but the 0 after the decimal in the formatting string means that the number is printed as an integer with no decimal. Thus, we end up with three spaces before the 12. Those three spaces, plus the two characters in 12, add up to a total column width of 5.
- In the context of the format shown above, think about what the column size is for the Price column, as well as how many decimal places you should round to.
- You **do NOT** need to write a test function for this part of the assignment, but you will need to for all of the rest of the functions you write.

Problem 2.4b) Options 3-6: Statistics

Add support for options 3-6 so that they find and print the appropriate statistics about the list of prices. Your job is to fill in code for the following functions: `average()`, `standard_deviation()`, `argmin()`, and `argmax()`, as well as the functions for their test cases. You may write any additional helper functions. See below for some important guidelines.

Notes:

- You must use separate helper function(s) for each of these options. Look at the code under the provided `tts()` function to see how these helper functions are used (i.e. your helper functions return values which the tts function then prints).
- Don't forget that you cannot use the built-in sum, min, or max functions. Instead, you should use loops of your own construction to compute the necessary values.
- For options 5 and 6, your helper functions only need to return the day on which the min/max occurs (i.e. the argmin and argmax). Look at `tts()` (lines 73-78) to see how this value is used to report the day on which the min/max occurs *and* the min or max price.
    - Remember that the day numbers are 0 indexed, so the first element in the list of prices corresponds to day 0.

Problem 2.4c) Option 7: Test a Threshold

Add support for Option 7 *(Test a threshold)*. Look at `tts()` to see how our support code asks the user to enter a threshold:

```
threshold = float(input('Enter the threshold: '))
```

Your job is to update the `check_threshold(prices, threshold)` function, which takes in the list of prices, `prices`, as well as the threshold value, `threshold`, from the code above. Use a loop to determine if there are any prices above that threshold, and return a boolean (True or False) -- True if there are prices above that threshold, False if there are not.

For example, if the list of prices is `[10,30,20]`, the following user interaction should take place:

```
Enter your choice: 7

Enter the threshold: 25
There is at least one price over 25
```

And here's another user interaction for the same list of prices:

```
Enter your choice: 7

Enter the threshold: 40
There are no prices over 40.
```

As long as the `check_threshold` function correctly returns True or False, these messages should be correctly displayed. See our support code under `tts()` to see how these messages are displayed (lines 79-85). You may also write any additional helper functions.

Notes:

- Remember when implementing this helper function that all that matters is whether *any* values of the list are over the threshold. You do not need to print out exactly how many values there are over the threshold.
- Make sure to write tests cases for this under `test_check_threshold()`

Problem 2.4d) Option 8: Time-Travel Investment Plan

Add support for option 8 so that it implements the "time travel" investment strategy discussed in lecture under the `tt_strategy()` function. It should use loops to find the best days on which to buy and sell the stock whose prices are given in the list of prices. The buy and sell days that you determine should maximize the profit earned, but the sell day must be greater than or equal to the buy day. In other words, you cannot sell a stock before you have bought the stock. Refer to the docstring in the support code under the `tt_strategy()` function to see the right format in which to return these values. You may write any additional helper functions.

For example, if the list of prices is `[40, 10, 20, 35, 25]`, the output of this option should look like this:

```
Buy on day 1 at price 10
Sell on day 3 at price 35
Total profit: 25
```

Notes:

- You may find it helpful to adapt the example diff function that will be covered in lecture.
- Once again, you must fill in the helper function for the option, `tt_strategy()`, as well as the test cases, `test_tt_strategy()`
- Look at `tts()` lines 86-99 to see how the values obtained in `tt_strategy()` are used to print the message to the user. Also look at lines 99 to see how an appropriate message is displayed when no investment opportunities are available (for example in a flat or falling market).

Congrats on finishing! Thank you for helping Ginger and Billy. Hopefully they both get chosen as astronauts :)

Please let us know if you find any mistakes, inconsistencies, or confusing language in this document or have any concerns about this and any other CS4 document by posting on Piazza or filling out our anonymous feedback form.