CS4051 Fundamentals of Computing Coursework

Muhammod Belal Uddin

Student ID: 23032697

Year 2024

Semester: 2nd semester

Module leader: Svetla Hubenova

Contents

[Introduction 3](#_Toc166272714)

[Literature Review 3](#_Toc166272715)

[Methodology 4](#_Toc166272716)

[Implementation: 5](#_Toc166272717)

[Testing 10](#_Toc166272718)

[Conclusion: 10](#_Toc166272719)

[References 10](#_Toc166272720)

[Appendix 10](#_Toc166272721)

## Introduction

This project aims to teach us how to construct a Python code that calculates student marks. In the first portion, we learn how to enter marks, compute basic statistics such as mean, median, and mode, and manage user interactions using a menu system. Part two adds further complexity by demanding numerical input, ensuring a minimum number of entries before computations, and incorporating skewness calculation. We study how to handle different inputs and read data from files in the last part. We are continuously asked to annotate the code to show that we understand it better.

As a whole, this project consisted of many elements that enabled us to partake in learning and gaining a better understanding of the python programme and the language that is being used. By implementing our ability into basic techniques and elements, we get more accustomed to the programming and are able to apply all the skills and abilities we have learnt to show that we can progress into the topic we are learning.

Every section and part had its own new inclusion of different tasks and requirements for us to challenge and overcome. By adding new elements overtime, we did not only learn just that, but how to also see the difference in the quality of our ability. Being tasked with adding more improvements, we gained more insight and further knowledge of how much more practical the python language can be. Thus, making us more experienced and better in programming this specific language.

## Literature Review

***Mode:***

The mode is the most common number in your data collection. To find the mode, count how often each number appears; the mode is the number that appears the most frequently. The mode of a list of integers may be found using the mode() function in the Python statistics package. For example, the code that follows finds the mode of a given integer list.

(LinkedIN, n.d.)

***Median:***

The term "median" refers to the middle figure. Note: In the event when the number of data values is odd, it delivers the exact middle value. If there are an even number of data items, the average of the two middle values is returned. It delivers the exact middle value if the data values contain odd numbers. The average of the two middle values is returned if the number of data items is even.

(w3schools, n.d.)

***Mean:***

Within the field of math, a mean represents the average of a set of numbers that is derived from the sum of the numbers and their division by the total number of numbers. The mean() method in Python, which calculates the average of a given data collection, may be used to illustrate this. A code made out of a set of numbers would sum up all of its values and divide that total by the number of values. (W3Schools, n.d.)

***Skewness:***

A statistical indication of an unequal distribution is skewness. The normal distribution is the most often used form of probability distribution and data. It is an approach to determining or measuring a distribution's shape. Rather of using the frequency distribution, a critical statistical technique is employed to evaluate the uneven performance. (turing, n.d.)

## Methodology

The project focuses on developing a Python program to manage student marks, including functionalities such as data input, basic statistical calculations (mean, median, mode), menu-based user interactions, numerical input validation, minimum entry requirements, and skewness calculation.

We adopted an Agile methodology for its iterative nature, allowing us to incrementally develop and refine the program based on feedback and evolving requirements. Agile facilitates collaboration, adaptability, and responsiveness to changes, which are crucial for a learning project like this.

Our application follows a hierarchical structure, organized into modules corresponding to different functionalities (e.g., data input, statistical calculations, user interface). This hierarchical approach enhances modularity, readability, and maintainability, essential aspects for learning and understanding the codebase.

Requirements:

Applying functionalities to input student marks and perform basic statistical calculations.

Developing a menu system for user interaction and navigation through different functionalities.

Adding complexity by incorporating numerical input validation, minimum entry requirements, and skewness calculation.

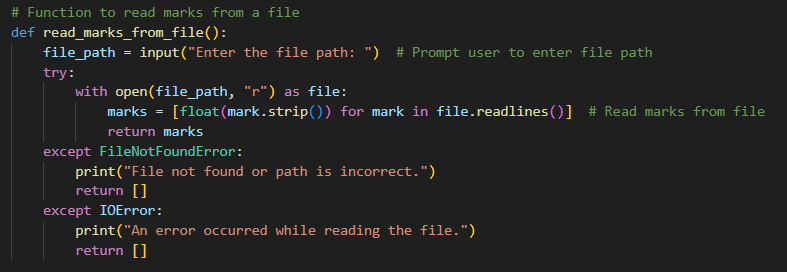
Demonstrating the ability to handle various input scenarios and read data from files, enhancing the program's versatility and practicality for real-world applications.

