**CIS-298 Intro to Python**

**With Professor Robert Mann**

**HW #7**

**Student: Demetrius Johnson**

**13 March 2023**

**Due: 17 March 2023 at 3:30pm**

# #Author: Demetrius Johnson

#Author: Demetrius Johnson

#CIS-298 Intro to Python with Professor Robert Mann

#HW7 - Working with Files

#Remember to copy/paste your code for each question separately.

#Then, follow it with a snippet showing the output.

#Due: Thu March 16, 2023 3:30pm

# **Description**

For this assignment, you will be processing a CSV file containing scores from a game tournament.

# **Requirements**

## You shall manually create a CSV file to store names and scores.  The file format is last name, first name, score 1, score 2, score 3, score 4.  Scores range from 1-50, or -1 if no score available. Enter at least 7 players into your data file.

#You shall manually create a CSV file to store names and scores.

#The file format is last name, first name, score 1, score 2, score 3, score 4.

#Scores range from 1-50, or -1 if no score available.

#Enter at least 7 players into your data file.

import random

#I will recycle my code for random number generator from hw6 and make some adaptations

def get\_random\_num(positive\_range=50, negative\_range=1):

#random() function returns a number between 0 and 1

#Multiply pos\_range\*random() and subtract neg\_range\*random() in order to make the values have a range of [pos\_range,neg\_range]

random\_num = positive\_range \* random.random() - negative\_range \* random.random()

return random\_num if random\_num >=1 else -1 #increase chances of -1 being returned --> return -1 for all numbers < 1

#use list comprehension and random number function to create csv (comma separated values) string

number\_of\_players = 700

all\_player\_names\_and\_scores = [

#03d simply means fill 0's (if necessary) and make the integer length of 3...

"P\_%03d\_last,P\_%03d\_first,%d,%d,%d,%d\n" #added a newline at the end so when it is written to file each player+stats are written on their own line

%(iterator,

iterator,

get\_random\_num(),

get\_random\_num(),

get\_random\_num(),

get\_random\_num()) #generate 4 random numbers [-1 to 50, 0 not included] for all players

for iterator in range(number\_of\_players)]

#use join() to make player\_names list into a single string with each element separated by ''

#we need player\_names string without the final '\n' char, so get range [0,last\_char)

all\_player\_names\_and\_scores = ''.join(all\_player\_names\_and\_scores)

all\_player\_names\_and\_scores = all\_player\_names\_and\_scores[:-1]

#default path is current working directory (environment) of where the current

#Python process is running in (so no need to specify the path in this case, just file name):

#open for reading (r) and writing without truncating (+) and in text mode (t) = 'r+t'

#open for writing (and truncating the file) and in text mode = 'wt'

player\_scores\_file = open("player\_names\_and\_scores.csv", 'wt')

for player in all\_player\_names\_and\_scores:

player\_scores\_file.write(player)

player\_scores\_file.close() #done with writing in player scores; close file.

Text

Description automatically generated

## Read in the tournament results data file (created above).

#Read in the tournament results data file (created above):

player\_scores\_file = open("player\_names\_and\_scores.csv", 'rt')

player\_scores\_data = player\_scores\_file.read() #get all data from the file (string) then close file.

player\_scores\_file.close()

Here is the output of my list from reading in the data from the file (for 3o generic players):

Background pattern

Description automatically generated with low confidence

## Players with one or more -1 scores are disqualified and their names are written to an output file using the format: name “didn’t participate in round(s) “ and list the rounds in which the player did not participate (those with a -1 value). Raise a custom exception to process disqualifying rounds.

 #Players with one or more -1 scores are disqualified and their names are written to an output file using the format:

# "NAME didn't participate in round(s)", and list the rounds in which the player did not participate (those with a -1 value).

#Raise a custom exception to process disqualifying rounds.

