

## COMP 690 Data Fusion Fall 2008 Programming Assignment 3

Due in the Digital Drop Box by 11:00 PM on Wednesday, November 18

Each of the following problems is stated with an indication of its ease or difficulty, the number of non-blank lines in my solution, and the relevant course slides. The descriptions used for the PowerPoint slides are

- Python3: the slides that begin with those on the Python Standard Library (`Python3.zip` on Blackboard)
- matplotlib: the slides on matplotlib (`Matplotlib.zip` on Blackboard)
- NumPyIntro: the introductory slides to NumPy (`NumPyIntro.zip` on Blackboard)

**1** (Easy, 6 lines, Python3: 10-22). Write a Python script that prints the month and day for the current day and the same day of the week for the next five weeks. For a given day, print the full month name and number in that month (e.g., **November 13**). (Use `strftime()` to get the proper formatting.) The following is the output from my script when I ran it on November 6.

```
E:\SomeFolder>prob1.py
November 06
November 13
November 20
November 27
December 04
December 11
```

**2** (Pretty easy, 15 lines, Python3: slides 4-6 [module `os`], 8-9 [module `glob`], 11-12 [class `date`], 13-17 [class `timedelta`], 22 [method `strftime()`]). You have a folder called `Diary` that is a child of your current folder. This folder contains `.txt` files for recent months and the current month. The name of such a file will typically be the three-letter abbreviation for the month (beginning with one uppercase letter) followed by the two-digit year and then the extension `.txt`, but we allow any initial part of the month's name that is at least three letters long and begins with an uppercase letter. (We might copy a file for the month or start out producing it manually.) For example, the file for November 2009 would normally be `Nov09.txt`, but it could also be, e.g., `Novem09.txt` or `November09.txt`.

The file for a given month is supposed to contain an entry for each day. The first line of such an entry is of the form

```
<Weekday>, <Month> <Day>, <Year>
```

where

<Weekday> is the full name of the day of the week (e.g., "Monday"),

<Month> is the three-letter abbreviation of the month (including the period),

<Day> is the day of the month (always two digits), and

<Year> is the four-digit year.

An example is

```
Monday, Nov. 05, 2009
```

The next lines of a daily entry contain the user's reflection on the day. The entry ends with a blank line. The following is an example of part of the file for November 2009 (containing entries for two days).

Monday, Nov. 05, 2009

Today was rainy. I didn't go for a walk.

Tuesday, Nov. 06, 2009

Today was sunny. I went for a walk.

For this problem, you write a program that produces and fills in these diary files. The steps are as follows.

- Change to the `Diary` folder.
- Look for a file for the current month and year (where the month could be three or more digits).
  - If such a file is found, open it for appending.
  - Otherwise, make a file name consisting of the three-letter current month, the two-digit current year, and the extension `.txt`, and open this file for writing.
- Prompt for and read the user's reflections for the day. (You may use `raw_input()`.)
- Write to the file the entry for today in the format indicated above.
- Close the file

Look at slide 22 of Python3, on `strftime()`. The format directives not shown there that you will need for this problem are

