ASSIGNMENT 2

BIT 100

INTRODUCTION TO PROGRAMMING



Name : I Nyoman Surya Pradipta

Student Number : E1900344

divingAssoc.py

# Name : I Nyoman Surya Pradipta

# Student ID : E1900344

# Date : April 28, 2020

# This class defines the attributes of the object participants

# who take part in diving competitions.

class Diver:

"""The class that defines simple object types that

represent dive competitors."""

# The nextDiverID variable is a class variable that is

# shared with all instances / objects of Diver class.

# save the next diver ID, and initialize it with 1.

nextDiverID = 1

# The constructor method for initializing the creation of objects from

# the diver class.

# Constructor with name and score arguments to initialize

# the object's attributes with this parameter value.

def \_\_init\_\_(self, name, scores):

# Declare name variable.

# Use the word "self" which shows that "variable" belongs

# to the class "Diver".

self.name = name

# Declare scores variable.

self.scores = scores

# Access class variables with dot notation (.)

self.id = Diver.nextDiverID

# id is automatically generated using class variables.

Diver.nextDiverID += 1

# A method that reads the name variable value.

def getName(self):

return self.name

# A method that reads the id variable value

def getID(self):

return self.id

# A method that reads the scores variable value

def getScores(self):

return self.scores

# A method that updates value of a name variable

def setName(self, newName):

self.name = newName

# A method that compute a score that represents the average score

# of the 5 remaining scores, after removing

# the highest and lowest score.

def getAverageScore(self):

# Create object average.

average = self.scores

average = (sum(average) - max(average) -

min(average)) / (len(average) - 2)

# Format 2 decimal places after the comma.

return format(average, '.2f')

# A string method ( \_\_str\_\_ ) which return

# a single string containing the details of a competitor.

def \_\_str\_\_(self):

# method str.format()

return "{} ({}) with scores {}, and average score {}"\

.format(self.name, self.id, self.scores,self.getAverageScore())

# This class declares object which maintains a list of Competitor objects.

class Competition:

"""The class defines the object which is the container

of the Competitor object."""

# A constructor with one argument of type ‘str’ used to

# initialise the competition name.

def \_\_init\_\_(self, competitionName):

# Initialise an empty list for storing divers.

self.diverList = []

# Declare competition name variable.

self.competitionName = competitionName

# A method that reads the competition name variable.

def getCompetitionName(self):

return self.competitionName

# A method that reads the divers variable.

def getDiverList(self):

return self.diverList

# A method that updates value of a competition name variable.

def setCompetitionName(self, newCompetitionName):

self.competitionName = newCompetitionName

# A method that updates value of a divers variable.

def setDiverList(self, newDiverList):

self.diverList = newDiverList

# A method will save a reference to the Diver object.

def register(self, diverObject):

# The append () method adds items to diverList.

self.diverList.append(diverObject)

# A method that returns divers with an average score so far.

def highestAverageScoreSoFar(self):

# Create object highest.

highest = self.diverList[0]

# Divers that are stored in the diverList.

for diver in self.diverList:

if diver.getAverageScore() > highest.getAverageScore():

highest = diver

# Returns divers with the highest average score so far.

return highest

# Returns the number of divers stored in the collection

def noOfDivers(self):

return len(self.diverList)

# Returns the details of all divers, one per line.

def \_\_str\_\_(self):

# More than one diver.

if len(self.diverList) > 1:

strAll = "Divers: \n"

# Less than one diver.

else:

strAll = "Diver: \n"

# The number start with 1.

number = 1

# Items in diverList.

for i in self.diverList:

# Format display.

# Adding dot(.) after number.

strAll += str(number) + ". " + str(i) + "\n"

# Numbers added by 1.

number += 1

# Returns the details of all divers.

return strAll

# A method that accepts an integer representing the index

# where the diver is stored in the list.

def getDiver(self, index):

# The found diver with that index is returned.

if index < len(self.diverList):

return self.diverList[index]

# If this index is out of bounds, the method returns none.

else:

return "None"

# A method that save all diver information to file.

def saveToFile(self, fileName):

# Write file.

output = open(fileName, "w")

# Executing one per line in diverList.

for i in self.diverList:

# Initialization.

formatScore = ""

# Executing one per line in getScores.

for j in i.getScores():

# Adding comma(,) before scores

formatScore += "," + str(j)

# Save one per line to file.

print(str(i.getName()) + formatScore, file=output)

# Close file.

output.close()

# Print if success

print("Data successfully save to " + fileName)

# A method that load all the divers info from the file.

def readFromFile(self, fileName):

# Set the nextDiverID to 1 before reading the data from file.

Diver.nextDiverID = 1

# Read file.

openFile = open(fileName, "r")

# Read every line that split by comma(,).

readFile = openFile.readlines()

for line in readFile:

diverDataLine = line.split(",")

# 0 Index is Diver's name.

diverName = diverDataLine[0]

# 1 until 7 is Diver's score.

scoreAll = []

for i in range(1, len(diverDataLine)):

scoreDiver = eval(diverDataLine[i])

scoreAll.append(scoreDiver)

# Register Diver.

diverData = Diver(diverName, scoreAll)

self.register(diverData)

# Close file.

openFile.close()

# Print if success.

print("Data successfully loaded from " + fileName)

# A method that accept the criteria

# that determine the sorting order of the Divers collection.

def sortAllDivers(self, sortDiver):

# If input equal to "n"

if sortDiver.lower() == "n":

# A list for storing sort data objects.

sortData = []

# Add items to sortData list.

for h in self.diverList:

sortData.append(h)

