# Calculator

* In this program, I created a function that randomly generates and returns a tuple of two or more positive integers.It randomly generates arithmetic questions (addition, subtraction, multiplication, or division) and evaluates the user’s answers. It also provides feedback for each answer and generates descriptive statistics for performance analysis.

import random

import matplotlib.pyplot as plt

students\_performance = {}

def generate\_question():

"""Generate random numbers and return a math question."""

num1 = random.randint(1, 10)

num2 = random.randint(1, 10)

return num1, num2

def ask\_question(question\_type):

"""Ask a math question based on the user's chosen type."""

num1, num2 = generate\_question()

if question\_type == 1:

answer = num1 + num2

print(f"What is {num1} + {num2}?")

elif question\_type == 2:

answer = num1 - num2

print(f"What is {num1} - {num2}?")

elif question\_type == 3:

answer = num1 \* num2

print(f"What is {num1} \* {num2}?")

elif question\_type == 4:

# Ensure no division by zero

while num2 == 0:

num2 = random.randint(1, 10)

answer = num1 / num2

print(f"What is {num1} / {num2}? (Round to 2 decimal places)")

else:

# Randomly select any type of question

question\_type = random.randint(1, 4)

return ask\_question(question\_type)

return answer

def get\_user\_input():

"""Prompt the user to select the type of arithmetic problem."""

print("Choose the type of arithmetic problem to study:")

print("1. Addition")

print("2. Subtraction")

print("3. Multiplication")

print("4. Division")

print("5. Random Mixture")

while True:

try:

choice = int(input("Enter your choice (1-5): "))

if 1 <= choice <= 5:

return choice

else:

print("Please enter a valid number (1-5).")

except ValueError:

print("Invalid input. Please enter a number (1-5).")

def provide\_feedback(correct):

"""Give feedback based on the user's answer."""

if correct:

responses = ['Very good!', 'Nice work!', 'Keep up the good work!']

print(random.choice(responses))

else:

responses = ['No. Please try again.', 'Wrong. Try once more.', 'No. Keep trying.']

print(random.choice(responses))

def display\_statistics(user\_name, performance):

"""Display statistics for a specific user."""

correct = performance["correct"]

incorrect = performance["incorrect"]

total = correct + incorrect

accuracy = (correct / total) \* 100 if total > 0 else 0

print(f"\nStatistics for {user\_name}:")

print(f"Total Questions: {total}")

print(f"Correct Answers: {correct}")

print(f"Incorrect Answers: {incorrect}")

print(f"Accuracy: {accuracy:.2f}%")

# Plotting the performance in a graph

labels = 'Correct', 'Incorrect'

sizes = [correct, incorrect]

colors = ['lightgreen', 'lightcoral']

explode = (0.1, 0) # explode the 'Correct' slice

plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.

plt.title(f"Performance of {user\_name}")

plt.show()

def display\_all\_statistics():

"""Display combined statistics for all users."""

all\_correct = sum(user["correct"] for user in students\_performance.values())

all\_incorrect = sum(user["incorrect"] for user in students\_performance.values())

total\_questions = all\_correct + all\_incorrect

print("\nCombined Statistics for All Students:")

print(f"Total Questions Answered: {total\_questions}")

print(f"Total Correct Answers: {all\_correct}")

print(f"Total Incorrect Answers: {all\_incorrect}")

# Plotting the combined performance

labels = 'Correct', 'Incorrect'

sizes = [all\_correct, all\_incorrect]

colors = ['lightgreen', 'lightcoral']

explode = (0.1, 0) # explode the 'Correct' slice

plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.

plt.title("Combined Performance of All Students")

plt.show()

def main():

user\_name = input("Enter your name: ")

question\_type = get\_user\_input()

score = 0

total\_questions = 5

# Initialize or update the performance data for the current user

if user\_name not in students\_performance:

students\_performance[user\_name] = {"correct": 0, "incorrect": 0}

for \_ in range(total\_questions):

correct\_answer = ask\_question(question\_type)

try:

user\_answer = float(input("Your answer: "))

if round(user\_answer, 2) == round(correct\_answer, 2):

provide\_feedback(True)

students\_performance[user\_name]["correct"] += 1

score += 1

else:

provide\_feedback(False)

students\_performance[user\_name]["incorrect"] += 1

except ValueError:

print("Invalid input. Please enter a numeric value.")

students\_performance[user\_name]["incorrect"] += 1

print(f"\nYou got {score} out of {total\_questions} questions correct!")

