"""

Project for Week 4 of "Python Data Analysis".

Processing CSV files with baseball stastics.

Be sure to read the project description page for further information

about the expected behavior of the program.

"""

import csv

##

## Provided code from Week 3 Project

##

def read\_csv\_as\_list\_dict(filename, separator, quote):

"""

Inputs:

filename - name of CSV file

separator - character that separates fields

quote - character used to optionally quote fields

Output:

Returns a list of dictionaries where each item in the list

corresponds to a row in the CSV file. The dictionaries in the

list map the field names to the field values for that row.

"""

table = []

with open(filename, newline='') as csvfile:

csvreader = csv.DictReader(csvfile, delimiter=separator, quotechar=quote)

for row in csvreader:

table.append(row)

return table

def read\_csv\_as\_nested\_dict(filename, keyfield, separator, quote):

"""

Inputs:

filename - name of CSV file

keyfield - field to use as key for rows

separator - character that separates fields

quote - character used to optionally quote fields

Output:

Returns a dictionary of dictionaries where the outer dictionary

maps the value in the key\_field to the corresponding row in the

CSV file. The inner dictionaries map the field names to the

field values for that row.

"""

table = {}

with open(filename, newline='') as csvfile:

csvreader = csv.DictReader(csvfile, delimiter=separator, quotechar=quote)

for row in csvreader:

rowid = row[keyfield]

table[rowid] = row

return table

##

## Provided formulas for common batting statistics

##

# Typical cutoff used for official statistics

MINIMUM\_AB = 500

def batting\_average(info, batting\_stats):

"""

Inputs:

batting\_stats - dictionary of batting statistics (values are strings)

Output:

Returns the batting average as a float

"""

hits = float(batting\_stats[info["hits"]])

at\_bats = float(batting\_stats[info["atbats"]])

if at\_bats >= MINIMUM\_AB:

return hits / at\_bats

else:

return 0

def onbase\_percentage(info, batting\_stats):

"""

Inputs:

batting\_stats - dictionary of batting statistics (values are strings)

Output:

Returns the on-base percentage as a float

"""

hits = float(batting\_stats[info["hits"]])

at\_bats = float(batting\_stats[info["atbats"]])

walks = float(batting\_stats[info["walks"]])

if at\_bats >= MINIMUM\_AB:

return (hits + walks) / (at\_bats + walks)

else:

return 0

def slugging\_percentage(info, batting\_stats):

"""

Inputs:

batting\_stats - dictionary of batting statistics (values are strings)

Output:

Returns the slugging percentage as a float

"""

hits = float(batting\_stats[info["hits"]])

doubles = float(batting\_stats[info["doubles"]])

triples = float(batting\_stats[info["triples"]])

home\_runs = float(batting\_stats[info["homeruns"]])

singles = hits - doubles - triples - home\_runs

at\_bats = float(batting\_stats[info["atbats"]])

if at\_bats >= MINIMUM\_AB:

return (singles + 2 \* doubles + 3 \* triples + 4 \* home\_runs) / at\_bats

else:

return 0

##

## Part 1: Functions to compute top batting statistics by year

##

def filter\_by\_year(statistics, year, yearid):

"""

Inputs:

statistics - List of batting statistics dictionaries

year - Year to filter by

yearid - Year ID field in statistics

Outputs:

Returns a list of batting statistics dictionaries that

are from the input year.

"""

filtered = []

filtered = list(filter(lambda x: x[yearid] == str(year), statistics))

return(filtered)

def top\_player\_ids(info, statistics, formula, numplayers):

"""

Inputs:

info - Baseball data information dictionary

statistics - List of batting statistics dictionaries

formula - function that takes an info dictionary and a

batting statistics dictionary as input and

computes a compound statistic

numplayers - Number of top players to return

Outputs:

Returns a list of tuples, player ID and compound statistic

computed by formula, of the top numplayers players sorted in

decreasing order of the computed statistic.

