Assignment 7

Objectives: Practice working using dictionaries and files, as well as their methods. Learn how to use basic functionality of the csv module. Sort dictionaries and apply new tools to old problems.

Note: Include DocStrings in each script you submit analogous to the following:

```
"""Retrieves name and phone# of members of the house of representatives



Submitted by Mauricio Arias. NetID: ma6918
This script scans a csv document for the correct information. It
processes it using dictionaries and strings.
"""
```

Part 1. Data structures: records. (4 points total assigned as described below.)
In the file House_of_representatives_117 S22.csv, a listing of all the members of the current house of representatives (Congress 117) is provided. For this assignment only voting members will be considered: state and district (at large or numbered). There are 435 voting representatives in the house. However, some of them are listed as vacancies in the party column. (We will ignore this aspect.) There are 6 non-voting members, listed as 5 delegates and 1 commissioner: 5 territories and D.C., repectively.

Part 1.1. Making the mini-dictionaries and the encompassing dictionary. (**2 points**)
Open the file and read the labels using readline().
Start a pythonic loop to process the file.
For each district/state combination make a mini-dictionary with the different columns as key value pairs: use the label of the column as the key and the datum in the csv file as the value. (Try to figure out how to do this part. However, if you get stuck here or decide to use indices, stop and check the hint at the end of this section: indices are not OK for this assignment.)
All these mini-dictionaries or records will be stored in an encompassing dictionary using a tuple with the information for state and district as the key: for example, the tuple `("New York", 12)` will be the key to the information for the representative for NYU's district. The full set of columns will be stored as a mini-dictionary using the labels for the columns as keys and the values for the representative as the values. Gather all the information only once, while storing it in the encompassing dictionary. Once ready proceed to the next part.

Part 1.2 Getting a query from the user and performing a search in the encompassing dictionary. (**1 point**)
Ask the user for a state and a district number. Create a tuple to match what was used in the encompassing dictionary.
Find the information for the representative by checking the dictionary for the appropriate information. Make sure you know when no match was found.

Part 1.3. Displaying the information appropriately. (**1 point**)
Provide the full name in the format "given name" "family name" and the phone number of the representative. For example for the 12th district of New York (where NYU is located), the information is:

```
The representative for district 12 in the state of New York is Carolyn
Maloney. The phone number is (202) 225-7944.
```

If no match was found, report that:

```
No representative was found for district 5 in the state of Iowa.
```

Keep asking until an empty string is given. Notice that this time we only need to read the file once because we are relying on powerful data types. Make sure you close the file once you are done with it. Submit your script as `[NetID]_house_dictionary.py`.

Hint: Learn about zip(). This tool allows joining the information of two lists so that the elements can be matched and used simultaneously in pythonic loops.

Part 2. Sorting.
Sorting of dictionaries is done using the sorted function on the corresponding list of keys. Review what sorted does by checking its documentation page. In particular figure out how to sort a list of keys. Start with simple examples such as

```
fruits = {"apples": 1, "pears": 10, "bananas": 5, "grapes": 0}
ordered_fruits = sorted(fruits)
print(ordered_fruits)
```

Try your own lists or dictionaries. Notice how sorted returns a list and not a dictionary. In particular, it gets a list with the keys of the dictionary and works on it. This is important as this function does not work with the dictionary itself. Make sure you understand what this resulting list is. This list can now be used to generate a new dictionary. Try it using a dictionary comprehension. (If you want to go further, check mappings in Python. Don't use them in what you submit though: let's keep things simple.)

Now let's sort a list of keys using the values of the corresponding data in dictionary. In the code below, the notation `key=lambda x: fruits[x]` defines an unnamed simple function that takes a single argument x and performs a simple evaluation: it retrieves the value for the key x in this case, namely `fruits[x]`. The result of this evaluation is what is used in the comparisons to order the elements of the list, where x takes on each key as the sorted function goes through the list. Hence the list of keys will be sorted by the corresponding values in the dictionary.

```
fruits = {"apples": 1, "pears": 10, "bananas": 5, "grapes": 0}
ordered_fruits_by_quantity = sorted(fruits, key=lambda x: fruits[x])
print(ordered_fruits_by_quantity)
```