# Display statistics for the current user

display\_statistics(user\_name, students\_performance[user\_name])

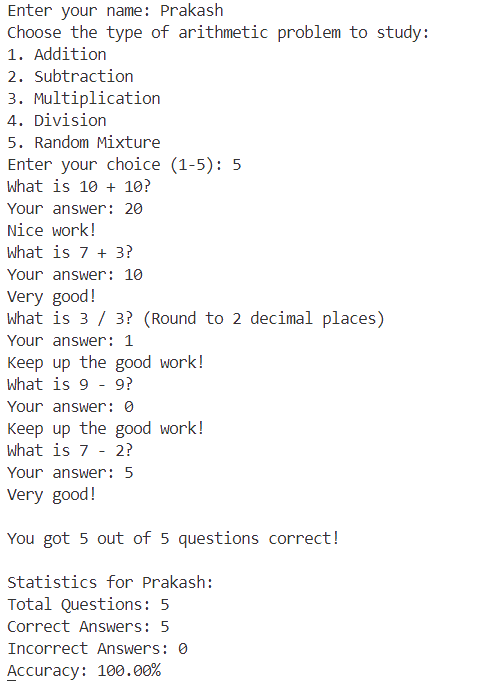
# Optionally display combined statistics for all students

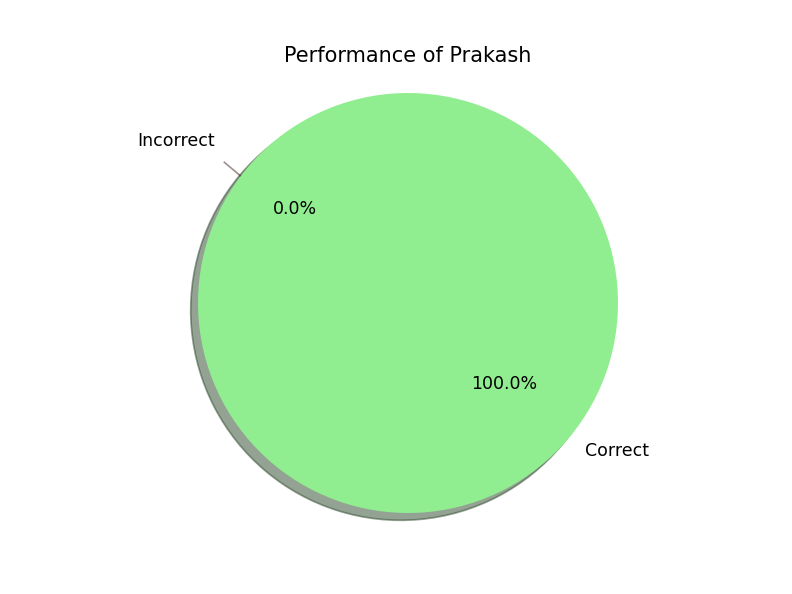
display\_all\_statistics()

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

main()

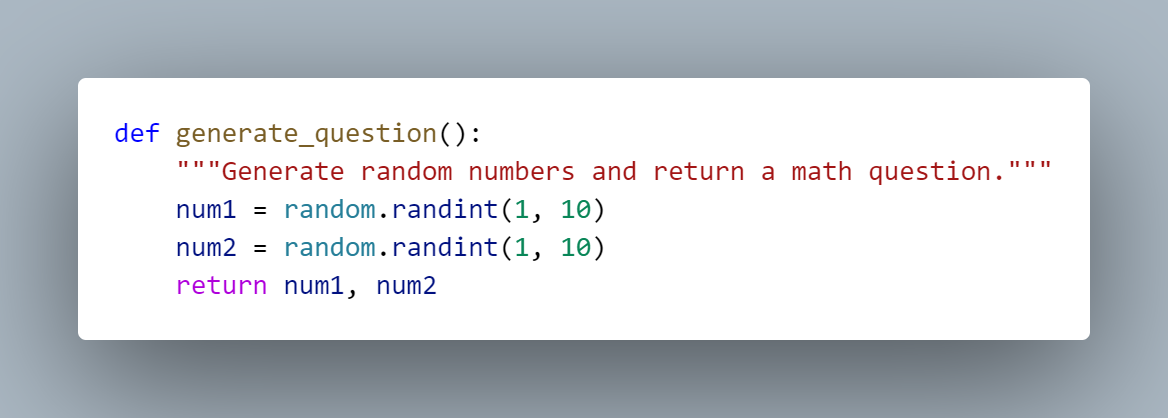
The output from the above program is given below:





* **Function to Generate Random Numbers**

The function generate\_question randomly generates two positive integers, which are then used to formulate a math question. This function ensures that each question posed to the student is unique and varied, later this function is called by the ask\_question function to generate the numbers used in a math question.



