# PROG8421 - Programming for Big Data

**Assignment 3**

**Group Members:**

|  |  |
| --- | --- |
| **Name: Sudhan Shrestha** | **Name: Jose Encarnacion** |
| **Student ID: 8889436** | **Student ID: 8982860** |

**Question 1**

Use a list for the Test Scores program In this exercise, you’ll modify a Test Scores program that gets the test scores that a user enters and then calculates and displays the average test score. You’ll enhance this program by storing the test scores in a list and then getting and displaying other statistics for the test scores, like this:

1. In PyCharm, open the test\_scores.py file that’s given on eConestoga
2. Review the code, and test the program.
3. Modify the get\_scores() function so the test scores are stored in a list named scores. This list should be returned by the function when all scores have been entered. The function should still make sure that the entries are valid, but the score\_total and count variables aren’t needed and shouldn’t be updated.
4. Modify the process\_scores() function so the scores list is its only argument. Then, this function should use a for statement to total the scores in the list. It should use the len() function to get the number of scores in the list. And it should get the average by dividing the total scores by the length.
5. Modify the main () function so the list that’s returned by the get\_scores() function is stored in a variable. Then, modify the call to the process\_scores() function so it passes just the scores list to it.
6. Test this program to make sure everything is working right.
7. Enhance this program by getting and displaying all of the other statistics shown above. For an odd number of scores, the median score is the score that has the same number of scores below it as above it. For an even number of scores, calculate the median by taking the average of the two middle numbers

**Code**

#! ./env/bin/ python

from typing import List

from os import system

import statistics

def display\_welcome():

print()

print("The Test Scores program")

print("Enter 'x' to exit")

print("")

# 3. Modify the get\_scores() function so the test scores are stored in a list named scores.

# This list should be returned by the function when all scores have been entered.

# The function should still make sure that the entries are valid,

# but the score\_total and count variables aren't needed and shouldn't be updated.

def get\_scores() -> List:

data = list()

while True:

score = input("Enter test score: ")

if score == "x":

return data

else:

try:

score = int(score)

if score >= 0 and score <= 100:

data.append(score)

else:

print("Test score must be from 0 through 100. " +

"Score discarded. Try again.")

except ValueError:

print("Invalid input. Please enter a number.")

# 4. Modify the process\_scores() function so the scores list is its only argument.

# Then, this function should use a for statement to total the scores in the list.

# It should use the len()function to get the number of scores in the list.

# And it should get the average by dividing the total scores by the length.

def process\_scores(list\_score: List) -> None:

elem\_count = len(list\_score)

if elem\_count > 0:

score\_total = sum(list\_score)

count = elem\_count

average = round((score\_total / count), 2)

minimum = min(list\_score)

maximum = max(list\_score)

stdev = round((statistics.stdev(list\_score)), 2)

# Calculate median

sorted\_scores = sorted(list\_score)

if elem\_count % 2 == 1:

median = sorted\_scores[elem\_count // 2]

else:

median = (sorted\_scores[elem\_count // 2 - 1] + sorted\_scores[elem\_count // 2]) / 2

# Calculate mode (if exists)

try:

mode = statistics.mode(list\_score)

except statistics.StatisticsError:

mode = "No unique mode"

# Print all statistics

print()

print("Score total: ", score\_total)

print("Number of Scores: ", count)

print("Average Score: ", average)

print("Minimum Score: ", minimum)

print("Maximum Score: ", maximum)

print("Median Score: ", median)

print("Mode: ", mode)

print("Standard Deviation:", stdev)

else:

print()

print("No input score for calculation")

# 5. Modify the main () function so the list that's returned by the get\_scores() function is

# stored in a variable. Then, modify the call to the process\_scores() function so it passes

# just the scores list to it.

def main():

display\_welcome()

list\_score = get\_scores() # return list

process\_scores(list\_score) # recive a list for process

print("")

print("Bye!")

# if started as the main module, call the main function

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Comment**

The display\_welcome() function displays the welcome message and instructions for the program. The get\_score() prompts the user to enter a test score and stores it in a list until they want to exit by entering ‘x’. Furthermore, it validates if the score entered is between 0 and 100. The function process\_scores() computes the total score, count of scores, average score, minimum score, maximum score, median score, mode (if it exists), and standard deviation for the user-entered scores and outputs them.

