Homework Turnin

Name:	Alexander A Miller
Email:	alexandermiller@email.arizona.edu
Section:	1E
Course:	CS 120 17au
Assignment:	hw4
Receipt ID:	8a0f81ed488120108310bff96a6c6241

Turnin Successful!

The following file(s) were received:

biodiversity.py (7566 bytes)

```
"""
File: biodiversity.py
Author: Alexander Miller
Purpose: looking at density of fauna and flora (separately) in nat'l parks
* density defined as diversity / area
* read in filename concerning park information from input
    * organize pinfo into dict that maps park name to tuple containing info
    * info includes area of park, and flora and fauna counts
* read in filename concerning species information from input
    * check the category section for flora or fauna
        * algae, fungi, nonvascular plant, vascular plant: flora
        * amphibian, bird, crab/lobster/shrimp, fish, insect,\
        invertebrate, mammal, reptile, slug/snail,\
        spider/scorpion: fauna
        * other: ignore
* make tallies for each park about its fauna and flora
* be able to handle systems where park has no species listed\
    or where species has no park listed
    * in these cases (either park not in parklist or park information empty)
* print for all parks in pinfo the number of flora and fauna per acre (see output)
* USE TUPLE
* CHECK CASE SENSITIVITY
"""

def main():
    parks = str(input())
    species = str(input())
    filename = parks
    file_list, carbon_copy_list = readfiles(filename)
    assert type(file_list) == list
    assert type(carbon_copy_list) == list
    file_dictionary = dictionary_builder(file_list, carbon_copy_list, filename)
    assert type(file_dictionary) == dict
    filename = species
    file_list, carbon_copy_list = readfiles(filename)
    # not using carbon_copy_list on this go round
    # included for continuity
    file_dictionary = species_handler(file_dictionary, file_list)
    file_dictionary = find_averages(file_dictionary)
    assert type(file_dictionary) == dict
    print_fn(file_dictionary)


def readfiles(filename):
    """
    Description: reads in information based on filename passed to it
```

```
    Parameters: filename
    Returns: file_list, carbon_copy_list
    Pre-condition: filename exists and is a csv
    Post-Condition: will have a list of contents and a case-sensitive carbon copy
    """
    ### ASSUMPTION: input file is a CSV; file exists
    openfile = open(filename,'r')
    file_list = []
    carbon_copy_list = [] # using a carbon copy version which is case sensitive\
    # in order to maintain integrity of original spelling for printing
    for line in openfile:
        # objective: remove any white space from ends and make lower case
        line = line.strip()
        carbon_copy = line
        line = line.lower()
        # objective: remove empty lines
        if len(line) > 0:
            file_list.append(line.split(','))
            interim_copy = carbon_copy.split(',')
            carbon_copy_list.append(interim_copy[0])
    openfile.close()
    return file_list, carbon_copy_list



def dictionary_builder(file_list, carbon_copy_list, filename):
    """
    Description: builds our main operating dictionary
    Parameters: file_list, carbon_copy_list, filename
    Returns: file_dictionary
    Pre-condition: lines in file had information in the proper locations
    Post-Condition: file_dictionary is a dictionary containing relevant
    area of park and placeholders for flora, fauna, and their averages
    """
    assert type(file_list) == list
    assert type(carbon_copy_list) == list
    file_dictionary = {}
    accum = 0
    while accum != len(file_list):
        ### ASSUMPTION: lines are organized with the information in the proper location
        # objective: only adjoin non-comment lines
        if file_list[accum][0] != '#' and '#' not in file_list[accum][0]:
            # handling first occurrences
            if file_list[accum][0] not in file_dictionary:
                # logging the case-sensitive key inside of itself \
                # for expedient recovery to print
                file_dictionary[file_list[accum][0]] = (carbon_copy_list[accum],) \
                                                       + (file_list[accum][2],)
            else: # handling multiple occurrences
                file_dictionary[file_list[accum][0]] = \
                file_dictionary[file_list[accum][0]] + (file_list[accum][2],)
        accum = accum + 1
    # objective: add 4 lists containg 0 to the end of each dictionary entry
    # in order to make tallies for flora and fauna, and their averages
    for i in file_dictionary:
        assert type(file_dictionary[i]) == tuple
        file_dictionary[i] = file_dictionary[i] + ([0],[0],[0],[0])
    return file_dictionary



def species_handler(file_dictionary, file_list):
    """
    Description: operates on the species information and adjoins to dictionary
    Parameters: file_dictionary, file_list
    Returns: file_dictionary
    Pre-condition: file_dictionary and file_list should be dict and list resp.
    Post-Condition: file_dictionary now contains flora and fauna counts
    """
    assert type(file_dictionary) == dict
    assert type(file_list) == list
    flora = ('algae','fungi','nonvascular plant','vascular plant')
    fauna = ('amphibian','bird','crab/lobster/shrimp','fish','insect',\
             'invertebrate','mammal','reptile','slug/snail',\
             'spider/scorpion')
    accum = 0
```

```
    while accum != len(file_list):
        # naming some variables in order to keep the nesting straight later
        assert type(file_list[accum]) == list
        respective_park = file_list[accum][0]
        respective_type = file_list[accum][1]
        if respective_park in file_dictionary:
            if respective_type in flora:
                file_dictionary[respective_park][2][0] = \
                    file_dictionary[respective_park][2][0] + 1
            if respective_type in fauna:
                file_dictionary[respective_park][3][0] = \
                    file_dictionary[respective_park][3][0] + 1
        accum = accum + 1
    return file_dictionary


def find_averages(file_dictionary):
    """
    Description: adds flora and fauna averages to dictionary
    Parameters: file_dictionary
    Returns: file_dictionary
    Pre-condition: file_dictionary already contains flora and fauna info
    Post-Condition: file_dictionary will now also contain flora and fauna averages
    """
    assert type(file_dictionary) == dict
    for i in file_dictionary:
        assert type(file_dictionary[i][2][0]) == int
        assert type(file_dictionary[i][3][0]) == int
        # objective: find average flora
        file_dictionary[i][4][0] = file_dictionary[i][2][0] / \
                                   int(file_dictionary[i][1])
        # objective: find average fauna
        file_dictionary[i][5][0] = file_dictionary[i][3][0] / \
                                   int(file_dictionary[i][1])
    return file_dictionary
```

```
def print_fn(file_dictionary):
    """
    Description: print function
    Parameters: file_dictionary
    Returns: none
    Pre-condition: file_dictionary has the averages as well as the park name
    Post-Condition: none
    """
    for i in file_dictionary:
        park_name = file_dictionary[i][0]
        flora_per_acre = file_dictionary[i][4][0]
        fauna_per_acre = file_dictionary[i][5][0]
        # controlling for only entries which contain relevant data
        if file_dictionary[i][2][0] != 0 \
           or file_dictionary[i][3][0] != 0:
            print("{} -- flora: {:f} per acre; fauna: {:f} per acre"\
                  .format(park_name, flora_per_acre, fauna_per_acre))
        else: # fail cases
            print("{} -- no data available".format(park_name))


main()
```