* **Function to Ask Math Questions**

The function ask\_question uses the randomly generated numbers to formulate a specific math question based on the chosen arithmetic type (addition, subtraction, multiplication, or division). If the user chooses a random mixture, it selects one of these types at random. Basically the purpose of this function is to prompt the user with a math question based on the selected type of arthematics.

def ask\_question(question\_type):

"""Ask a math question based on the user's chosen type."""

num1, num2 = generate\_question()

if question\_type == 1:

answer = num1 + num2

print(f"What is {num1} + {num2}?")

elif question\_type == 2:

answer = num1 - num2

print(f"What is {num1} - {num2}?")

elif question\_type == 3:

answer = num1 \* num2

print(f"What is {num1} \* {num2}?")

elif question\_type == 4:

# Ensure no division by zero

while num2 == 0:

num2 = random.randint(1, 10)

answer = num1 / num2

print(f"What is {num1} / {num2}? (Round to 2 decimal places)")

else:

# Randomly select any type of question

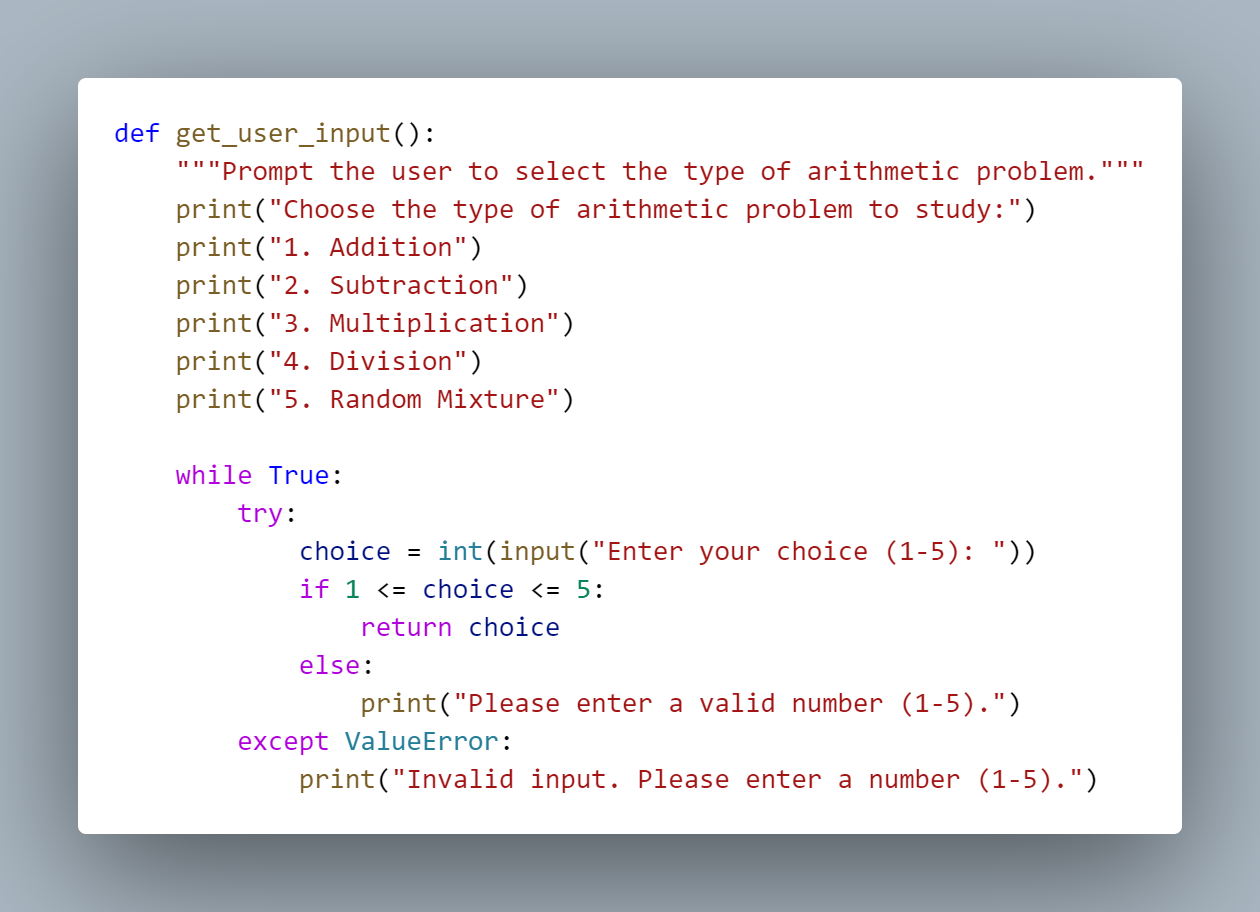
question\_type = random.randint(1, 4)