- `%b`: three-letter abbreviated month name
- `%Y`: four-digit year

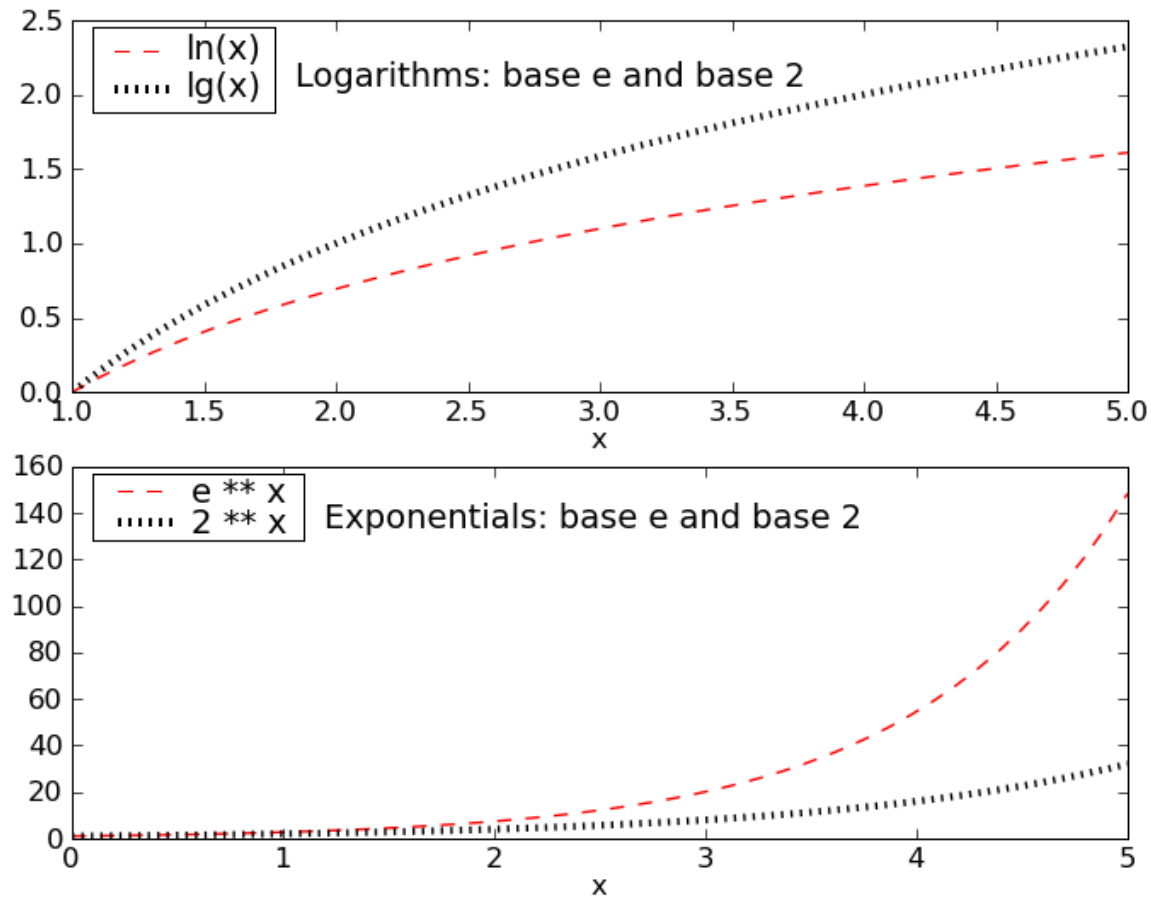
**3** (Easy, 7 lines, Python3: 34, see also <http://docs.python.org/library/math.html>). Write a program that prompts for and reads the polar coordinates of a point (where the angular coordinate is in degrees) and outputs the Cartesian coordinates of the points. Use only two decimal places in the output. The following are two example runs.

```
C:\SomeFolder>prob3.py
The angular coordinate: 45
The radial coordinate: 10
( 7.07, 7.07)
```

```
C:\SomeFolder>prob3.py
The angular coordinate: 225
The radial coordinate: 10
(-7.07, -7.07)
```

Python trig functions expect arguments in radians. Recall that  $\pi/180$  is the factor for converting from degrees to radians.

4 (Medium difficulty, 21 lines, matplotlib, NumPyIntro). Produce the figure shown below. One subplot shows  $\ln(x)$  (red dashes) and  $\log_2(x)$  (black dots) for  $x \in [1.0, 5.0]$ , while the other shows  $e^x$  (red dashes) and  $2^x$  (black dots) for  $x \in [0.0, 5.0]$ . To have the dotted lines show up, make their linewidth 3.0.



To get a NumPy array of equally spaced points, use the NumPy function `linspace()`. The NumPy functions for  $\ln(x)$ ,  $\log_2(x)$ , and  $e^x$  are, respectively, `log(x)`, `log2(x)`, and `exp(x)`. For  $2^x$ , use the NumPy `pow()` function as `pow(2,x)`. Recall that a NumPy scalar function applies element-wise to an array, as shown below.

```
>>> from numpy import *
>>> x = linspace(1,4,8)
>>> print pow(2,x)
[  2.    2.69180039  3.62289466  4.87605462  6.56268285
  8.83271611 11.88795431 16.    ]
```

5. (Pretty hard, 32 lines, matplotlib, NumPyIntro). We have a square dartboard with coordinates  $[0-20] \times [0-20]$ . We throw a dart at the board 17 times, and the following file, `prob5.dat`, records (to the nearest whole number) the locations where the dart landed ( $x$  first, followed by  $y$ ).

```
10 10
12 8
5 15
9 13
15 5
7 9
1 12
12 8
8 12
15 5
9 13
10 10
14 9
1 12
7 9
1 12
9 11
```

Write a program that reads in this file and produces the figure on the next page, which shows the various locations where the dart landed. The more times the dart landed at a given coordinate pair, the larger is the circle centered at that pair. In this scatter plot, the array of sizes (the value of the keyword argument `s` to `scatter()`) is the array of hits at the coordinates given by the same index of the arrays of values of  $x$  and  $y$  transformed as follows. Each size is raised to the power 3.5 then multiplied by 20. (With NumPy arrays, you can do this to all elements in the array at once.)

