DATA203 Foundational Python (Prof. Maull) / Fall 2025 / HW1

Points Possible	Due Date	Time Commitment (estimated)
15	Tuesday, September 30 @ midnight	*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
- List, Loops and Leaves: Get more practice using `for` loops and lists.
- Dictionary Deluge: Understand and use dictionaries for complex 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/hw1`. 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_hw1_files.zip`, `maull_hw1_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 seamless. 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 this time it will be from the Internet Archive:

- in a cell type `!wget https://ia801306.us.archive.org/15/items/fouraddressesats00howa/fouraddressesats00howa`
- execute the cell
- verify the file was retrieved by opening it

(50%) List, Loops and Leaves: Get more practice using `for` loops and lists.

We learned in lecture that looping is a necessary tool for computation in Python and all coding. This part of the assignment will focus on loops and the string type.

I have provided a starter notebook for you to use, which will greatly enhance you ability to complete the assignment. See that in the folder here. The name of the notebook is `hw1/hw1_starter.ipynb`:

- https://github.com/kmhuads/f25_data203/tree/main/hw1/hw1_starter.ipynb

Look at this file and see what is in it – use it since in the notebook are some scaffolding code. You will need to study it and use it in the solutions being asked.

You will use the starter notebook for your answer to write the Python code required to complete it.

§ Task: Write a code that converts all the sentences in the `sentences` list to lowercase and prints the cleaned list.**

- A `for` loop will be necessary and you will also need to revist `str.lower()`.
- You can simply use `print()` to display the lowercased, unpunctuated sentences.

Example:

If

```
s = ['Pumpkin patches are often filled with adorable pumpkins and Fall decorations!!!']
```

then your code will output:

```
pumpkin patches are often filled with adorable pumpkins and fall decorations
```

There are many ways to solve this, so don't overthink it!

Use the starter notebook and answer write the Python code required to complete it.

§ Task: Write code that compares all sentences in the `sentences` list to one another and prints the list of common words or an empty list when there are no commonalities.

Starting with the Python code defininig `sentences`:

```
sentences = [
    "The vibrant colors of the changing leaves in autumn were breathtakingly beautiful.",
    "As the days get shorter, people start to cozy up with warm \
     drinks and snuggled blankets indoors.",
    "In the fall, apple cider donuts and pumpkin pie are a classic combination at many cafes.",
    "The crisp air and cooling temperatures make fall the perfect time \
     for outdoor activities like hiking and apple picking.",
    "As the weather cools down, people start to wear layers of sweaters and scarves to stay warm.",
    "The smell of wood smoke and fallen leaves fills the air during the fall season.",
    "In many parts of the world, the fall equinox marks the official beginning of autumn.",
    "The bright orange foliage of the sugar maples in the forest is a stunning sight to behold.",
    "As the nights get longer, people start to enjoy outdoor movies and \
     bonfires with friends and family."
]
```

Your output MUST look like this:

```
S1: The vibrant colors of the changing leaves in autumn were breathtakingly beautiful.
S2: The vibrant colors of the changing leaves in autumn were breathtakingly beautiful.

  Have the following words in common:
  -> breathtakingly, in, vibrant, the, leaves, were, changing, autumn, colors, of, beautiful
```

- As before you will need to use the lowercased version of the sentence (with `str.lower()`) or you will lose a point (and many of your answers will be wrong).
- Two `for` loops will be necessary – the outer loop will loop over each sentence as will the inner loop.
- You can simply use `print()` to display the list of words in common.
- You must use the provided function `common_words()`.
- Your output will contain 100 blocks (there are 10 sentences and each will be compared with all 10).

§ Task: Explain in your own words what the provided function `common_words()` does. Include the main data type used in the function and *why* that was an appropriate choice.

(50%) Dictionary Deluge: Understand and use dictionaries for complex data.

In this part we are going to work with dictionaries and loops again, as well as learn about two common file types that you will run across in Python (and other languages).

The first file type is call JSON or "JavaScript Object Notation" and is one of the most ubiquitous file formats today. It allows you to store data of all kinds, but in our particular case, as we will find out, it is well suited for mapping data, key-value paired data or what we now know to be dictionaries in Python.

You will want to pay close attention to this because being able to store and retrieve dictionaries and key-value paired data is greatly simplified when using JSON.

Here is a fragment of a JSON file, which will commonly have the file extension `.json`:

```
{
    "appetizers":
    {
        "Ackee and Saltfish Cakes":
        {
            "description_en":
                "Cakes crispy made with ackee fruit, salted cod fish,
                onions, bell peppers, and spices. A popular appetizer in
                Jamaica, these cakes are often served with a side of tangy
                tartar sauce.",
            "description_es":
                "Cakes crujientes hechos con fruto de ackee, pescado salado,
                cebolla, pimientos y especias. Un aperitivo popular en Jamaica,
                estos pasteles se sirven com\u00fanmente con
                una servilleta de salsa \u00e1cida.",
            "price": 12.99
        }
    }
}
```

Another file format you will run across is YAML, which stands for "Yet Another Markup Language". I personally find YAML to be a great way to store data more easily that JSON at first, and then when a JSON file is necessary, convert it accordingly. YAML is a superset of JSON and is actually easier to read in it's native file format (mostly because of the lack of curly braces and quotes), but nearly all computer systems can manipulate and display both formats beautifully, so it is really just a matter of personal preference until you run into a use case where the one might be more convenient over the other.

You just need to remember that YAML can be easily converted to JSON and visa versa.

YAML files commonly have the file extension `.yaml`. Here is an example based on the same fragment as before.

```
appetizers:
  Ackee and Saltfish Cakes:
    description_en: Cakes crispy made with ackee fruit, salted cod fish, onions, bell
      peppers, and spices. A popular appetizer in Jamaica, these cakes are often served
      with a side of tangy tartar sauce.
    description_es: "Cakes crujientes hechos con fruto de ackee, pescado salado, cebolla,\
```

```
      \ pimientos y especias. Un aperitivo popular en Jamaica, estos pasteles se sirven\
      \ com\xFAnmente con una servilleta de salsa \xE1cida."
    price: 12.99
```

In Python you can import each easily:

```
import json
import yaml
```

Official documentation is here:

- yaml module documentation on pyyaml.org
- json module documentation on Python.org

§ Task: Load the menu from the `data/` folder into a dictionary using the YAML and JSON versions.

§ Task: Use resulting dictionary structure from the prior task to calculate the cost of all entrees on the menu.

Here you will access the `entrees` key of the menu dictionary, then you will loop over all the `price` sub-keys and sum them up. There are a number of straight-forward ways to do this.

§ Task: Use the provided order below to write the code to summarize the order. Your output will include (1) the total item count in the order, (2) the order total (sum of all items), and then the breakdown of the total by category (`appetizer`, `entree`, `desert`).

Here is the order:

```
order001 = \
{
    'order': {
        'appetizers':
            {
            'Callaloo Fritters': 2,
            'Conch Fritters': 2,
            'Jerk Chicken Wings': 2,
            'Ackee and Saltfish Cakes': 1,
            },
        'deserts':
            {
            'Guava Duff': 3,
            'Mango Sorbet': 2,
            'Sweet Potato Pudding Cake': 1,
            'Coconut Rice Pudding': 1,
            },
        'entrees':
            {
            'Curry Goat': 1,
            'Brown Stew Fish with Rice and Peas': 1,
            'Jerk Chicken': 1
            }
        }
}
```

Your output should look like (with the correct values):

```
Order Item Count: 1234
Order Total: $123.45
    Appetizers: $456.78
```

```
Entrees: $345.67
Deserts: $890.12
```