return ask\_question(question\_type)

return answer

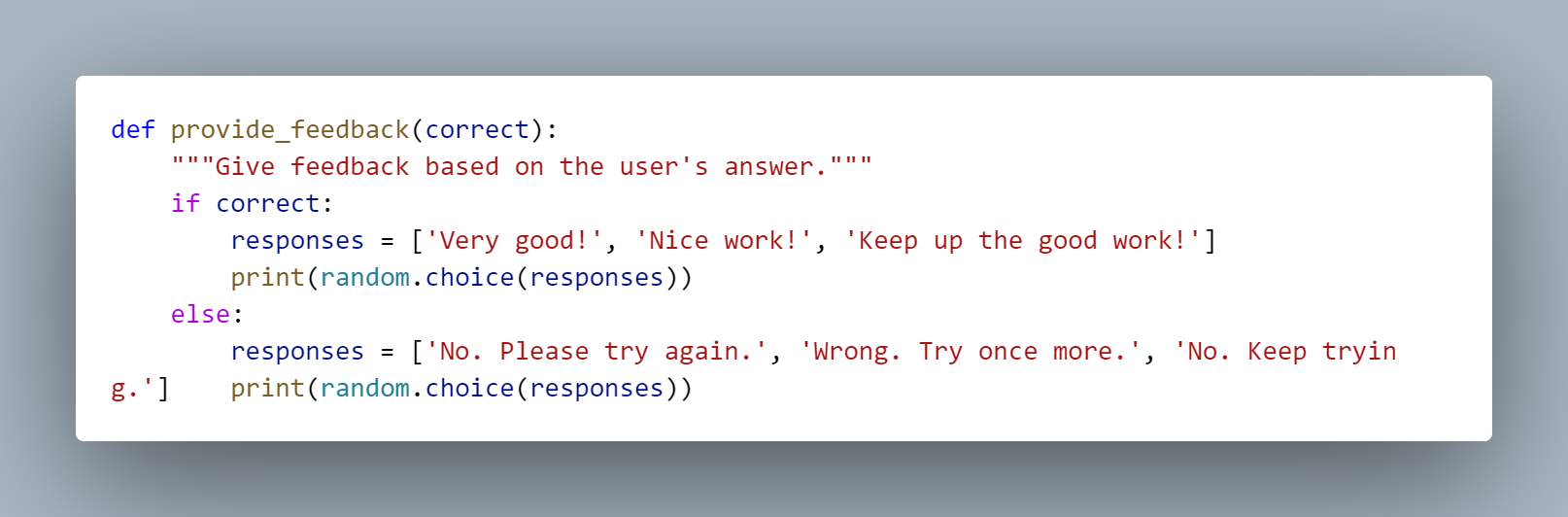
* **Function for User Input Selection**

The get\_user\_input function prompts the user to choose the type of arithmetic problem they wish to practice. It ensures that the user inputs a valid choice from 1 to 5.



* **Function to Provide Feedback**

The provide\_feedback function gives immediate feedback to the student based on whether their answer was correct or incorrect. It randomly selects from a list of possible feedback messages to maintain engagement.



* **Function to Display User Statistics**

The display\_statistics function presents the performance statistics for the specific user, including the number of correct and incorrect answers and the overall accuracy percentage. It also generates a pie chart to visualize these results.



* **Function to Display Combined Statistics for All Users**

The display\_all\_statistics function aggregates the performance data of all users and displays combined statistics, including a pie chart for visualization



# String

* The program `string\_1.py` defines a function that takes a string input from a student and returns a list of tuples, each containing a unique letter from the string along with its frequency. It ignores punctuation, case sensitivity, and spaces while processing. Additionally, the function checks if the input string contains every letter of the alphabet.

import string

from collections import Counter

from itertools import permutations

import matplotlib.pyplot as plt

# Function to summarize letters in a string

def summarize\_letters(input\_string):

filtered\_string = ''.join(filter(str.isalpha, input\_string.lower()))

frequency = Counter(filtered\_string)

letter\_summary = list(frequency.items())

return letter\_summary

# Function to check if the string has all the letters of the alphabet

def has\_all\_letters(input\_string):

filtered\_string = ''.join(filter(str.isalpha, input\_string.lower()))

alphabet = set(string.ascii\_lowercase)

return alphabet.issubset(set(filtered\_string))