## Implementation:

A screen shot of a computer code

Description automatically generated

The student\_marks() function, defined by this code, allows users to enter student marks. First, an empty list is initialised to hold the marks. After then, it goes into an endless loop that keeps asking users to write "done" or input marks separated by commas to end it. Should 'done' be input, the function ends and the loop breaks. If you supply marks, the input string is divided into two parts separated by commas, and the leading and following whitespace is removed from each component. After that, an effort is made to transform each component into a floating-point number and add it to the list of marks. The user is notified if incorrect input causes the conversion to fail. The function provides a list with all of the valid marks that were entered when the user has finished entering marks.



This code defines the read\_marks\_from\_file() function, which reads marks from a file that the user chooses. Initially, it requests that the user specify the file path. Within a try-except block, it attempts to open the file using the specified file path in read mode ("r"). Upon successful file opening, the programme scans the lines, eliminates any leading or trailing whitespace, converts the format of each line to a floating-point integer, and compiles these signals into a list. This marks list is then returned to you. If the path is incorrect or the file cannot be found, an empty list is generated and a FileNotFoundError is raised.

A computer screen shot of text

Description automatically generated

The user can input numerical marks one at a time until they wish to stop using the enter\_new\_numbers() method, which is defined by this code. First, an empty list called marks is created in order to store the entered marks. It enters an infinite loop that utilises while True until the user enters "done" to signal that the input is complete. Somewhere along the loop, the user is prompted to enter a mark. If the input is tagged as "done," the loop breaks, ending the input process. If not, it attempts to convert the input into a floating-point number and adds it to the marks list. If a ValueError stops the input from being switched on, the user is requested to provide a correct number.After the user finishes inputting marks, the function prints the total number of marks entered and returns the list of marks.

A screen shot of a computer program

Description automatically generated

The enter\_more\_numbers(marks) function is defined in the provided code snippet. This function accepts as input a list named marks that most likely already has some number marks in it. The function asks the user to enter further marks, divided with commas, or to write "done" to signal the end of input, all within an endless loop that is started by while True. The input process is ended by the loop if the user inputs "done." If not, the function divides the input string from the user into many parts using commas and loops over each component. Every component has its leading and following whitespace removed. The code then tries to use float() to convert each component into a floating-point number and appends the outcome to the marks list. But if a ValueError prevents the conversion from succeeding, the code alerts the invalid input.

A black screen with white text

Description automatically generated

only uses the exit() method to quickly end the programme after printing a message to the user letting them know that the application is closing. This procedure, when called, guarantees a smooth and quick programme departure without requiring extra code execution. This feature comes in useful and gives the user a clear indicator that the application is shutting down when they want it to do so silently or in a certain circumstance.

A screenshot of a computer program

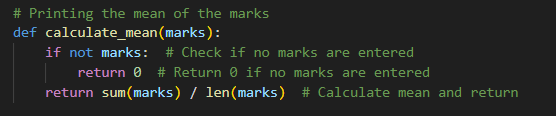
Description automatically generated

It determines a list of marks' mode or modes. It first determines whether the list of marks is empty. If so, the method returns and informs the user that no marks were input. The counts of each mark are then initialised into an empty dictionary called mark\_counts. The count for each mark in the dictionary is increased as it iterates through the list of marks, or it is initialised to 1 if the mark is seen for the first time. It finds the maximum count(s) from the dictionary to determine the mode(s) after counting the occurrences of each mark. It prints out the mode if there is just one. If there are more than one mode, they are printed out with commas between them.

A computer screen shot of a program code

Description automatically generated

This code, determines how skewed a list of marks is. The first thing it does is see if the list of marks is empty. The method returns 0 if it is, meaning that no marks have been input. If not, the skewness computation is carried out. First, by adding up all of the marks and dividing by the entire number of marks, it determines the mean of the marks. Next, taking into account if the number of marks is even or odd, it determines the median of the marks. The next step is to compute the squared deviations of each mark from the mean, determine the variance, and then take the square root to get the standard deviation of the marks. Lastly, the skewness is calculated using the formula (mean - median) \*\* 3 / std\_dev. By measuring the asymmetry of the mark distribution around its mean, skewness sheds light on the distribution pattern of the data.

