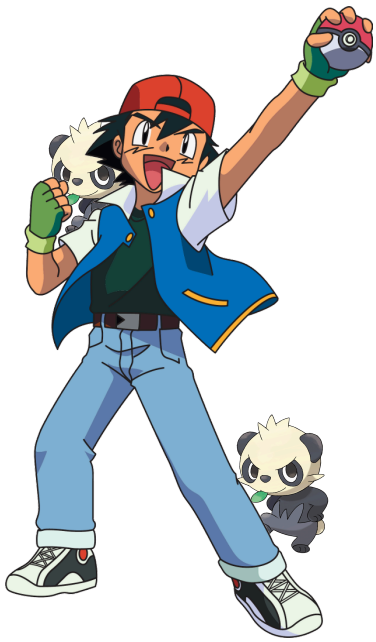
HW3: ~~Python~~ Pandas, I Choose You!



Requirements:

Just like last week, you should expect to submit your assignments to both GitHub and as a .zip file to Brightspace.

You should follow a similar format where you have a folder named `HW3` . You should then have a folder for each question underneath this parent folder (i.e. `Q1`, `Q2`, `Q3`). You should also have a folder named `data` under `HW3` that contains all of your data files. Each of your solutions should use the same data files (instead of having a separate copy in each problem folder).

Under each problem folder, you should have a file called main.py. In cases where you are prompted for a response instead of a solution, you can create a file called `answer.txt`.

A note on our datasets:

If you take a look at your datasets, you'll notice that there are just a bunch of numbers. In some form or another, in each file, you will find a column called `id`. In some of these files, you'll notice that there is a field called `something_id`, which aligns with one of our other datasets (i.e. between regions and locations).

This is a strategy called normalization that allows us to utilize identifiers between different datasets so that we can later connect them together. We'll talk more about this throughout the semester, but what you should be aware of now is if asked to grab data from one file and then lookup data from another, these fields can be useful to link between the two.

Problems:

1. Create a list of users and their test scores. Using this list, convert it to a DataFrame so we can begin utilizing pandas to work with the data. Your list should have at least ten entries. This application should run and then exit by printing out just the names from your DataFrame.

2. Create an application that keeps track of a user's score in a set of games. Your application should interact with the user and provide them three options:
    a. Add Score
    b. Check Scores
    c. Exit

    For option one, you should ask the user to provide a name for the user and the score for that game. After the user provides their inputs, they should then be returned to the main menu. They should be able to provide as many entries as they like until they opt for option three (exit).

    For option two, you should ask the user to provide a name for the user. You should then return a DataFrame containing the scores for that particular user before returning to the main menu.

3. Extend Q2 by adding an additional option that looks like the following
    a. Show Total Scores

    This option should provide the sum of all the scores for the provided username.

4. Extend Q3 by adding validation. If a user provides a name that doesn't exist within your dataset, you should return the line "User does not exist" before returning the user to the main menu.

5. Using the documentation at https://pandas.pydata.org/docs/reference/index.html, find the parameter for `read_csv` that would allow me to change the delimiter for a CSV from a comma to something else of my own choosing.

6. Using `poke.csv`, write an application that consumes this data and then gives me a summary of the data. Specifically, produce an application that will read this data in and return the following output:

```
The pokemon dataset consts of X columns and Y rows. It has the following
column names [list of column names here].
```

7. Using `poke.csv`, prompt the user for the name of a pokemon. If it exists within the dataset return the DataFrame containing data for only that pokemon. If it doesn't exist, instead return the message `This pokemon does not exist`.

8. Using `poke.csv`, prompt the user for a number. If the data provides isn't a number, return the message "Please provide a number". If the user again provides input that isn't a number, then return the message "Error" and exit immediately.

   If the input is a number, return the data for the pokemon at the index of that number and the pokemon that has that value as their `id`. This should all be returned within a single DataFrame object.

9. Revisiting a problem from HW2, let's recreate our team builder using pandas. Our menu should have four options:
   a. Add To Team
   b. Drop From Team
   c. Print Team
   d. Exit

   Similar to last week, a team can only have six members of the team. If you attempt to add more than that, you should return the message "Too many team members". The pokemon also needs to exist within our dataset. If it does not, you should return the message "This pokemon does not exist". If it does, you can add it to your team.

   Using the solution from last week, you can manage the team in the same way. If a user wants to drop a member from the team – they should be able to provide some identifier that you indicate to drop them from the team.

   If choosing to print the team,you should return a DataFrame showing rows for each of the current members of the team.

   After each option, the user should be returned to the main menu until they decide to exit.

10. Utilizing the `regions.csv` and `locations.csv` datasets, accept input from the user for a specific region name. You should then return the number of locations that exist in that region. If the region name does not exist in our dataset, then return the message "This region does not exist".

11. Utilizing the `locations.csv` dataset, return a list of locations that are not associated with a region.

12. We arrive to find our dear friend, Ash Ketchum, looking to get to know the world around him. He wants to be able to ask questions about the Pokemon that he is hunting and the locations that he'll need to travel to. In order to accomplish this, we need to build him a Python application, as every great Poke master requires, to help him track down this information.

In order to accomplish this, when run, the application should provide him a menu that looks like the following:

```
Hello, Ash!

1) Search By Name
2) Search By Region
3) Exit
```

If a user selects by name:

a. The user should be prompted for a pokemon name
b. It should be validated that the pokemon exist in our dataset
c. If it does not, tell Ash that this mysterious pokemon cannot be found
d. If it does exist, query our dataset and return the name, id, height and weight. This should be returned in a dataframe.

If a user selects by region:

e. Prompt the user for a name of a region
f. It should be validated that the region exists in our dataset
g. If it does not, tell Ash this this region does not exist
h. If it does exist, query our dataset and return the locations that belong to this region. This should be returned in a dataframe, but contain only the location names.

After each option, Ash should be returned to the main menu where he can then decide to continue searching or Exit.