"""

formulacalc = []

for row in range(0, len(statistics)):

playerid = statistics[row][info["playerid"]]

stat = (formula(info, statistics[row]))

if stat == 0:

pass

else:

formulacalc.append((playerid, stat))

#sorts by second dield which is the number

formulacalc.sort(key=lambda pair: pair[1], reverse=True)

#reduces the list by the number required

finallist = formulacalc[0:numplayers]

return finallist

def lookup\_player\_names(info, top\_ids\_and\_stats):

"""

Inputs:

info - Baseball data information dictionary

top\_ids\_and\_stats - list of tuples containing player IDs and

computed statistics

Outputs:

List of strings of the form "x.xxx --- FirstName LastName",

where "x.xxx" is a string conversion of the float stat in

the input and "FirstName LastName" is the name of the player

corresponding to the player ID in the input.

"""

#reads in names

mastfile = read\_csv\_as\_list\_dict(info["masterfile"], info["separator"], info["quote"])

new\_list = []

for idx in range(0, len(top\_ids\_and\_stats)):

for mastrow in range(0, len(mastfile)): #findsmatches in playerid

if top\_ids\_and\_stats[idx][0] == mastfile[mastrow][info["playerid"]]:

playername = mastfile[mastrow][info["firstname"]] + " " \

+ mastfile[mastrow][info["lastname"]]

stat = top\_ids\_and\_stats[idx][1]

# f is used to restrict number of characters after decimal place

new\_list.append((str(f'{stat:.3f}') + ' --- ' + playername))

return new\_list

def compute\_top\_stats\_year(info, formula, numplayers, year):

"""

Inputs:

info - Baseball data information dictionary

formula - function that takes an info dictionary and a

batting statistics dictionary as input and

computes a compound statistic

numplayers - Number of top players to return

year - Year to filter by

Outputs:

Returns a list of strings for the top numplayers in the given year

according to the given formula.

"""

# read the csv for stats

statistics = read\_csv\_as\_list\_dict(info["battingfile"], info["separator"], info["quote"])

#filter by year

stat\_by\_year = filter\_by\_year(statistics, year, info["yearid"])

#calcualte the stat and filter by numplayers

topplayers = top\_player\_ids(info, stat\_by\_year, formula, numplayers)

#matches the names in master file to the player id

topplayers\_names = lookup\_player\_names(info, topplayers)

return topplayers\_names

##

## Part 2: Functions to compute top batting statistics by career

##

def aggregate\_by\_player\_id(statistics, playerid, fields):

"""

Inputs:

statistics - List of batting statistics dictionaries

playerid - Player ID field name

fields - List of fields to aggregate

Output:

Returns a nested dictionary whose keys are player IDs and whose values

are dictionaries of aggregated stats. Only the fields from the fields

input will be aggregated in the aggregated stats dictionaries.

"""

new\_list = {}

for row in range(0, len(statistics)):

if statistics[row][playerid] in new\_list.keys():

for col in fields:

#once matching stat found converts to int for calc

player = statistics[row][playerid]

value = int(new\_list[player][col])

value += int(statistics[row][col])

new\_list[player][col] = int(value)

value = 0

#if player is not in new list adds new row and copies in columns

else:

newplayer = statistics[row][playerid]

new\_list[newplayer] = {}

new\_list[newplayer][playerid] = newplayer

for newcol in fields:

new\_list[newplayer][newcol] = int(statistics[row][newcol])

return new\_list

def compute\_top\_stats\_career(info, formula, numplayers):

"""

Inputs:

info - Baseball data information dictionary

formula - function that takes an info dictionary and a

batting statistics dictionary as input and

computes a compound statistic

numplayers - Number of top players to return

"""

statistics = read\_csv\_as\_list\_dict(info["battingfile"], \

info["separator"], info["quote"])

#calculate the sum of each stat for each player

player\_career\_stats = aggregate\_by\_player\_id(statistics, \

info["playerid"], info["battingfields"])

#had a bit of trouble. owl list wanted the agregate list in a weird way

#ended up having to bring the dict down to lists that the top\_player\_ids

#and the lookup\_player\_names would work with

career\_list = []

final\_list = []

for row in player\_career\_stats:

career\_list = [player\_career\_stats[row]]

final\_list.append(career\_list)

career\_stats = []

for key in range(0, len(final\_list)):

#temp = final\_list[key][0]

career\_stats.append(final\_list[key][0])

#print(career\_stats)

career\_stats = top\_player\_ids(info, career\_stats, formula, numplayers)

#career\_stats.sort(key=lambda pair: pair[1], reverse=True)

topplayers\_names = lookup\_player\_names(info, career\_stats)

#return []

return topplayers\_names

##

## Provided testing code

##

def test\_baseball\_statistics():

"""

Simple testing code.