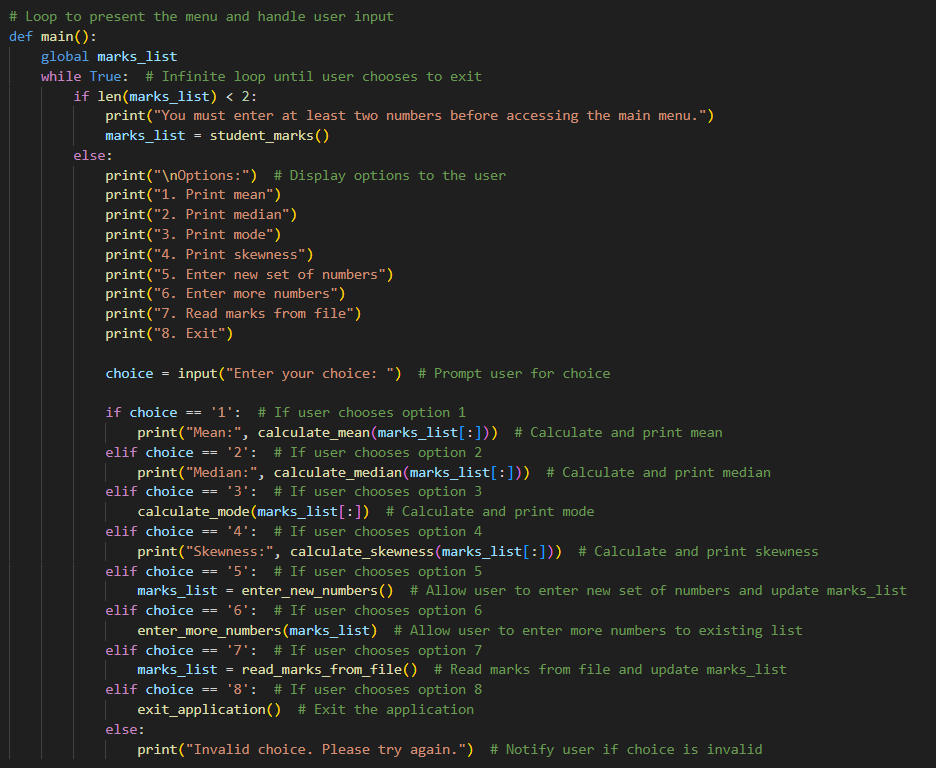


It determines the list of marks' mean, or average. It initially verifies that there are no marks in the list. The method returns 0 if it is, meaning that no marks have been input, as there are no values to take the mean from. If not, the computation is carried out by using the sum() function to add up all of the marks in the list, and then the len() function to divide the total by the number of marks. Ultimately, the computed mean value is returned. This function offers a simple method for calculating the average mark, a popular measure for central tendency in data analysis, using a provided list of marks.

A computer screen shot of a black background

Description automatically generated

Finds the median of a list of marks. The list of marks is first sorted in ascending order using the sorted() method. It then calculates the number of marks in the sorted list using the len() function. If the number of marks is even, the median is calculated by averaging the two middle values. It calculates the average and the index of the middle value. If the number of marks is odd, the middle value is used to compute the median. In the end, the calculated median value is given back.



The main() method defined in this code serves as the program's primary menu for handling student marks. It then goes into an endless loop where the menu items are shown nonstop until the user choose to end it. It first determines if the list of marks is empty or has fewer than two numbers in it. If so, it asks the user to enter at least two numbers. When a list has two or more numbers in it, the user is presented with a menu where they may choose to print the mean, median, mode, and skewness of the marks, input new numbers, add new numbers to the list already in it, read marks from a file, or close the programme.

The application does the appropriate action in accordance with the user's selection. For example, it reads marks from a file, computes and shows the mean, median, mode, or skewness of the marks, and lets the user enter new marks or add more marks to the list already there. It may also be used to terminate the programme. The application alerts the user and asks them to try again if they enter an incorrect option. All in all, this code simplifies the administration and analysis of student grades by arranging the program's functions into an intuitive menu interface.