**Screenshot**

**A screenshot of a computer program

Description automatically generated**

**Error Handling**

**A screenshot of a computer program

Description automatically generated**

**Question 2**

Use a list to store the players. Update the program (discussed in class) so it allows you to store the players for the starting lineup. This should include the player’s name, position, at bats, and hits.

A screenshot of a sports program

Description automatically generated

Specifications

* Use a list of lists to store each player in the lineup.
* Use a tuple to store all valid positions (C, 1B, 2B, etc).
* Make sure that the user’s entry for position is valid, and entries for hits and at bats make sense.
* Make sure you test all menu options(screenshots)

**Code**

#! ./env/bin/ python

from typing import List

from os import system

# Question-2. Use a list to store the players

# Update the program (discussed in class) so it allows you to store the players for the

# starting lineup. This should include the player's name, position, at bats, and hits. In

# addition, the program should calculate the player's batting average from at bats and hits.

valid\_positions = ('C', '1B', '2B', '3B', 'SS', 'LF', 'CF', 'RF', 'P') # 2. Use a tuple to store all valid positions

lineup = [] # 1. Use a list to store the players

def display\_menu():

print()

print("============================================================")

print("Baseball Team Manager")

print("MENU OPTIONS")

print("1 - Display lineup")

print("2 - Add player")

print("3 - Remove player")

print("4 - Move player")

print("5 - Edit player position")

print("6 - Edit player stats")

print("7 - Exit program")

print("POSITIONS")

print(", ".join(valid\_positions))

print("============================================================")

print()

def display\_positions():

print("POSITIONS")

print(", ".join(valid\_positions))

def display\_lineup():

if not lineup:

print("No players in the lineup.")

else:

print("Current lineup:")

for i, player in enumerate(lineup, start=1):

name, position, at\_bats, hits = player

average = hits / at\_bats if at\_bats > 0 else 0

print(f"{i}. Name: {name}, Position: {position}, At bats: {at\_bats}, Hits: {hits}, Batting Average: {average:.3f}")

def add\_player():

name = input("Name: ")

while True:

position = input("Position: ")

if position in valid\_positions: # 3. Making sure the position entered is valid

break

else:

print("Invalid position. Try again.")

display\_positions()

while True:

try:

at\_bats = int(input("At bats: "))

if at\_bats >= 0: # 3. Validating the number of at bats

break

except ValueError:

pass

print("Invalid entry. At bats must be a non-negative integer.")

while True:

try:

hits = int(input("Hits: "))

if 0 <= hits <= at\_bats: # 3. Validating the number of hits

break

except ValueError:

pass

print("Invalid entry. Hits must be a non-negative integer and cannot exceed at bats.")

lineup.append([name, position, at\_bats, hits])

print(f"{name} was added.")

def remove\_player():

display\_lineup()

try:

index = int(input("Enter the player number to remove: ")) - 1

if 0 <= index < len(lineup):

removed\_player = lineup.pop(index)

print(f"{removed\_player[0]} was removed.")

else:

print("Invalid player number.")

except ValueError:

print("Invalid entry. Please enter a valid player number.")

def move\_player():

display\_lineup()

try:

old\_index = int(input("Enter the player number to move: ")) - 1

if 0 <= old\_index < len(lineup):

new\_index = int(input("Enter the new position in the lineup: ")) - 1

if 0 <= new\_index < len(lineup):

player = lineup.pop(old\_index)

lineup.insert(new\_index, player)

print(f"{player[0]} was moved to position {new\_index + 1}.")

else:

print("Invalid new position.")

else:

print("Invalid player number.")

except ValueError:

print("Invalid entry. Please enter valid numbers.")

def edit\_player\_position():

display\_lineup()

try:

index = int(input("Enter the player number to edit position: ")) - 1

if 0 <= index < len(lineup):

while True:

position = input("New position: ")

if position in valid\_positions:

lineup[index][1] = position

print(f"{lineup[index][0]}'s position was updated to {position}.")

break

else:

print("Invalid position. Try again.")

display\_positions()

else:

print("Invalid player number.")

except ValueError:

print("Invalid entry. Please enter a valid player number.")

