DATA203 Foundational Python (Prof. Maull) / Fall 2024 / HW1

Points Possible	Due Date	Time Commitment (estimated)
15	Sunday, September 29	*up to* 15 hours

- **GRADING:** Grading will be aligned with the completeness of the objectives.
- **INDEPENDENT WORK:** Copying, cheating, plagiarism and academic dishonesty *are not tolerated* by University or course policy. Please see the syllabus for the full departmental and University statement on the academic code of honor.

OBJECTIVES

- Explore JupyterHub Python *shell* commands inside cells
- Understand and use dictionaries of data.

WHAT TO TURN IN

You are being encouraged to turn the assignment in using the provided Jupyter Notebook. To do so, make a directory in your Lab environment called `homework/hw0`. Put all of your files in that directory. Then zip or tar that directory, rename it with your name as the first part of the filename (e.g. `maull_hw0_files.zip`, `maull_hw0_files.tar.gz`), then download it to your local machine, then upload the `.zip` to Canvas.

If you do not know how to do this, please ask, or visit one of the many tutorials out there on the basics of using zip in Linux.

If you choose not to use the provided notebook, you will still need to turn in a `.ipynb` Jupyter Notebook and corresponding files according to the instructions in this homework.

ASSIGNMENT TASKS

(0%) Explore JupyterHub Python *shell* commands inside cells

In the last time we learned to run the terminal console commands in JupyterLab, which is a great way to perform command-line tasks and is an essential tool for basic scripting that is part of a data scientist's toolkit. Last time we used a terminal console in the lab environment this time we familiarize ourselves with Jupyter *shell* escape commands **within** a notebook.

Study:

- Python Data Science Handbook: IPython and Shell Commands

for full documentation on *shell* … they are **very** useful!

§ Task: Use Jupyter *shell* commands to perform the same commands as last time.

Basic file operations go a long way to understand the way Linux works. In this part, you will understand folders, files and making revisions to a file. These files will be visible within Jupyter, which makes moving from one platform to another seemless. We will create a folder, file and make edits.

- type `!mkdir your_folder_name` to create a folder in filesystem *in the current folder where you are*
- create a file by type `touch README.md` the `touch` command creates a file if it does not already exist, otherwise it will change the timestamp of that file when it is "touched"
- edit the file in Jupyter with the text editor
- to see the contents of your file typing `!cat README.md`

§ Task: Use *shell* command `wget` to quickly obtain remote files in Linux

As before get a remote file from LoC:

- in a cell type `!wget https://tile.loc.gov/storage-services/service/pnp/cph/3c10000/3c19000/3c19900/3c19985v`

- execute the cell
- verify the file was retrieved by opening it

(100%) Understand and use dictionaries of data.

We learned in lecture that dictionaries are very flexible data structures in Python.

I have provided a function for you to use, which operates on the files located in this Github repository:

- https://github.com/kmhuads/f24_data203/tree/main/hw1

You will notice there are three files:

- `data/appetizers.csv`
- `data/deserts.csv`
- `data/entrees.csv`

I recommend you look at these files and see what is in them. CSV files form the basis of many data files that you will interact with in your future as a data scientist – *maybe* one day AI will take over our reliance to write just a little code to deal with them, but we will see … but not today.

I have provided code and a notebook for you to work with to get the work done.

Here is that function below, should you want to use your own notebook. If you choose to do so, you will need to also make a copies of the corresponding files mentioned above.

```
def make_menu():
  import csv
  import glob

  m = {}

  for f in glob.glob("data/*.csv"):
      k = f.split(".")[0]
      rows = csv.reader(open(f))

      _m = {}
      for itm in rows:
          _m[itm[0]] = [itm[1], itm[2]]
      m[k] = _m

  return m
```

§ Task: Practice explaining code.

Use the function `make_menu()` above to answer the questions below and **provide your answers in a Jupyter Notebook.**

1. Explain in your own words what the function does. Your explanation will include the description of the inputs, and you will need to review the Python libraries `glob` and `csv` to include the details of the function. What is most important is `glob.glob([pathname])` returns a list of pathnames strings (including filename) that match the pattern given, and `*` acts as a *wildcard*.

   Please note `csv.reader(file_object)` returns a row iterator over a file object, allowing one to loop over the rows of the file one by one using `for ... in:`.

   Finally, you might want to use `pprint` to display the data in a way that allows you to see the result of `make_menu()`. Study `pprint.pprint()` to learn more. You may want to look directly into the `csv` files to help you understand what is going on.

   For example, if you have a dictionary `d` and write `pprint.pprint(d)` you will get something like below:

   ```
   d = {"Nigeria": {"Lagos": {"population": 14000000}, "Kano": {"population": 3500000}}}
   pprint.pprint(d)
   ```

produces a a more readable version:

```
{'Nigeria': {'Kano': {'population': 3500000},
             'Lagos': {'population': 14000000}}}
```

§ Task: Perform basic dictionary operations.