## Testing

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

## Conclusion:

While the project has provided valuable learning experiences in Python programming and software development concepts, there are areas where improvement could be made. For instance, allocating more time for in-depth testing and code review could enhance the program's reliability and robustness. Additionally, considering ethical and social implications, such as data privacy and inclusivity, could enrich the learning experience and foster a deeper understanding of responsible software development practices.

# References

(n.d.). Retrieved from w3schools: https://www.w3schools.com/python/ref\_stat\_median.asp#:~:text=The%20median%20is%20the%20number,of%20the%20two%20middle%20values.

(n.d.). Retrieved from W3Schools: https://www.w3schools.com/python/ref\_stat\_mean.asp#:~:text=mean()%20method%20calculates%20the,how%20many%20values%20there%20are.

(n.d.). Retrieved from turing: https://www.turing.com/kb/calculating-skewness-and-kurtosis-in-python#:~:text=Calculating%20Skewness%20and%20Kurtosis%20in,distribution%20is%20a%20normal%20distribution.

*LinkedIN*. (n.d.). Retrieved from https://www.linkedin.com/pulse/mean-median-mode-now-python-ml-concepts-com#:~:text=The%20mode%20is%20the%20value,of%20a%20list%20of%20numbers.

## Appendix

# Function to collect student marks from user input

def student\_marks():

    marks = []  # Initialize an empty list to store the marks

    while True:  # Infinite loop until user enters 'done'

        mark\_input = input("Enter mark(s) separated by commas (or 'done' to finish): ")  # Prompt user for input

        if mark\_input.lower() == 'done':  # Confirm if user wants to proceed

            break  # Exit the loop if 'done' is entered

        # Split the input string by commas and iterate through each part

        for mark\_str in mark\_input.split(','):

            mark\_str = mark\_str.strip()  # Remove leading/trailing whitespace

            try:

                marks.append(float(mark\_str))  # Convert the input to float and append to the marks list

            except ValueError:

                print(f"Invalid input: '{mark\_str}'. Please enter a valid number.")  # Notify user if input is not a valid number

    return marks  # Return the list of marks

# Function to read marks from a file

def read\_marks\_from\_file():

    file\_path = input("Enter the file path: ")  # Prompt user to enter file path

    try:

        with open(file\_path, "r") as file:

            marks = [float(mark.strip()) for mark in file.readlines()]  # Read marks from file

            return marks

    except FileNotFoundError:

        print("File not found or path is incorrect.")

        return []

    except IOError:

        print("An error occurred while reading the file.")

        return []

# Function to allow the user to enter a new set of numbers

def enter\_new\_numbers():

    marks = []  # Initialize an empty list to store the marks

    while True:  # Infinite loop until user enters 'done'

        mark = input("Enter a mark (or type 'done' to finish): ")  # Prompt user for input

        if mark.lower() == 'done':  # Check if user wants to finish

            break  # Exit the loop if 'done' is entered

        else:

            try:

                marks.append(float(mark))  # Convert the input to float and append to the marks list

            except ValueError:

                print("Please enter a valid number.")  # Notify user if input is not a valid number

    print("You have entered", len(marks), "marks.")  # Inform user about the number of marks entered

    return marks  # Return the list of marks

# Function to allow the user to enter more numbers

def enter\_more\_numbers(marks):

    while True:

        mark\_input = input("Enter more mark(s) separated by commas (or 'done' to finish): ")

        if mark\_input.lower() == 'done':

            break

        for mark\_str in mark\_input.split(','):

            mark\_str = mark\_str.strip()

            try:

                marks.append(float(mark\_str))

            except ValueError:

                print(f"Invalid input: '{mark\_str}'. Please enter a valid number.")