# Function to present descriptive statistics and graphs

def present\_statistics(letter\_summary):

letters, frequencies = zip(\*letter\_summary)

print("Descriptive Statistics:")

print(f"Total Unique Letters: {len(letters)}")

print(f"Most Frequent Letter: {max(letter\_summary, key=lambda x: x[1])}")

print(f"Least Frequent Letter: {min(letter\_summary, key=lambda x: x[1])}")

plt.bar(letters, frequencies)

plt.xlabel('Letters')

plt.ylabel('Frequency')

plt.title('Letter Frequency in the String')

plt.show()

# Function to sort letters and remove duplicates

def sort\_letters(input\_string):

filtered\_string = ''.join(filter(str.isalpha, input\_string.lower()))

sorted\_unique\_letters = sorted(set(filtered\_string))

return ''.join(sorted\_unique\_letters)

# Function to generate anagrams

def generate\_anagrams(input\_string):

filtered\_string = ''.join(filter(str.isalpha, input\_string.lower()))

anagrams = set(''.join(p) for p in permutations(filtered\_string))

return list(anagrams)

# Input string from the user

input\_string = input("Enter a string: ")

# Summarize letters

letter\_summary = summarize\_letters(input\_string)

print("Letter Summary (Letter: Frequency):")

for letter, freq in letter\_summary:

print(f"{letter}: {freq}")

# Check if the string has all letters of the alphabet

if has\_all\_letters(input\_string):

print("The string contains all the letters of the alphabet.")

else:

print("The string does not contain all the letters of the alphabet.")

# Present descriptive statistics and graph

present\_statistics(letter\_summary)

# Sort letters and remove duplicates

sorted\_letters = sort\_letters(input\_string)

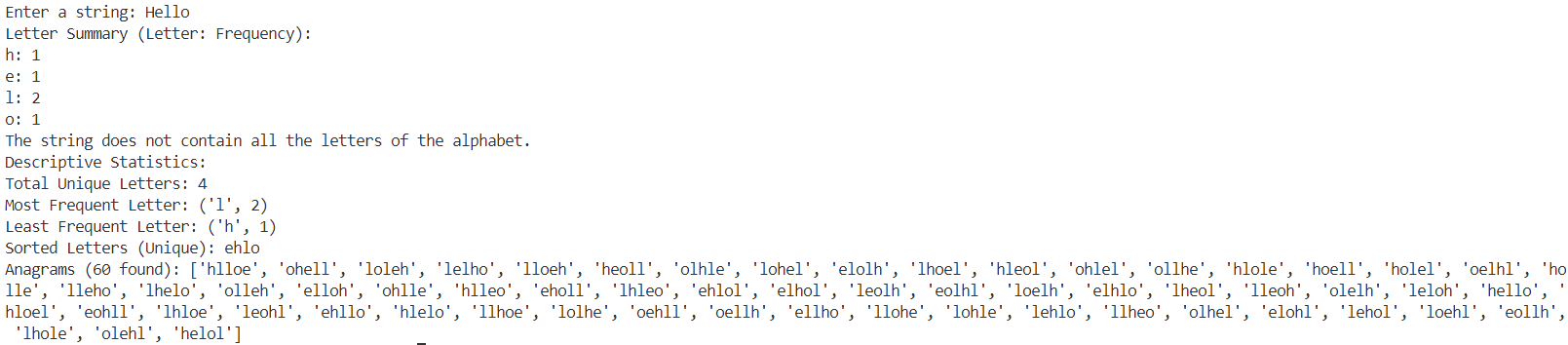
print("Sorted Letters (Unique):", sorted\_letters)

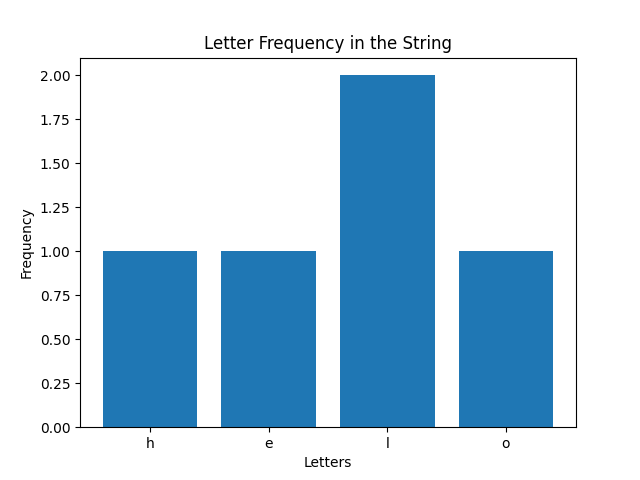
# Generate anagrams

anagrams = generate\_anagrams(input\_string)

print(f"Anagrams ({len(anagrams)} found):", anagrams)