#create custom exception class with Exception as base (parent) class

class DisqualifiedException(Exception):

#raise this exception if a player has a round with a score == -1

#note: all default constructors and other functions and variables from base class are inherited.

pass

#reset this file so that data from last program run will be truncated:

#use this to track disqualified players:

dq\_players\_file = open("dq\_player\_list.csv", 'w')

dq\_players\_file.close()

dq\_players\_file = open("dq\_player\_list.csv", 'at') #now reopen file in append mode, and in text mode

#use this to track only qualified players:

qualified\_players\_file = open("qualified\_player\_list.csv", 'w')

qualified\_players\_file.close()

qualified\_players\_file = open("qualified\_player\_list.csv", 'at') #now reopen file in append mode, and in text mode

check\_rounds = [False,False,False,False]

player\_scores\_data = player\_scores\_data.split('\n') #first we need to take out newline characters from string and create a list of player+data strings

for player in player\_scores\_data: #now we need to check each player from the list and check if they have a round with -1

for score in range(2,6): #elements 2-5 are scores (as strings, and we need to cast as int to check)

if int(player.split(',')[score]) == -1:

check\_rounds[score - 2] = True

else:

check\_rounds[score - 2] = False

try:

#cast as set for faster search since sets use hash tables to store items (and thus no duplicate items stored), so lookup is faster.

if True in set(check\_rounds):

raise DisqualifiedException("Exception raised: Found a disqualified player! Updating Disqualification Database File!")

else:

qualified\_players\_file.write(player + '\n')

except DisqualifiedException as dq\_exception: #catch the thrown exception and store the object (label/alias the object) in dq\_exception

print(dq\_exception) # \_\_str\_\_ function in Exception objects allows args string array/list of the Exeception object to be printed directly

#need round\_num+1 because we do not want element 0 to count as a round, but start from round 1;

#also, we need to enumerate so we know the location (round) of the True elements in rounds array:

dq\_rounds = [(round\_num + 1) for round\_num,dq\_val in enumerate(check\_rounds) if dq\_val == True] #using list comprehension...

dq\_players\_file.write("Player: %s, %s Disqualified at round(s): %s\n"

%((player.split(',')[0]),

(player.split(',')[1]),

str(dq\_rounds)))

print(

"Player: %s, %s Disqualified at round(s): "

%((player.split(',')[0]),

(player.split(',')[1])),

dq\_rounds

)

#close files:

qualified\_players\_file.close()

dq\_players\_file.close()

Text

Description automatically generated with medium confidence

Above, I am showing how my custom exception is working…

Text

Description automatically generated

Showing my disqualified player list file including the rounds they were disqualified…

This is the same thing shown to the user when exception is raised:

Text

Description automatically generated

## Generate and display to the user the following:

### Person(s) with highest total score from all four rounds (score 1 – score 4) and their score (all players). Include -1 scores.

### Person(s) with lowest total score from all four rounds and their score (all players). Include -1 scores.

#Generate and display to the user the following:

#Person(s) with highest total (sum) score from all four rounds (score 1 – score 4) and their score (compared to all players). Include -1 scores.

#Person(s) with lowest total score from all four rounds and their score (compared to all players). Include -1 scores.

#print all player scores:

for player in player\_scores\_data:

print(player)

#now get max and min sum scores:

maxscore = -5 #initiate max with smallest possible score (I may or may not use this fact)

minscore = 200 #initiate min with largest possible score (I may or may not use this fact)

#track players with max or mnin score:

max\_sum\_players = []

min\_sum\_players = []

player\_scores\_SUM = []

#use this loop to collect sums of the scores of all players, then we can find the player with max sum scores:

#note: we need to strip - sign in case we encounter a negative number (string with - and digits)

#in order to use isdigit function to still consider -# strings as numeric:

#I will make this a function to recycle later in my code...

def get\_player\_scores\_SUM (scores\_data):

sums = []

for player in scores\_data:

rounds = [ int(score) for score in player.split(',') if score.strip('-').isdigit() ] #use list comprhension to extract the scores of each player, then sum them

sums += [[player, sum(rounds)]]

return sums

player\_scores\_SUM = get\_player\_scores\_SUM(player\_scores\_data)

#sort based on key = list\_element[2], since the third element is where the sum score is stored for each player in the player\_scores\_SUM list

#note: setting key to sort with key = lambda is simply creating a function with no label,

#passing list\_element to it, and returning whatever list\_element[2] returns, which is assigned to key;

#underneath, key is really a pointer to the third element in each list\_element:

player\_scores\_SUM\_max = sorted(player\_scores\_SUM, key = lambda list\_element: list\_element[1], reverse=True) #reverse=True == descending order == max to min

player\_scores\_SUM\_min = sorted(player\_scores\_SUM, key = lambda list\_element: list\_element[1])

#now take advantage of our sorted lists to get max and min sums, and also the name(s) of the player(s) who have the max and min scores:

#front of each sorted list will be the max or min score, respectively:

maxscore = player\_scores\_SUM\_max[0][1]

minscore = player\_scores\_SUM\_min[0][1]

for player in player\_scores\_SUM\_max:

if player[1] == maxscore:

max\_sum\_players += [player[0]] #add player to max score list

else:

break #have reached the end of players with the max score

for player in player\_scores\_SUM\_min:

if player[1] == minscore:

min\_sum\_players += [player[0]] #add player to min score list

else:

break #have reached the end of players with the min score

#now print player(s) with max score:

print("The player(s) with the max score sum of %d is(are): "%maxscore)

for player in max\_sum\_players:

print(player)

print()#newline

#now print player(s) with min score:

print("The player(s) with the min score sum of %d is(are): "%minscore)

for player in min\_sum\_players:

print(player)

print()#newline

Text

Description automatically generated

### The median and mean (to one decimal place) overall total scores using only qualified participants

#The median and mean (to one decimal place) overall total scores using only qualified participants:

qualified\_players\_file = open("qualified\_player\_list.csv", 'rt') #now reopen file in text mode

qualified\_players\_list = qualified\_players\_file.read().split('\n')

qualified\_players\_list.pop() #need to delete last element, which is just storing a empty string from file (I could truncate last char in file but this is faster)

player\_scores\_SUM = get\_player\_scores\_SUM(qualified\_players\_list)

import statistics

#sorting just for debugging and checking my program for correctness:

sum\_scores\_only = sorted([sums[1] for sums in player\_scores\_SUM])

median\_sum = round(statistics.median(sum\_scores\_only), 1)

mean\_sum = round(statistics.mean(sum\_scores\_only), 1)

#need to use %.1f to specify to string replacement % operator that I want it to only print out the number with precision of 1 decimal place.

print("[Excluding dq'd players] The median of player score sums is %.1f and the mean is %.1f "%(median\_sum,mean\_sum))

A computer screen capture

Description automatically generated with low confidence

Here is a bigger screenshot for seeing easier; also notice above -1 scores omitted from player scores:



### The mean score for each round, only using participants with valid scores (1-50) in that round, shown to one decimal place.

### Ex. Bob Mann has scores 48, -1, 13, -1 so would be included in average score calculation for rounds 1 & 3 only

 def get\_round\_scores(player\_scores\_data, selected\_round):

selected\_round\_scores = []

selected\_round += 1 #elements 0 and 1 is first and last name, then round scores start at element 2

for player in player\_scores\_data: #now we need to check the selected round for each player and exclude -1 scores

if int(player.split(',')[selected\_round]) != -1:

selected\_round\_scores += [int(player.split(',')[selected\_round])]

return selected\_round\_scores

#now store the array of scores for each individual round (4 rounds) in round\_scores

round\_scores = [get\_round\_scores(player\_scores\_data,round\_x) for round\_x in range(1,5)]

for round\_number, print\_round\_scores in enumerate(round\_scores): #enumerate returns list of tuples: (element number starting at 0, element value)

print("Scores [excluding dq'd (-1) scores] for round %d are: "%(round\_number + 1), print\_round\_scores)

#now acquire the mean for each round

mean\_round\_scores = [statistics.mean(scores) for scores in round\_scores]

#finally, print mean for each round to one decimal place:

for round\_number, print\_round\_mean in enumerate(mean\_round\_scores): #enumerate returns list of tuples: (element number starting at 0, element value)

print("Mean score [excluding dq'd (-1) scores] for round %d is: "%(round\_number + 1), round(print\_round\_mean, 1))

A computer screen capture

Description automatically generated with low confidence

Here is a bigger screenshot for seeing easier; also notice above -1 scores omitted from player scores:

Text

Description automatically generated

### Write to a second output file the name and total score (exclude -1 values) of all participants, highest score to lowest score. In case of ties, list tying participants in alphabetical order, by last name then first name.

#Write to a second output file the name and total score (exclude -1 values) of all participants,

#highest score to lowest score. In case of ties,

#list tying participants in alphabetical order, by last name then first name.

qualifying\_total\_scores\_players\_file = open("player\_names\_and\_qualifying\_total\_scores.csv", 'wt')

player\_scores\_and\_total = []

for player in player\_scores\_data: #now we need to check each player from the list and check if they have a round with -1

scores = []

for score in range(2,6): #elements 2-5 are scores (as strings, and we need to cast as int to check)

scores += [int(player.split(',')[score])]

scores = sorted(scores, reverse = True) #reverse=True == descending order == max to min == highest to lowest score

while scores[-1] == -1: #pop all -1 scores since they are not qualifying scores that should count towards the sum

scores.pop()