# Function to allow the user to exit the application

def exit\_application():

    print("Exiting the application.")  # Notify user about exiting

    exit()  # Exit the program

# Printing the mode of the marks

def calculate\_mode(marks):

    if not marks:  # Check if no marks are entered

        print("No marks entered.")  # Notify user if no marks are entered

        return

    # Count occurrences of each mark

    mark\_counts = {}  # Initialize an empty dictionary to store mark counts

    for mark in marks:  # Iterate through the marks list

        if mark in mark\_counts:

            mark\_counts[mark] += 1  # Increment count if mark already exists in dictionary

        else:

            mark\_counts[mark] = 1  # Initialize count to 1 if mark is encountered for the first time

    # Find the mode(s)

    max\_count = max(mark\_counts.values())  # Find the maximum count

    modes = [mark for mark, count in mark\_counts.items() if count == max\_count]  # Get marks with maximum count

    if len(modes) == 1:  # Check if there is only one mode

        print("Mode of the numbers:", modes[0])  # Print the mode

    else:

        print("Modes of the numbers:", ", ".join(str(mode) for mode in modes))  # Print multiple modes if exist

# Function to calculate skewness

def calculate\_skewness(marks):

    if not marks:  # Check if no marks are entered

        return 0  # Return 0 if no marks are entered

    n = len(marks)

    mean = sum(marks) / n  # Calculate the mean

    # Step 2: Calculate the median

    sorted\_marks = sorted(marks)

    if n % 2 == 0:

        median = (sorted\_marks[n // 2 - 1] + sorted\_marks[n // 2]) / 2

    else:

        median = sorted\_marks[n // 2]

    # Step 3: Calculate the standard deviation

    squared\_deviations = [(x - mean) \*\* 2 for x in marks]

    variance = sum(squared\_deviations) / n

    std\_dev = variance \*\* 0.5

    # Calculate the skewness

    skewness = (mean - median) \*\* 3 / std\_dev

    return skewness

# Printing the mean of the marks

def calculate\_mean(marks):

    if not marks:  # Check if no marks are entered

        return 0  # Return 0 if no marks are entered

    return sum(marks) / len(marks)  # Calculate mean and return

# Printing the median of the marks

def calculate\_median(marks):

    sort\_marks = sorted(marks)  # Sort the marks in ascending order

    n = len(sort\_marks)  # Get the number of marks

    if n % 2 == 0:  # Check if the number of marks is even

        middle\_index = n // 2  # Calculate the middle index

        median = (sort\_marks[middle\_index - 1] + sort\_marks[middle\_index]) / 2  # Calculate the median

    else:

        median = sort\_marks[n // 2]  # If the number of marks is odd, median is the middle value

    return median  # Return the median

# Loop to present the menu and handle user input

def main():

    global marks\_list

    while True:  # Infinite loop until user chooses to exit

        if len(marks\_list) < 2:

            print("You must enter at least two numbers before accessing the main menu.")

            marks\_list = student\_marks()

        else:

            print("\nOptions:")  # Display options to the user

            print("1. Print mean")

            print("2. Print median")

            print("3. Print mode")

            print("4. Print skewness")

            print("5. Enter new set of numbers")

            print("6. Enter more numbers")

            print("7. Read marks from file")

            print("8. Exit")

            choice = input("Enter your choice: ")  # Prompt user for choice

            if choice == '1':  # If user chooses option 1

                print("Mean:", calculate\_mean(marks\_list[:]))  # Calculate and print mean

            elif choice == '2':  # If user chooses option 2

                print("Median:", calculate\_median(marks\_list[:]))  # Calculate and print median

            elif choice == '3':  # If user chooses option 3

                calculate\_mode(marks\_list[:])  # Calculate and print mode

            elif choice == '4':  # If user chooses option 4

                print("Skewness:", calculate\_skewness(marks\_list[:]))  # Calculate and print skewness

            elif choice == '5':  # If user chooses option 5

                marks\_list = enter\_new\_numbers()  # Allow user to enter new set of numbers and update marks\_list

            elif choice == '6':  # If user chooses option 6

                enter\_more\_numbers(marks\_list)  # Allow user to enter more numbers to existing list

            elif choice == '7':  # If user chooses option 7

                marks\_list = read\_marks\_from\_file()  # Read marks from file and update marks\_list

            elif choice == '8':  # If user chooses option 8

                exit\_application()  # Exit the application

            else:

                print("Invalid choice. Please try again.")  # Notify user if choice is invalid

# Running main function

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

    marks\_list = []  # Initialize global variable 'marks\_list'

    main()