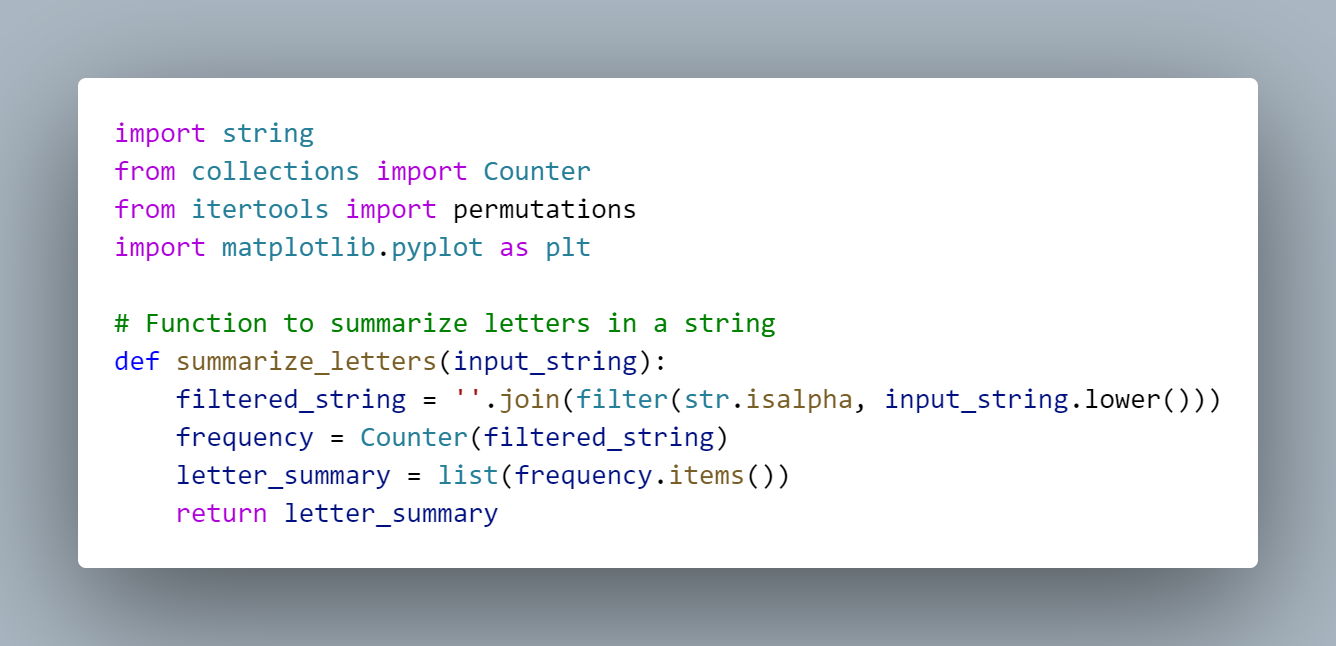
The output from the above program is given below:





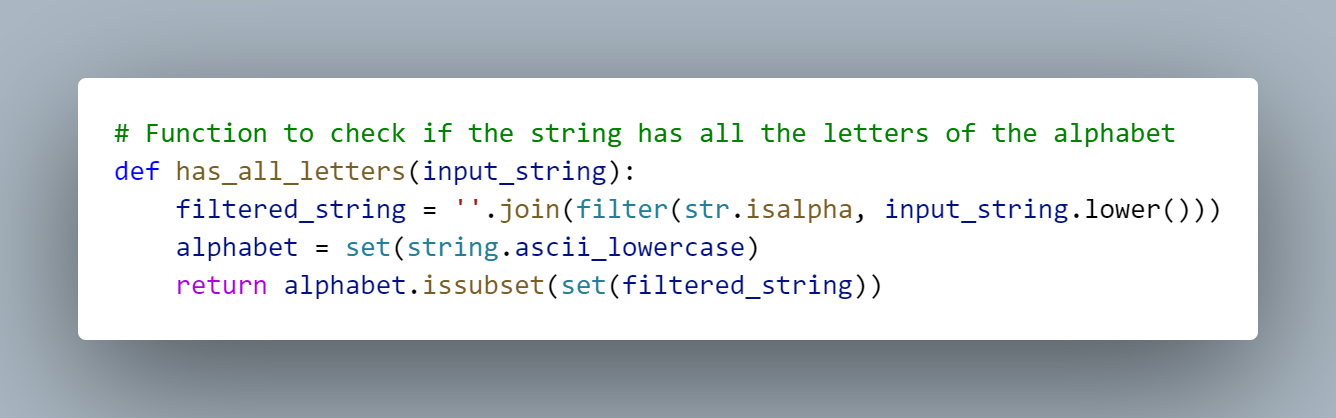
* **Function: summarize\_letters(input\_string) (Summarizing Letters)**