#make lname and fname a single element for sorting alphabetically later on...

player\_scores\_and\_total += [[player.split(',')[0] + ', ' + player.split(',')[1],

scores,

sum(scores)]]

#Note on sorting based on mutliple criteria (hierachical sorting):

'''

Sort Stability and Complex Sorts

Sorts are guaranteed to be stable. That means that when multiple records have the same key, their original order is preserved.

data = [('red', 1), ('blue', 1), ('red', 2), ('blue', 2)]

sorted(data, key=itemgetter(0))

[('blue', 1), ('blue', 2), ('red', 1), ('red', 2)]

Notice how the two records for blue retain their original order so that ('blue', 1) is guaranteed to precede ('blue', 2).

This wonderful property lets you build complex sorts in a series of sorting steps.

For example, to sort the student data by descending grade and then ascending age, do the age sort first and then sort again using grade:

s = sorted(student\_objects, key=attrgetter('age')) # sort on secondary key, (default = ascending --> reverse=False)

sorted(s, key=attrgetter('grade'), reverse=True) # now sort on primary key, descending (reverse=True)

[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

\*\*\* IT IS EXTREMELY IMPORTNAT TO SORT INNER (SECONDARY) KEYS FIRST

AND WORK YOUR WAY OUT TO THE PRIMARY KEY

IN ORDER TO SORT HIERARCHICALLY IN THE ORDER YOU DESIRE\*\*\*

'''

#now sort players based on total score (highest to lowest), then lastname and firstname (alphabetical lowest to highest) and write to output file:

#To achieve this, we need to sort based on lname\_fname first, then sort based on totals (since order is preserved with the sorted function, then

#the lname\_fname orders will be preserved when tie breakers occur during the primary sort.

player\_scores\_and\_total = sorted(player\_scores\_and\_total, key = lambda list\_element: list\_element[0], reverse=False) # lname\_fname, ascending order

player\_scores\_and\_total = sorted(player\_scores\_and\_total, key = lambda list\_element: list\_element[2], reverse=True) #total, desecnding order

for player in player\_scores\_and\_total:

qualifying\_total\_scores\_players\_file.write("%s scores and total (excluding dq'd rounds): %s, total=%d\n"%

(player[0], #lname, fname

str((player[1])), #scores

player[2]) #total

)

qualifying\_total\_scores\_players\_file.close()

Table

Description automatically generated with medium confidence

Above is the output file for the total scores sorted in descending order, tie breaker sort by last name then first name.

# I/O Files (name + screenshot + description)

## dq\_player\_list.csv

Output file containing disqualified players and the rounds they were disqualified in.

Text

Description automatically generated

## player\_names\_and\_qualifying\_total\_scores.csv

Output file containing all player data, but including only qualifying scores (non -1 rounds)

Text

Description automatically generated

## player\_name\_and\_scores.csv

Input file used for this lab containing player data

Text

Description automatically generated

## qualified\_player\_list.csv

Output file containing list of only qualified players (players with non -1 score rounds) and their data.

Text

Description automatically generated

# Special notes:

* To run my script and generate (automatically) the input file, simply go to command prompt, navigate to the directory, and run the script (enter the script name; note: make sure default program is python for .py programs in Windows), then I/O files will be generated in the same directory that the script was ran in.
* If you do not want my automatically generated Input file to be used for the names and data, then go to the script and change the file to open toa different name:

Text

Description automatically generated

Note: if you would like to use the file I generated, then simply run the program to generate the file; then change the filename and change the above line for open() string.

Otherwise, I set my program to overwrite that input file with new random data for the number of players specified to generate.

* You can change the variable **number\_of\_players** at the very beginning of the script  in order to decide the number of players you want my program to generate for the input file that will be used. I did this for testing, as it makes it much easier to test by creating the file first with some random scores for x number of players. Then I simply reopen the file and read from it to complete this lab.
* You will notice that when a player is disqualified for multiple rounds, my program will properly list it and not include the scores in the parts of the program where they are not to be included (you will see this on the output screen and also in the output files generated); hint: try changing number of players to something like 700 or 1000. Note, for testing, I made player names include numbers as part of their name; so if you go above 9999 number of players, it will be harder to verify my program.
* I know it is late, but I worked extremely hard and I learned a lot and am becoming a great python programmer (despite hating it! C++ is the best!)