"""

#

# Dictionary containing information needed to access baseball statistics

# This information is all tied to the format and contents of the CSV files

#

baseballdatainfo = {"masterfile": "Master\_2016.csv", # Name of Master CSV file

"battingfile": "Batting\_2016.csv", # Name of Batting CSV file

"separator": ",", # Separator character in CSV files

"quote": '"', # Quote character in CSV files

"playerid": "playerID", # Player ID field name

"firstname": "nameFirst", # First name field name

"lastname": "nameLast", # Last name field name

"yearid": "yearID", # Year field name

"atbats": "AB", # At bats field name

"hits": "H", # Hits field name

"doubles": "2B", # Doubles field name

"triples": "3B", # Triples field name

"homeruns": "HR", # Home runs field name

"walks": "BB", # Walks field name

"battingfields": ["AB", "H", "2B", "3B", "HR", "BB"]}

baseballdatainfo = {"masterfile": "master1.csv", # Name of Master CSV file

"battingfile": "batting1.csv", # Name of Batting CSV file

"separator": ",", # Separator character in CSV files

"quote": '"', # Quote character in CSV files

"playerid": "player", # Player ID field name

"firstname": "firstname", # First name field name

"lastname": "lastname", # Last name field name

"yearid": "year", # Year field name

"atbats": "atbats", # At bats field name

"hits": "hits", # Hits field name

"doubles": "doubles", # Doubles field name

"triples": "triples", # Triples field name

"homeruns": "homers", # Home runs field name

"walks": "walks", # Walks field name

"battingfields": ["atbats", "hits", "doubles", \

"triples", "homers", "walks"]}

# test functions on batting1.csv

print("Top 5 batting averages in 2020")

top\_batting\_average\_2020 = compute\_top\_stats\_year(baseballdatainfo, batting\_average, 5, 2020)

for player in top\_batting\_average\_2020:

print(player)

print("")

print("Top 5 batting averages in 1923")

top\_batting\_average\_1923 = compute\_top\_stats\_year(baseballdatainfo, batting\_average, 5, 1923)

for player in top\_batting\_average\_1923:

print(player)

print("")

print("Top 10 batting averages in 2010")

top\_batting\_average\_2010 = compute\_top\_stats\_year(baseballdatainfo, batting\_average, 10, 2010)

for player in top\_batting\_average\_2010:

print(player)

print("")

print("Top 10 on-base percentage in 2010")

top\_onbase\_2010 = compute\_top\_stats\_year(baseballdatainfo, onbase\_percentage, 10, 2010)

for player in top\_onbase\_2010:

print(player)

print("")

print("Top 10 slugging percentage in 2010")

top\_slugging\_2010 = compute\_top\_stats\_year(baseballdatainfo, slugging\_percentage, 10, 2010)

for player in top\_slugging\_2010:

print(player)

print("")

# You can also use lambdas for the formula

# This one computes onbase plus slugging percentage

print("Top 10 OPS in 2010")

top\_ops\_2010 = compute\_top\_stats\_year(baseballdatainfo,

lambda info, stats: (onbase\_percentage(info, stats) +

slugging\_percentage(info, stats)),

10, 2010)

for player in top\_ops\_2010:

print(player)

print("")

print("Top 20 career batting averages")

top\_batting\_average\_career = compute\_top\_stats\_career(baseballdatainfo, batting\_average, 4)

for player in top\_batting\_average\_career:

print(player)

print("")

# Make sure the following call to test\_baseball\_statistics is

# commented out when submitting to OwlTest/CourseraTest.

#test\_baseball\_statistics()