# The number of sessions used to check data from the start.

for i in range(len(sortData)):

for j in range(len(sortData) - 1):

# Comparing data name.

if sortData[j + 1].getName() < sortData[j].getName():

temp = sortData[j]

sortData[j] = sortData[j + 1]

sortData[j + 1] = temp

# If input equal to "a"

elif sortDiver.lower() == "a":

# A list for storing sort data objects.

sortData = []

# Add items to sortData list.

for h in self.diverList:

sortData.append(h)

# The number of sessions used to check data from the start.

for i in range(len(sortData)):

for j in range(len(sortData) - 1):

# Comparing data average score.

if sortData[j + 1].getAverageScore() \

< sortData[j].getAverageScore():

temp = sortData[j]

sortData[j] = sortData[j + 1]

sortData[j + 1] = temp

return sortData

main.py

# Name : I Nyoman Surya Pradipta

# Student ID : E1900344

# Date : April 28, 2020

# Import all class from divingAssoc file.

from divingAssoc import \*

# A method for providing a user interface.

class DivingCompetition:

def \_\_init\_\_(self):

# Enter name of competition.

competitionTitle = input("Enter competition title: ")

# Create a single Competition object.

self.competition = Competition(competitionTitle)

self.main()

# Provides a selection menu for users.

def main(self):

choice = -1

while choice != 0:

# Name of competition.

print("\n" + self.competition.getCompetitionName())

print("~" \* 30)

# Menu.

print("1. Register Diver")

print("2. Display all divers")

print("3. Display current leader")

print("4. Update information of diver")

print("5. Display all divers, sorted according to name "

"or average score")

print("6. Save data to file")

print("7. Load data from file")

print("\n" + "0. Quit" + "\n")

# User input choices.

choice = eval(input("Your choice? "))

if choice == 0:

# If the number of divers in competition more than 0.

if self.competition.noOfDivers() > 0:

# Call the class competition

currentLeader = \

str(self.competition.highestAverageScoreSoFar())

# Print if success.

print("The winner is: " + currentLeader)

# If the number of divers in competition less than 0.

else:

print("Competition is cancelled due to "

"lack of response.")

elif choice == 1:

# Print text.

print("Adding diver")

# User input name of diver.

name = input("Name: ")

# Initialise an empty list for storing score.

score = []

# Executing data.

for i in range(7):

# Get input data diver from user.

scoreDiver = eval(input("Score " + str(i + 1) + ": "))

# Add data from scoreDiver to score

score.append(scoreDiver)

# Object diver.

dataDiver = Diver(name, score)

# Register diver.

self.competition.register(dataDiver)

# Print if success.

print("Addition success...")

elif choice == 2:

# If there are divers in the competition

if self.competition.noOfDivers() > 0:

# Print registered divers.

print(self.competition)

# If there's no diver data.

else:

# Print text.

print("No diver has signed up yet")

elif choice == 3:

# If there are divers in the competition

if self.competition.noOfDivers() > 0:

# Call the method highestAverageScoreSoFar.

currentLeader = \

str(self.competition.highestAverageScoreSoFar())

# Print the current leader.

print("Current leader is: " + currentLeader + "\n")

# If there's no diver data.

else:

print("No diver has signed up yet")

elif choice == 4:

# Print divers

print(self.competition)

# Enter the ID to be changed.

idUpdate = eval(input("Which diver to update? "))

# Update ID.

if idUpdate <= self.competition.noOfDivers():

print(self.competition.diverList[idUpdate - 1])

# Enter the name to be changed.

nameUpdate = input("New name? <Enter> to skip ")

# Update name.

if nameUpdate != "":

self.competition.diverList[idUpdate - 1]\

.setName(nameUpdate)

# Enter the score to be changed.

scoreUpdate = input("Update score? <Y>es of "

"<Enter> to skip ")

# update score.

if scoreUpdate.lower() == "y":

oriScore = \

self.competition.diverList[idUpdate - 1]\

.getScores()

for i in range(7):

inScore = input("Score " + str(i + 1) +

" <Enter> to skip ")

if oriScore[i] != inScore and inScore != "":

oriScore[i] = eval(inScore)

# If there is no update input name and score

if nameUpdate == "" and scoreUpdate == "":

print("Update aborted")

# If there is an update enter the name and score

if nameUpdate != "" or scoreUpdate.lower() == "y":

print("Information Updated")

# if not, print invalid number.

else:

print("Invalid number")

elif choice == 5:

# Enter sort by name or average.

sortDiver = input(str("Sort according to <N>ame "

"or <A>verage score? "))

# Call method.

sortData = self.competition.sortAllDivers(sortDiver)

# Start number by 1.

number = 1

# Print sort data diver.

for i in sortData:

print(str(number) + ". " + str(i))

# Number added with 1.

number += 1

elif choice == 6:

# Enter file name to save.

fileName = input("File name to save? ")

# Call the method save.

self.competition.saveToFile(fileName)

elif choice == 7:

# If there are divers in the competition.

if self.competition.noOfDivers() > 0:

# Ask to user.

saveCurrentData = input("Do you want to "

"save current data? (Y/N) ")

if saveCurrentData == "y" or saveCurrentData == "Y":

# Ask file name.

fileName = input("File name to save? ")

# Save file.

self.competition.saveToFile(fileName)

# Delete data.

self.competition.diverList.clear()

elif saveCurrentData == "n" or saveCurrentData == "N":

print("Data not saved...")

# Delete Data

self.competition.diverList.clear()

# Load file.

fileName = input("File name to load: ")

# Call method readFromFile.

self.competition.readFromFile(fileName)

# Running the program.

def main():

DivingCompetition()

main()

Output











































