DATA203 Foundational Python (Prof. Maull) / Fall 2024 / HW2

Points Possible	Due Date	Time Commitment (estimated)
20	Wednesday, October 23	*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
- Practice writing functions

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 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%) Practice writing functions

We learned in lecture that functions are the foundation of programming in Python.

We get a lot more practice writing them combining what we know about lists and dictionaries to get interesting results.

In the last homework you started to use functions as a prelude to this homework, which is more practice on a broader set of computational problems.

§ Task: Write a random password generator function.

Write a function called `password_gen()`. It will take one parameter: *n_len*, which is the password length parameter.

The function will return an empty String as well as print a if *n_len* is too small. The password length parameter must be **greater or equal to** 8 and the message if it is less than this, must be `"[warn] The password must be 8 or more characters."`

The password returned will be at **String** of 8 or more characters. The password should include a random set of characters which are uppercase, lowercase and numbers.

HINT: Study the `String` class or `chr()`. Recall we talked about the Python built-ins and PSL. You will notice there are constants within this library for `ascii_lowercase`, `ascii_lowercase` and `digits` which nicely match what the solution is supposed to be using.

Your code will not be more than a few lines, so do not overthink it and use loops.

You may find libraries like `random` to be invaluable.

Here are some examples: (your actual answers will vary)

```
password_gen(10)
# ==> lNmRht5Os5

password_gen(18)
# ==> oXu7XWQ2RbP1A7yoyg

password_gen(40)
# ==> fBUud2lWfGSKSz2LDLFD40lzuVYAWWtJZ0X5TO1W
```

To reiterate:

- `password_gen()` must take 1 parameter, called `n_len`
- the function must return a string with a random password
- the returned string must be 8 or more **random** characters from ALL of the following:
  1. lowercase letters
  2. uppercase letters
  3. numbers `{0 .. 9}`
- if `n_len` is less than 8, the return string is:
  - `"[warn] The password must be 8 or more characters."`

§ Task: Write a class assignment function.

Assume you are given three numbers. These numbers represent readings from an air quality sensor on campus next to the food trucks on campus.

You will write a function `assess_AQ()` that takes a **tuple** parameter `aq_reading` and returns a **tuple** with the assignment of the AQ reading to a class with additional data.

The input `aq_reading` will be a tuple of three numbers. The numbers represents a reading from the first sensor, second and third sensors in that order, so that

```
reading_001 = (45.3, 22.7, 88.7)
```

represents the readings from `sensor_001`, `sensor_002` and `sensor_003`.

Your function will return a tuple of the form:

```
(51.6, 3, "H", "The PM2.5 measurement is high.  Those with respiratory conditions should be advised.")
```

That is to say:

- the first element of the **returned** tuple is the **mean** of the three readings.
- the second element of the returned tuple is the **number** of readings (sometimes a sensor might not be working).
- the third element of the returned tuple is a single character String representing the severity of the **average** reading using:
  - `"H"` → average reading above 51
  - `"M"` → average reading between 27.5 and 51.9 (inclusive)
  - `"L"` → average reading less than 27.5
- the final element is a String message using:
  - if average reading **above 100** →

    ```
    "The PM2.5 measurement is very-high.  Air quality is very poor and a threat for most."
    ```

  - if average reading **above 51 and less than 100** →

    ```
    "The PM2.5 measurement is high.  Those with respiratory sensitivities should be advised."
    ```

  - if the average reading is **between 27.5 and 51.9** (inclusive) →

    ```
    "The PM2.5 measurement is moderate.  Changes in air quality conditions should be monitored closely."
    ```

  - if the average reading is **below 27.5** →

    ```
    "The PM2.5 measurement is low.  Air quality poses low threat to most."
    ```

  To reiterate:

  - `assess_AQ()` must take 1 parameter, called `aq_reading`
  - the function must return a tuple with 4 values
  - the returned tuple `(t1, t2, t3, t4)` must have the following:
    1. `t1` is the average of the input `aq_reading`
    2. `t2` is the number of values in aq_reading (usually 3, but it may be less)
    3. `t3` is the AQ class (`H`, `M`, `L`)
    4. `t4` is the string representing a displayable message about the readings

§ Task: Articulating function operation

Write a description of 2-4 sentences of what you think this function does. Write the description in a way that someone who **does not** know Python could understand.

```
def mystery_function(a, b=(0, 100)):
  a_split = a.split(",")

  if len(a_split) != 5:
    return
  else:
    a_split = [int(a_) for a_ in a_split]

    if not min(a_split) < b[0] and not max(a_split) > b[1]:
        return
    else:
      if a_split[2] > sum(a_split[0:2]) and a_split[2] > sum(a_split[-2:]):
        return True
      elif a_split[2] > min(a_split[0:2]) and a_split[2] < max(a_split[-2:]):
        return True
      else:
        return False
```

Here are some sample executions:

```
mystery_function("1,2,3,4,5")
# returns -> True

mystery_function("1,2,6,4,5")
# returns -> False

mystery_function("1,2,12,4,5", b=(1,3))
# returns -> None

mystery_function("1,2,3,4,5", b=(0,13))
# returns -> True
```

In your answer please include:

- what the *parameters* to the function are (e.g. what are they and how are they used)
- what the function actually does in relation to the various inputs
- examples of why (and when) the function returns `True`, `False` and `None`

Again, make sure your answer is *plain and clear*.