def edit\_player\_stats():

display\_lineup()

try:

index = int(input("Enter the player number to edit stats: ")) - 1

if 0 <= index < len(lineup):

while True:

try:

at\_bats = int(input("New at bats: "))

if at\_bats >= 0:

break

except ValueError:

pass

print("Invalid entry. At bats must be a non-negative integer.")

while True:

try:

hits = int(input("New hits: "))

if 0 <= hits <= at\_bats:

break

except ValueError:

pass

print("Invalid entry. Hits must be a non-negative integer and cannot exceed at bats.")

lineup[index][2] = at\_bats

lineup[index][3] = hits

print(f"{lineup[index][0]}'s stats were updated.")

else:

print("Invalid player number.")

except ValueError:

print("Invalid entry. Please enter a valid player number.")

def main():

while True:

display\_menu()

try:

option = int(input("Menu option: "))

if option == 1:

display\_lineup()

elif option == 2:

add\_player()

elif option == 3:

remove\_player()

elif option == 4:

move\_player()

elif option == 5:

edit\_player\_position()

elif option == 6:

edit\_player\_stats()

elif option == 7:

print("Bye!")

break

else:

print("Invalid option. Try again.")

except ValueError:

print("Invalid entry. Please enter a number between 1 and 7.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Comment**

The display\_menu() displays the menu for the program, display\_position() displays all valid positions. The display\_lineup() shows the current player lineup; it also notifies the user if there are no players in the lineup, calculates the average batting, and displays it along with other player information. The function add\_player() allows the user to input relevant information to a new player; this function also validates that the position entered is correct, along with the player bats and hits and appends the information in a list. Function remove\_player() displays the player's current line and prompts the user to select a player to remove by number. The move\_player() allows the users to rearrange the player line by moving a player to a new position, edit\_player\_position() allows the users to select a player and edit their position, while edit\_player\_status() allows editing the player's status, such as bats and hits.

**Screenshot**

**1 - Display lineup**

**A screenshot of a computer

Description automatically generated**

**2- Adding player**

**A screenshot of a computer program

Description automatically generated**

**3 – Remove Player**

**A screenshot of a computer program

Description automatically generated**

**4 – Move Player**

**A screenshot of a computer program

Description automatically generated**

**5 – Edit Player Position**

**A screenshot of a computer program

Description automatically generated**

**6 – Edit player status**

**A screenshot of a computer program

Description automatically generated**

**7 - Exit program**

**A screenshot of a computer program

Description automatically generated**

**Error handling**

**A screenshot of a computer program

Description automatically generated**

**Validation throughout the options**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**Question 3**

Create a program that determines and displays the number of unique characters in a string entered by the user. For example, Hello, World! has 10 unique characters while zzz has only one unique character.

Specifications

* Use a dictionary or set to solve this problem.
* Define a function

**Code**

#! ./env/bin/ python

from typing import List

from os import system

# Question-3. Create a program that determines and displays the number of unique characters in a

# string entered by the user. For example, Hello, World! has 10 unique characters while zzz has only one

# unique character.

# Specifications

# • Use a dictionary or set to solve this problem.

# • Define a function

def count\_unique\_characters(input\_string:str)->int:

'''

Function count\_unique\_characters for count unique characters

Parameters: input\_string: str

Returns: int

'''

#set to store unique characters

uni\_characters = set(input\_string)

# Return the number of unique characters

return len(uni\_characters)

def main():

# input

print()

print("Unique Character Counter")

print()

str\_input = input("Enter a string: ").lower() # Convert input to lowercase to avoid case sensitivity

# Call the function and display the result

count = count\_unique\_characters(str\_input)

print(f"The number of unique characters in the string is: {count}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Comment**

The variable str\_input stores the user input and converts it to lowercase to avoid case sensitivity while counting unique characters. The count\_unique\_characters() function counts and returns the unique characters in a given string. The unique characters in the given string is then counted and displayed.

**Screenshot**

**A black background with white text

Description automatically generated**

**A black screen with white text

Description automatically generated**

A black screen with white text

Description automatically generated

**Question 4**

This exercise examines the process of identifying the maximum value in a collection of integers. Each of the integers will be randomly selected from the numbers between 1 and 100.