Now that you have a way to use `make_menu()` we will want to *actually* use it.

1. Write the Python code to print the **prices** and **total** of the following order:

```
appetizer .. quantity=1 .. Jamaican Patties
appetizer .. quantity=1 .. Conch Fritters
entree    .. quantity=1 .. Curry Goat
dessert   .. quantity=1 .. Mango Sorbet
```

   Your output will look like this:

```
Jamaican Patties ..  6.50
Conch Fritters   .. 10.50
Curry Goat       .. 21.50
Mango Sorbet     ..  4.00
===
TOTAL            .. 42.50
```

   To do this your code **must** pull the data from the dictionary, print the output **and** compute the total.

§ Task: Perform more complex Dictionary operations.

Now we will put looping and other tools to task for more complex operations.

1. You will need to take this Dictionary:

```
order001 =

{
'order':
    'appetizers':
    {
    'Guava Duff': 3,
    'Callaloo Fritters': 2,
    'Conch Fritters': 2,
    'Jerk Chicken Wings': 2,
    }
    'deserts':
    {
    'Mango Sorbet': 2,
    'Ackee and Saltfish Cakes': 1,
    'Sweet Potato Pudding Cake': 1,
    'Coconut Rice Pudding': 1,
    }
    'entrees':
    {
    'Curry Goat': 1,
    'Brown Stew Fish with Rice and Peas': 1,
    'Jerk Chicken': 1
    }
}
```

   And produce a function that given the Dictionary will use the menu from `make_menu()` and calculate the order total. Name the function `order_total()`.

   NOTE: The order Dictionary assigns the item quantity as *values* to the *menu item key*, so that for example

```
'Callaloo Fritters': 2,
```

means there were **2** Fritters in the order.

Your final cell might look like this:

```
menu = make_menu()

order_total(menu, order001)
```

which would return:

```
TOTAL: $999 <- the real total
```

2. Improve the function so that it produces the breakdown of each meal course. Your output will look like:

```
TOTAL: $999
  Appetizers: $333
  Entrees   : $600
  Deserts   : $ 67
```

§ Task: BONUS

This part will earn you *up to* an extra +8 BONUS points.

1. [**+4 points**] Write a function that uses Sets to compare how similar the descriptions of menu items are.
   a. For **1 point**, write the **function** that given 2 descriptions will return the count of the number of words in common **excluding** those in this list:

   ```
   exclusion_words = [
     "served", "and", "a", "with", "are", "these", "side",
     "in", "of"
   ]
   ```

   b. For **2 more points** (total of 3 points if prior part is successfully completed), use the function to determine which menu items are most similar to one another. You will need to loop over all items and return their similarity score, such that the output would look like this:

   ```
   'Guava Duff' <=> 'Curry Goat' : similarity = 2
   'Guava Duff' <=> 'Mango Sorbet' : similarity = 0
   ```

   c. For the **full 4 points**, print the **sorted** list of all menu items pairs and their similarity. The sort order is **descending** by highest similarity to lowest.

2. [**+2 points**] Write a function `make_order()` which takes the menu Dictionary from `make_menu()` and returns a **list** which contains an order with random items from the menu with:
   a. at least one appetizer, one entree and one desert

   The output might look like:

   ```
   menu = make_menu()
   make_order(menu)
   ```

   Minmally, you output would look like this:

   ```
   ['Jamaican Patties', 'Jerk Chicken Wings', 'Brown Stew Fish with Rice and Peas',
     'Brown Stew Fish with Rice and Peas', 'Jerk Chicken',
     'Curry Goat', 'Oxtail Stew', 'Coconut Rice Pudding',
     'Mango Sorbet', 'Guava Duff', 'Mango Sorbet', 'Coconut Rice Pudding',
     'Guava Duff']
   ```

   b. For **one more bonus point** (total of 3, if the prior bonus was successfully completed): make your function return a dictionary of the food order with the keys being the course (appetizer, entree, desert) and the values being the item.

Minimally, your output would look something like this:

```
ORDER #NNNN

{'appetizers':
            ['Callaloo Fritters', 'Ackee and Saltfish Cakes'],
'entrees': ['Jerk Chicken',
            'Curry Goat',
            'Jerk Chicken',
            'Curry Goat',
            'Brown Stew Fish with Rice and Peas'],
'deserts': ['Guava Duff',
            'Coconut Rice Pudding',
            'Sweet Potato Pudding Cake',
            'Guava Duff',
            'Mango Sorbet',
            'Sweet Potato Pudding Cake']}
```

c. For **one more bonus point** (total of 4, if the prior bonus successfully completed), make the order like the original dictionary as in part 1 above.