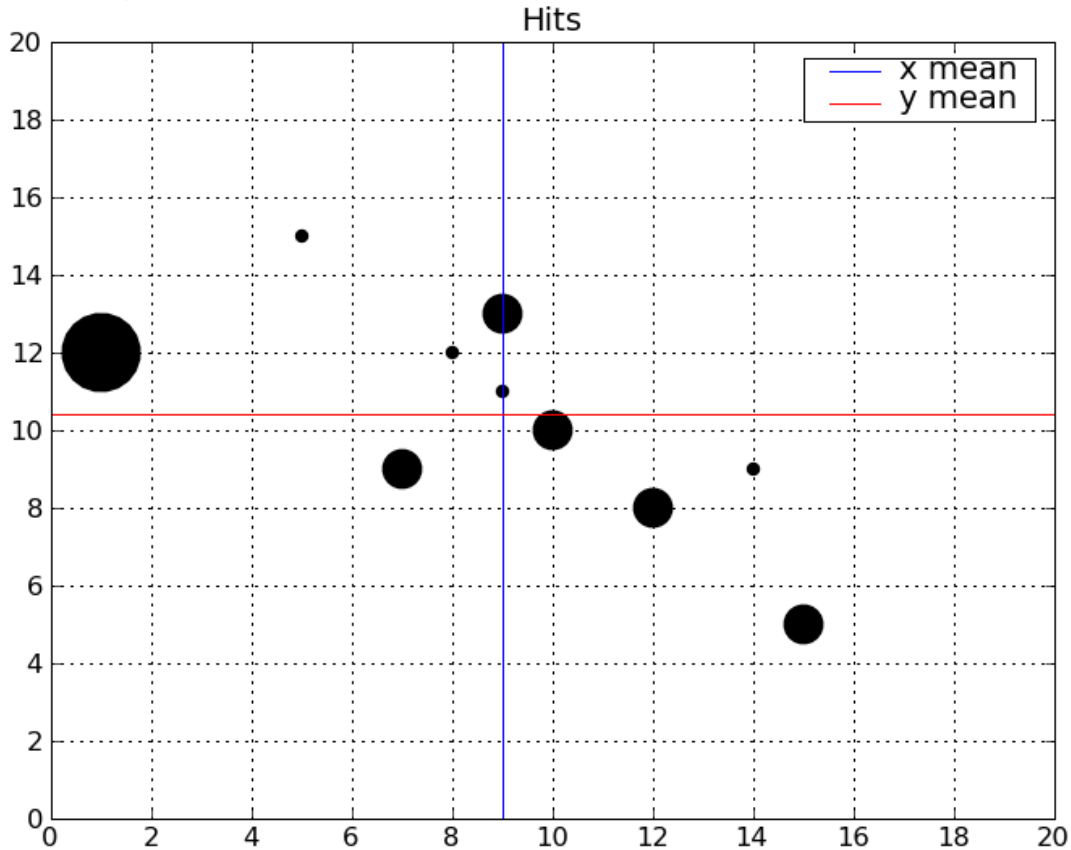
The mean of the  $x$  values is shown as a blue line parallel to the  $y$ -axis (using `axvline()`), and the mean of the  $y$  values is shown as a red line parallel to the  $x$ -axis (use `axhline()`). Note that, if `ar` is a NumPy array, then `ar.mean()` returns the mean of the values in `ar`. Be sure to set the tick labels as shown, to indicate the grid, to include the title and legend, and to show the entire board (all the way up to 20 in each dimension).

In outline, the way I did this is as follows. (You are welcome to try other ways.) Use a dictionary whose keys are tuples representing coordinate pairs. The following illustrates.

```
>>> d = {}
>>> d[(1,2)] = 1
>>> d[(3,2)] = 1
>>> d[(1,2)] += 1
>>> print d[(1,2)]
2
>>> print d
{(1, 2): 2, (3, 2): 1}
```

Read in the file a line at a time, forming a tuple of integers for each line. If the tuple is not already in the dictionary, add it with associated value 1. If it is in, then increment the associated value by 1. After the file is read, loop over the keys (tuples) of the dictionary and form three lists: one (for the  $x$  values) from the first element of each tuple, one (for the  $y$  values) from the corresponding second element, and one (from which the size will be derived) from the associated value. Form NumPy arrays from these lists, which

provide the three main arguments to `scatter()`, with the array of sizes transformed as indicated above. It is straightforward to add the other features of the figure.