This function takes an input string and returns a list of tuples containing unique letters and their frequencies. It ignores case sensitivity (i.e., 'a' and 'A' are considered the same) and filters out spaces and punctuation.



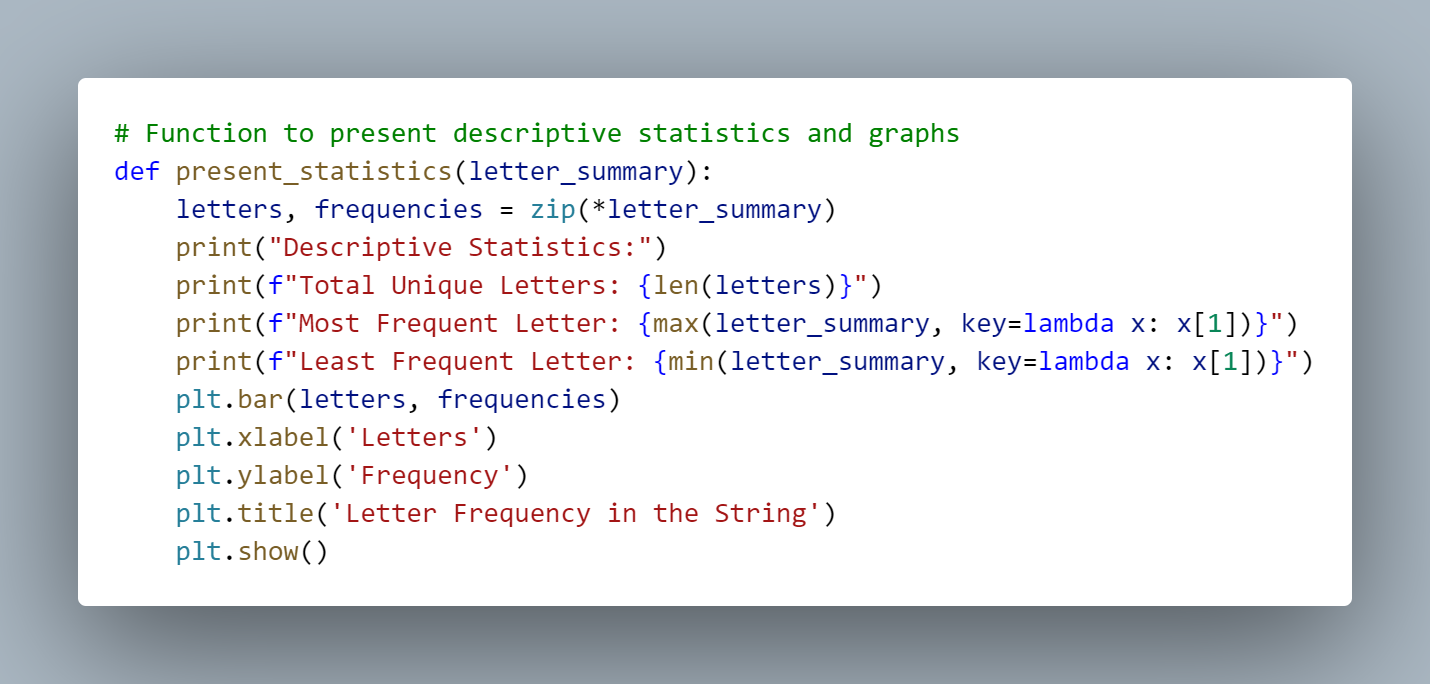
* **Function: has\_all\_letters(input\_string) (Checking for All Alphabet Letters)**

This function checks whether the input string contains all the letters of the English alphabet.



