NYU, Tandon School of Engineering
CS-UY 1114 Introduction to Programming and Problem Solving — Fall 2017

Homework #8
Due by Friday 12/1, 11:55pm

Submission instructions:

1. You should submit your homework in the NYU Classes system. You should turn in 2 '.py' files.
2. For questions 1, create one '.py' file containing all the functions required for this implementation. Name your file 'YourNetID_hw8_q1.py'
3. For question 2, implement your code in the supplied file (hw8_q2.py). Rename it so it will match the format 'YourNetID_hw8_q2.py'

Question 1: Unscramble It

a. Implement a function `create_permutation(n)` that creates and returns a list containing a random permutation of the numbers: 0, 1, 2, …, (n-1).

For example, one call to `create_permutation(6)` could return the list: `[3, 2, 5, 4, 0, 1]`, another call to `create_permutation(6)` could return the list: `[2, 0, 3, 1, 5, 4]`.

For more detail on permutation, check this link: https://en.wikipedia.org/wiki/Permutation

b. Implement a function `scramble_word(word)` that is given a word (a string), and returns a scrambled version of `word`, that is new string containing a random reordering of the letters of `word`.

For example, one call to `scramble_word('pokemon')` could return `'okonmpe'`, another call to `scramble_word('pokemon')` could return `'mpeoonk'`.

Implementation requirement: To determine the new order of the letters, use the function `create_permutation(n)`. For example, for the word `'pokemon'`, the scrambled word implied by the permutation `[1, 4, 5, 2, 3, 0, 6]` is `'omokepn'` (the first letter is the letter from index 1, the second letter is the letter from index 4, the third letter is the letter from index 5, and so on).

c. You are given a text file containing a bank of words (one word in each line). Write a `main()` function that randomly chooses a word from the file, scrambles it, prints the scrambled word to the user, and allows them to find the unscrambled word 3 times.

Have your program interact with the user as demonstrated below:

```
Unscramble the word:   o m o k e p n

Try #1: openkom

Wrong!

Try #2: pokemon

Yay you got it!
```

Notes:

1. Your `main` should use the function/s you implemented in the previous sections.
2. When printing the letters of the scrambled word, add a space between each two letters.

Question 2: What's the weather like?

Many things nowadays rely on storing and handling data. Data exists on many different platforms and in many different fields and applications, such as libraries, weather services, mobile applications companies, etc. Datasets can be enormous; Google and Facebook both own datacenters that occupy many square miles just to store small pieces of data of each user they service. Amazon Web Services and Cloudfront provide cloud hosting solutions to store data for other various apps, such as Foursquare, Yelp, Reddit, and many others. With such large data to handle, it is imperative to have a good system to store and access the data. Many solutions involve using some form of database, either with SQL solutions such as MySQL and SQLite or NoSQL solutions such as MongoDB. Apart from having a database backend system, it is up to the programmer to design a clean and proper database scheme with tables that make sense.

In data sciences, a lot of raw data is collected from experiments and can result in huge files. A lot of this data is sometimes extraneous, and thus is deleted from the files, and sometimes significantly slimming down the size of the file and database to be created. Sometimes, some data is moved around to another table or the columns are switched around for easier handling. This is what's known as *data scrubbing* or *massaging*.

Database managers have this term known as CRUD – Create, Read, Update, Delete. For this homework, we will be mainly focusing on the Create and Read parts using Python and CSV (Comma Separated Values) files. As you already know, computers can open, create, modify and delete files – and for this assignment, we'll be using Python to do exactly that.

***Part A – data massaging*:**

We have supplied you with a CSV file of weather data from several cities gathered from the National Weather Service. Look at the data in a text editor and get a good feel for what it contains. In this file, all the temperatures are in Fahrenheit and precipitation is measured in Inches.

We only want the city, date, high/low temperatures and the amount of precipitation.

In the Python file supplied, complete the implementation of the function:

```
clean_data(complete_weather_filename, cleaned_weather_filename)
```

This function gets two strings as parameters: The first is the **name** of the file containing all the weather information, and the second is the **name** of the new file that the function **creates**. After cleaning the data, the new file will have only the city, date, high and low temperatures and amount of precipitation.

The function should use the file containing the weather information, passed as the first parameter (`complete_weather_filename`), read in the data, select only the specific columns mentioned and create a new file containing only the data we want.

Note: some precipitation values are non-numeric. In those cases, put the value 0 in the "cleaned" file.

***Part B – data massaging, cont'd*:**

Since not everyone in the world calculates in Fahrenheit and Inches, we want to convert our data to metric units.

Implement the following functions:

`f_to_c(f_temperature)`

`in_to_cm(inches)`

Make these functions return values in their metric units.

Then, implement:

`convert_data_to_metric(imperial_weather_filename, metric_weather_filename)`

This function gets two strings as parameters: The first is the name of the file containing weather information in Fahrenheit and Inches units, and the second is the name of the new file that function **creates**. After executing this function, the new file will have the weather information in Celsius and Centimeters units.

The function should use the file containing the weather information, passed in the first filename parameter (`imperial_weather_filename`), read in the data, convert it to the metric system units and create a new file containing the weather information in the **same format** but in the metric units.

Notes: Assume that the weather information in the input file (`imperial_weather_filename`) is in the format that `clean_data` (the function from part A) created.

***Part C – Working with the data*:**

Now that we have a clean and usable data file, we can start making calculations with it!

Complete the implementation of the function:

`print_averages_per_month(city, weather_filename, unit_type)`

This function should print out the average high temperature and the average low temperature for each month for the `city` passed in. For each month, the average should include the data over all of the years in the weather file given.

Notes:

1. The second parameter, `weather_filename`, is a name of a file containing "cleaned" weather. That is, this file was either created by the function `clean_data` (implemented in part A), or by `convert_data_to_metric` (implemented in part B)
2. You may assume that the `unit_type` parameter will be either `"imperial"` or `"metric"`, and that it will match the units of the data in `weather_filename`.
3. Your code must work with all files storing the data in this format, containing data of any timeframe.

For example, if we call:

```
print_averages_per_month("San Francisco", "imperial weather.csv", "imperial")
```

It should print out in the following format (the actual data will obviously be different):

```
Average temperatures for San Francisco:
January: 64F High, 52F Low
February: 62F High, 49F Low
March: 66F High, 55F Low
.
.
.
```

Hints:

1. The date format in our files is Month/Day/Year. Use the split method to separate the different fields of the date.
2. Since you have to calculate 24 averages (high and low for each one of the 12 months), instead of holding 24 sum variables, you can use two lists: one for the sums of the highs, and one for the sums of the lows. For example if `high_sums` is the list that holds the sums of the high temperatures, then `high_sums[0]` will be the sum of the high temperatures for January, `high_sums[1]` will be the sum of the high temperatures for February and so on.
3. To make the printing easier, you may need an extra list with month names.

***Part D – Working with the data, cont'd*:**

Think of a question that could be interesting to investigate, using this data. Write a function that queries the data file to answer this question. Add a few lines of code in the `main` to interact with the user and call your function.

For example, if you choose to compare the average rainfall of two given cities, you'd write something like:

```
# Q: Given two cities, which has higher average rainfall?

def higher_rainfall(city1, city2, weather_filename)
      # city1 and city2 are names of two cities
      # return which is the one that has higher average rainfall
```