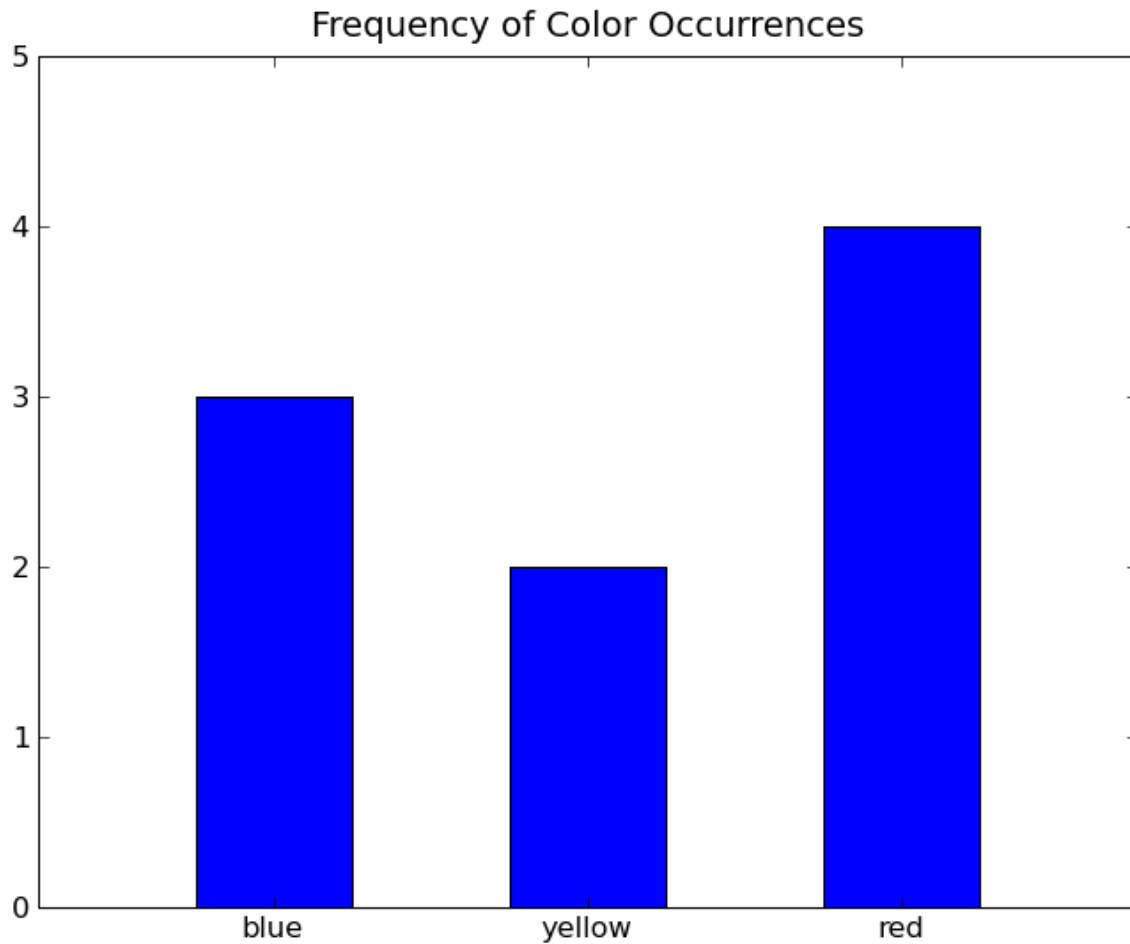
**6** (Not too hard, 20 lines, matplotlib [esp. the example on slide 60]). Write a program that counts the number of occurrences of “red”, “yellow”, and “blue” in the following file, `prob6.dat`, and constructs a bar chart indicating the number of occurrences of each as in the figure on the following page.

```
red
yellow
blue
black
red
blue
Yellow
red
yellow
red
blue
```

If a line has a word that is not one of “red”, “yellow”, or “blue”, then the program echoes back the word, stating that it is not a valid color. (Note that same case is required.) The following shows a run (which produces the figure on the next page).

```
E:\Old D Drive\c690f09\HW\HW3>prob6.py
[black] is not a valid color.
[Yellow] is not a valid color.
```

The width of the bars is 0.5, and their left sides are at 0.5, 1.5, .... The xticks are labeled with the color words and are located halfway between the left and right edges of the bars. The heights of the bars, of course, are the numbers of occurrences of the corresponding color words. The  $x$ -axis extends from the origin to one bar width to the right of the last bar, and the  $y$ -axis extends from the origin to one more than the largest count. Note that, where  $ar$  is a NumPy array,  $ar.max()$  is the largest value in it. Write the program so that, if one or more valid color words were added (or removed), a bar chart with similar appearance would be produced.



In outline, the way I got the counts is as follows. (Again, you are welcome to try other ways.) Make a set of the valid color words, and make a dictionary whose keys are these words, all with initial associated values of 0. As each line of the file is read, strip the newline, and check whether the result is in the set of color names. If not, report this fact and get the next line. If so, then increment the associated value in the dictionary. The remaining steps in the program are similar to those in the example on slide 60, but note that there are no error ticks or error bars here.