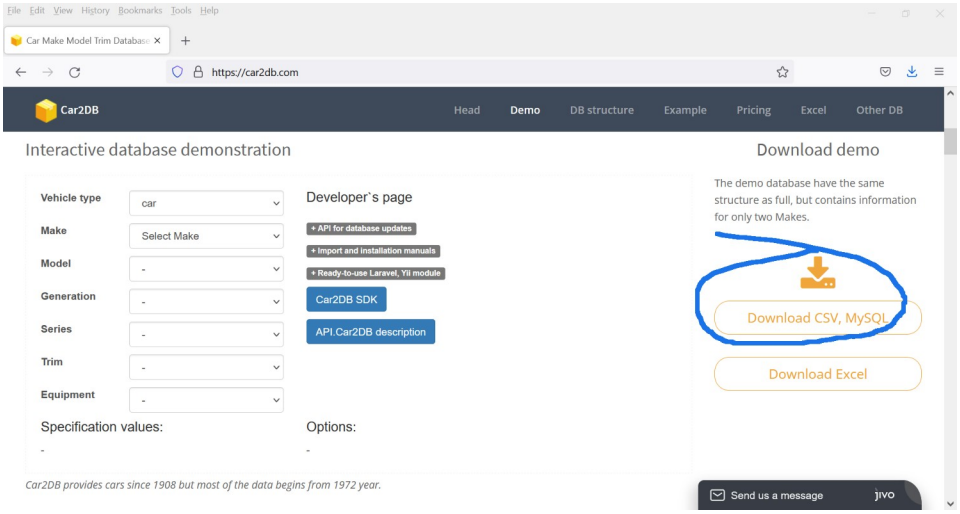
In these exercises we will use some help. There is a module that provides tools for reading csv files. This module is called csv. For the moment being we will use it with the default values.

Go to the python documentation page for csv and learn how to open a csv file and read it using the tools provided by the csv module. Don't use with blocks.

Notes: Read the following if you are getting stuck.
For task 2.1, notice that the documentation states that "*Each row read from the csv file is returned as a list of strings.*" Try opening the file the usual way, creating a csv.reader() and then iterating over the rows.

For task 2.2, notice that the documentation states that csv.DictReader() "maps the information in each row to a dict[ionary]." Try it by openning the file the traditional way, creating a csv.DictReader(), getting the fieldnames and then iterating through the rows.

Task 2.1. 2 points
Go to https://car2db.com/ and download the demo files on the right: they will download as a zip file.



Task 2.1.1. Opening a file using tools from the csv module. (**1 point.**)
Write a script that opens the car_equipment.csv and makes a dictionary using the first column as the key and the 'year' column as the value.

Task 2.1.2. Sorting simple data by key. (**0.5 points.**)
Order the dictionary by id in ascending order.
Save this as a csv file.

Task 2.1.2. Sorting simple data by value. (**0.5 points.**)
Order the dictionary by year in descending order.
Save the info as a csv file.
Use only basic tools and handle exceptions when ordering.
Describe how you handled the exceptions using comments in your script.

Save the script as `[NetID]_car_ids_w_year.py`.

Task 2.2. Sorting more complex data. (4 points as assigned below.)
The file `Food_contents_2019 S22.csv` is provided with information about different types of food. It has been cleaned up a bit and encoded using `latin1`.

Task 2.2.1. Open the file and read the rows using DictReader(). (**1 point**)
Go back to the documentation of the csv module and learn about `DictReader()`. Open the file above using latin1 for encoding. Play with the attribute `fieldnames` of the newly made `csv.DictReader` object. Make sure you understand what it is and how to use it. Try it. (Hint: It reads the rows as mini-dictionaries already.)

Read each row as a dictionary. Make an encompassing dictionary using whatever is the first column as the key, with the dictionary obtained from the row as the value. Notice that this is a dictionary inside of a dictionary, as a value, aka a mini-dictionary.

Task 2.2.2. Sorting the dictionary using specific columns. (**1 point**)
Sort this dictionary using "Energy (kcal)" in descending order. Notice that the values obtained in the dictionary are strings. Leave them as strings. Conversion to float is required when performing the comparisons, though. Save this information using `\t` as the separating character. Name the file `[NetID]_food_contents_by_energy.csv.`

Task 2.2.3. Processing the sorted lists to identify the keys ranked first. (**1 point.**)
Print the food item(s) with the highest energy content. (There might be more than one and your script should be able to handle that.)

Task 2.2.4. Handling different situations. (**1 point.**)
Sort this dictionary using "Carbohydrate (g)" in ascending order. Save this information using `\t` as the separating character. Name the file `[NetID]_food_contents_by_carbohydrate.csv.` Print the food item(s) with the highest carbohydrate content. (There might be more than one and your script should be able to handle that.)

Submit your script as `[NetID]_food_contents.py`.

Hint: Don't forget to convert the values of the energy and carbohydrate columns to floats when performing the comparisons.

To learn more: Try to open the file using the old method to handle csv files. What happens?