abundance.py (4575 bytes)

```
"""
File: abundance.py
Author: Alexander Miller
Purpose:
* examine the distribution of species across different
national parks and prints out the most widely distributed species
```

```
* read in the name of a file (sinfo) using input
    * for each line in sinfo use "scientific name" and park name to count
    the total number of national parks where species occurs
    * other data entries may be discarded
    * any line beginning with # should be discarded
    *CASE INSENSTIVE
    * close file
    * only reads the file once
* print out species that are found in the largest number of parks
* use a dictionary
* using the following output statement:
    print("{} -- {:d} parks".format(species_name, number_of_parks))
* asserts placed at beginning of fn to check any pre-conditions
* asserts at beginning of loops that compute value or transforms data which:
    * reflects compution of loop and is not directly tied to iteration condition (?)
        * asserts should check invariants within loop
    * think type(x) == b
    * where asserts are impossible/difficult can use comment instead:
        ### INVARIANT/ASSUMPTION: invariant/assumption
"""

def main():
    file_list = init_fn()
    file_dictionary = data_organization(file_list)
    assert type(file_dictionary) == dict
    success_list, max_val = data_processing(file_dictionary)
    assert type(success_list) == list
    print_fn(success_list, max_val)

def init_fn():
    """
    Description: gathers information from file
    Parameters: none
    Returns: file_list
    Pre-condition: file is CSV and exists
    Post-Condition: file_list is list
    """
    filename = input()
    ### ASSUMPTION: input file is a CSV; file exists
    openfile = open(filename,'r')
    file_list = []
    for line in openfile:
        # objective: remove any white space from ends and make lower case
        line = line.strip()
        line = line.lower()
        # objective: remove empty lines
        if len(line) > 0:
            file_list.append(line.split(','))
    openfile.close()
    return file_list

def data_organization(file_list):
    """
    Description: organizes information into dictionary
    Parameters: file_list
    Returns: file_dictionary
    Pre-condition: file_list is a list
    Post-Condition: file_dictionary is a dictionary
    """
    assert type(file_list) == list
    file_dictionary = {}
    accum = 0
    while accum != len(file_list):
        ### ASSUMPTION: lines are organized with the information in the proper location
        # objective: only adjoin non-comment lines
        if file_list[accum][0] != '#' and '#' not in file_list[accum][0]:
            # handling first occurrences
            if file_list[accum][2] not in file_dictionary:
                # logging the key inside of itself for expedient recovery
                file_dictionary[file_list[accum][2]] = [file_list[accum][2]] \
                                                       + [file_list[accum][0]]
            # handling multiple occurrences
            else:
                file_dictionary[file_list[accum][2]] = \
                file_dictionary[file_list[accum][2]] + [file_list[accum][0]]
        accum = accum + 1
```

```
    return file_dictionary


def data_processing(file_dictionary):
    """
    Description: processes dictionary to find most widely\
    distributed species
    Parameters: file_dictionary
    Returns: success_list, max_val
    Pre-condition: file_dictionary is a dictionary
    Post-Condition: success_list is a list; max_val is an integer
    """
    assert type(file_dictionary) == dict
    success_list = []
    max_val = 0
    for i in file_dictionary:
        ### ASSUMPTION: none (this will still operate even if dict is empty)
        if len(file_dictionary[i]) > max_val:
            max_val = len(file_dictionary[i])
    for x in file_dictionary:
        ### ASSUMPTION: none (this will still operate even if dict is empty)
        if len(file_dictionary[x]) == max_val:
            success_list = success_list + [file_dictionary[x][0]]
    return success_list, max_val

def print_fn(success_list, max_val):
    """
    Description: prints out successful entries in appropriate format
    Parameters: success_list, max_val
    Returns: none
    Pre-condition: success_list is a list; max_val is an integer
    Post-Condition: none
    """
    assert type(success_list) == list
    assert type(max_val) == int
    accum = 0
    while accum < len(success_list):
        print("{} -- {:d} parks".format(success_list[accum], max_val - 1))
        accum = accum + 1

main()
```