The collection of integers may contain duplicate values, and some of the integers between 1 and 100 may not be present.

**Specifications**

Use randrange and import the relative python library to generate the random numbers

**Code**

#! ./env/bin/ python

from typing import List

from os import system

import random

# Question-4. This exercise examines the process of identifying the maximum value in a collection of

# integers. Each of the integers will be randomly selected from the numbers between 1 and 100.

# The collection of integers may contain duplicate values, and some of the integers between 1 and 100

# may not be present.

# Specifications

# Use randrange and import the relative python library to generate the random numbers

def random\_collection(size: int)-> list:

'''

Function random\_collection for generate random collection of numbers

Parameters: size: int

Returns: list

'''

random\_integers = [random.randrange(1, 101) for \_ in range(size)]

return random\_integers

def main():

print("")

# input

size = int(input("Enter the size of collection: "))

collection = random\_collection(size)

# Identify the maximum value in the collection

max\_value = max(collection)

# Print the collection and the maximum value

print("Collection of random integers:", collection)

print("Maximum value in the collection:", max\_value)

if \_\_name\_\_ == "\_\_main\_\_":

main()

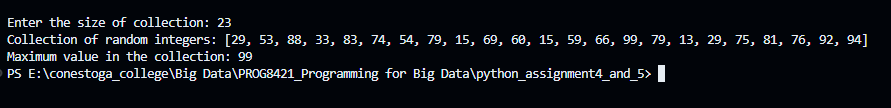
**Comment**

The random\_collection() function generates a random collection of integers based on the user-defined size. The collection of randomly generated integers is displayed, and the maximum value of the integer is displayed by utilizing the max() function.

**Screenshot**

**A black screen with white text

Description automatically generated**



**Question 5**

In the game of Scrabble™, each letter has points associated with it. The total score of a word is the sum of the scores of its letters. More common letters are worth fewer points while less common letters are worth more points.

The points associated with each letter are shown below:

A table with black and white text

Description automatically generated

Write a program that computes and displays the Scrabble™ score for a word.

**Specifications**

* Create a dictionary that maps from letters to point values. Then use the dictionary to compute the score.
* Use a function

**Code**

# Question 5

def scrabble\_score(word):

'''

Calculate the word score based on the scrabble rules

Parameters:

word (str): The word to be scored

Returns:

score (int): The word score

'''

# Points distribution for each letter

point\_sheet = {

1 : ['A', 'E', 'I', 'L', 'N', 'O', 'R', 'S', 'T', 'U'],

2 : ['D', 'G'],

3 : ['B', 'C', 'M', 'P'],

4 : ['F', 'H', 'V', 'W', 'Y'],

5 : ['K'],

8 : ['J', 'X'],

10 : ['Q', 'Z']

}

score = 0

for letter in word:

for key, value in point\_sheet.items():

if letter.upper() in value:

score += key

return score

def display\_title():

'''

Display the title of the game

Parameters: None

Returns: None

'''

print("")

print("Scrabble Score")

print("")

def play\_again():

'''

Prompt user to play again

Parameters: None

Returns: None

'''

again = input("Play again? (y/n) ")

if again.lower() == 'y':

game\_process()

elif again.lower() == 'n':

print("Bye!")

exit()

else:

print("Invalid input. Please enter 'y' or 'n'.")

print("")

play\_again()

def game\_process():

'''

User input and output for the game

Parameters: None

Returns: None

'''

while True:

word = input("Enter a word: ")

if word.isalpha() == False:

print("Invalid input. Please enter a word.")

else:

score = scrabble\_score(word)

print(word, "is worth", score, "points.")

print("")

play\_again()

def main():

display\_title()

game\_process()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Comment**

The dictionary point\_sheet stores the points for each word in the game. The function scrabble\_score() calculates the score of a given word based on the points distributed in the poin\_sheet dictionary and returns the score. The display\_title() displays the program title. The play\_again() function checks the validation for ‘y’ or ‘n’ entered when prompted if they want to continue the game. The function game\_process() includes the process of prompting the user to enter a word, validating whether alphabetic words are entered or not, and displaying the score to the user.

**Screenshot**

**A screenshot of a computer screen

Description automatically generated**

**Validation**

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer program

Description automatically generated