Homework Turnin

Name:	Alexander A Miller
Email:	alexandermiller@email.arizona.edu
Section:	1E
Course:	CS 120 17au
Assignment:	hw2
Receipt ID:	43b5180d36a2708ad8f8b1858dda98e9

Turnin Successful!

The following file(s) were received:

pokemon.py (7796 bytes)

```
"""
File: pokemon.py
Author: Alexander Miller
Purpose: this program will:
* read in a data file from input *without* prompting the user
* organize pokemon based on their type 1 (ignoring type 2)
* calculate the types with the highest average stats
* read in specific queries from user (non-case senstivie) and print results
* program terminates when user enters empty enter
"""


def main():
    x = init()
    data = list2dict(x)
    averages, list_of_keys = average_values(data)
    max_val_dictionary, max_val = max_value_dictionary(averages, list_of_keys)
    handle_queries(max_val_dictionary, max_val)

def init():
    """
    Description: reads in a file, converts that file to a list organized by lines
    Parameters: none
    Returns: x, a list of lists containing each line in the infile
    Pre-condition: infile should be a csv containing pokemon and all necessary information/stats
    Post-Condition: x will be a list of lists containing each line in infile
    """
    x = []
    inputfile = input()
    openfile = open(inputfile,'r')
    for line in openfile:
        list_form = line.split(',')
        x = x + [list_form]
    openfile.close()
    del x[0] #removes that pesky establishing line
    # objective: convert all entries to integers if applicable
    accum = 0
    while accum != len(x):
        accum2 = 0
        while accum2 != len(x[accum]):
            try:
                x[accum][accum2] = int(x[accum][accum2])
            except ValueError: # catches errors for assigning str as int
                pass
            accum2 = accum2 + 1
```

```
        accum = accum + 1
    return x


def list2dict(x):
    """
    Description: this takes the list x and returns data, a dictionary form
    Parameters: x, a list of lists containing the lines of the infile
    Returns: data, a 2D dictionary organized by class and then resp. pokemon
    Pre-condition: x is list of lists
    Post-Condition: data will be 2D dictionary
    """
    # objective:  data = {'type_1':{'name':('total_strength', 'hp', 'attack', 'defense', 'special_attack'
    data = {}
    accum = 0
    while accum != len(x):
        if x[accum][2] in data: #note: since my x values have already been made into integers, i can now
           data[x[accum][2]][x[accum][1]] = (x[accum][4],x[accum][5],x[accum][6],x[accum][7],x[accum][8],
            #data[type][name] = (total_strength, hp, etc.)
        else:
            data[x[accum][2]] = {} #must create first key to assign second key
            data[x[accum][2]][x[accum][1]] = (x[accum][4],x[accum][5],x[accum][6],x[accum][7],x[accum][8]
        accum = accum + 1
    return data


def average_values(data):
    """
    Description: establishes a dictionary of averages organized by type and returns a list of types as we
    Parameters: data, a dictionary of all the pokemon
    Returns: averages: a dictionary of averages organized by type; list_of_keys: a list of keys (types)
    Pre-condition: data is a 2D dict
    Post-Condition: averages will be dictionary of lists, list_of_keys will be list
    """
    averages = {}
    list_of_keys = []
    list_of_keys = list(data.keys())
    accum = 0
    # objective: establishing a dictionary of keys (pokemon types) to enter and calculate average values
    while accum != len(data):
        averages[list_of_keys[accum]] = [0,0,0,0,0,0,0,0] # ok if it overwrites duplicates, b/c it's just
        # note: need this to be a list b/c it will be constantly updating its values and it needs 8 entri
        accum = accum + 1
    # objective: finding totals among each category
    for i in averages: #cycles thru the relevant keys
        for x in data[i]: #cycles thru the names in each key
            accum = 0
            while accum != len(data[i][x])-1: #don't need the true/false entry at end for ave vals
                averages[i][accum] = averages[i][accum] + data[i][x][accum]
                accum = accum + 1
    # objective: making those totals into averages
    for i in data:
        count = (len(data[i])) # number of poke per type
        accum = 0
        while accum != len(averages[i]):
            averages[i][accum] = averages[i][accum] / count
            accum = accum + 1
    return averages, list_of_keys


def max_value_dictionary(averages, list_of_keys):
    """
    Description: creates a dictionary of types with the highest averages organized by query and a list of
    Parameters: averages, list_of_keys
    Returns: max_val_dictionary, max_val
    Pre-condition: averages will be dictionary and max_val will be list
    Post-Condition: max_val_dictionary will contain top types organized by query and max_val will be list
    """
    # objective: create a max value list
    max_val = [0,0,0,0,0,0,0,0]
    for i in averages:
        accum = 0 #resets with each iteration across ave dict.
        while accum != len(averages[i]):
            if max_val[accum] < averages[i][accum]:
                max_val[accum] = averages[i][accum] #overwrites lower scores
            else:
```

```
                pass
            accum = accum + 1
    # objective: make a dictionary of the max-value winners (restricted to areas included in queries)
    max_val_dictionary = {'total': [], 'hp': [], 'attack': [],'defense': [],'specialattack': [],'special
    accum = 0
    for x in max_val_dictionary:
        tally = 0 #tally exists to correlate the resp. key in keychain to i in averages
        for i in averages:
            if max_val[accum] == averages[i][accum]:
                max_val_dictionary[x] = max_val_dictionary[x] + [list_of_keys[tally]]
            tally = tally + 1
        accum = accum + 1
    return max_val_dictionary, max_val


def handle_queries(max_val_dictionary, max_val):
    """
    Description: handles queries by looping until empty enter is entered and processes valid queries, pri
    Parameters: max_val_dictionary, max_val
    Returns: none
    Pre-condition: max_val_dictionary is dictionary, max_val is list
    Post-Condition: responses will print and loop will continue until user enters empty enter
    """
    # objective: start a validation loop which will run as long as query is not empty space
    query_tuple = ('a',)
    validation_tuple = ('',) #initializers
    while validation_tuple != query_tuple:
        # objective: based on query, print max val
        query = str(input())
        query = query.lower()
        query_tuple = (query,) # critical to our validation loop, started above
        if query == 'total':
            index = 0
        elif query == 'hp':
            index = 1
        elif query == 'attack':
            index = 2
        elif query == 'defense':
            index = 3
        elif query == 'specialattack':
            index = 4
        elif query == 'specialdefense':
            index = 5
        elif query == 'speed':
            index = 6
        else:
            pass
        if query in max_val_dictionary:
            accum = 0
            poke_type = '' #initalize arbitrary value
            # objective: getting the appropriate string form of type in order to print
            while accum != len(max_val_dictionary[query]):
                if accum != 0:
                    poke_type = poke_type + ' & ' + max_val_dictionary[query][accum]
                else:
                    poke_type = max_val_dictionary[query][accum]
                accum = accum + 1
            print("{}: {}".format(poke_type,max_val[index]))


main()
```