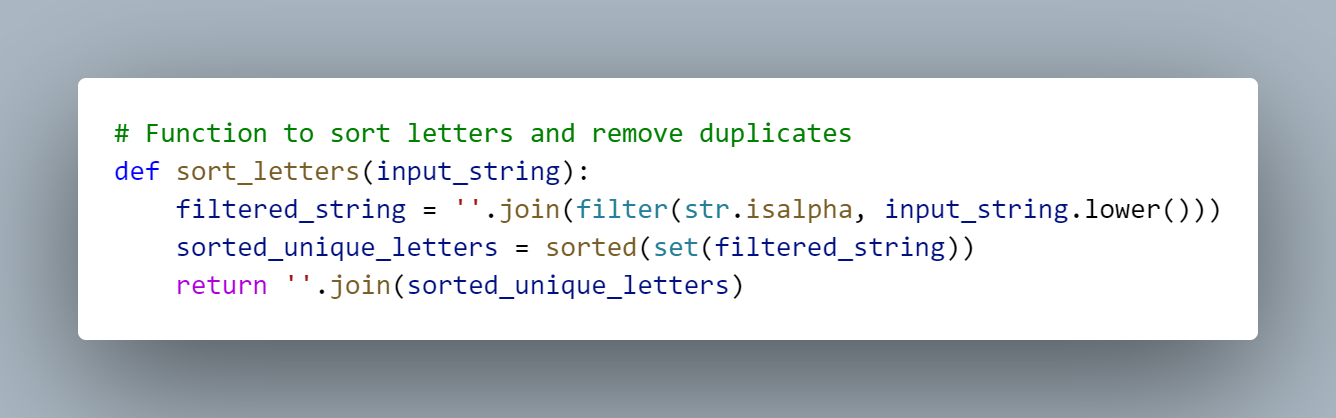
* **Function: present\_statistics(letter\_summary) (Presenting Descriptive Statistics)**

This function presents descriptive statistics about the letter frequencies and plots a bar graph using matplotlib, it unpacks letter\_summary to separate letters and their frequencies and displays descriptive statistics like total unique letters, the most frequent letter, and the least frequent letter.

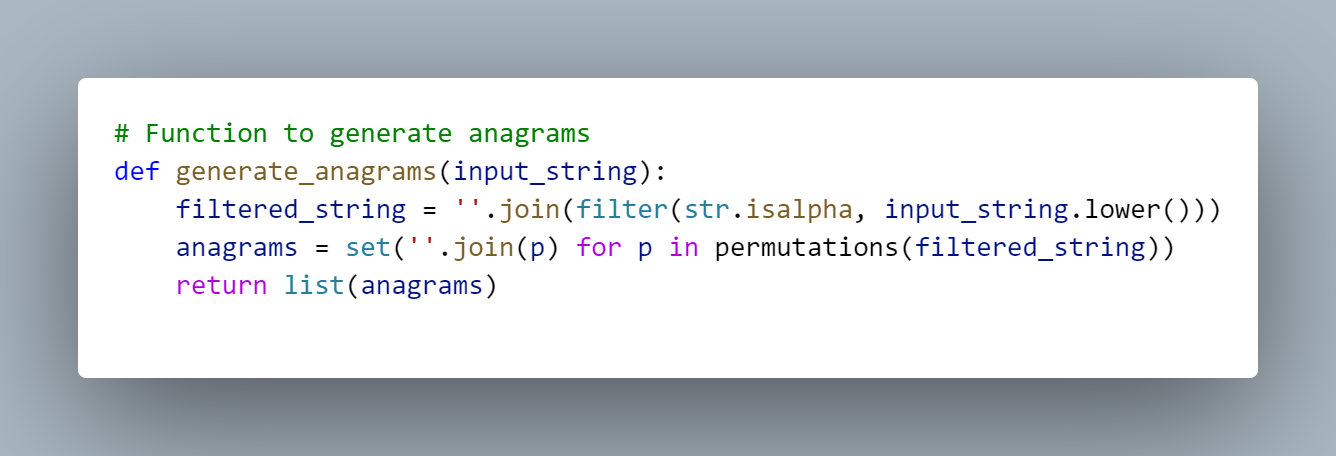
****

* **Function: sort\_letters(input\_string) (Sorting Letters and Removing Duplicates)**

This function begins by converting the input string to lowercase to ensure that letter comparisons are case-insensitive. It then filters out any non-alphabetic characters, keeping only the letters. To remove duplicate letters, it uses a Python `set`, which inherently contains only unique elements. After obtaining the unique letters, the function sorts them alphabetically using the `sorted` function. Finally, the sorted letters are joined back together into a single string, which is then returned as the output.



* **Function: generate\_anagrams(input\_string) (Generating Anagrams)**

The function first converts the input string to lowercase to ensure uniformity and filters out any non-alphabetic characters, retaining only the letters for further processing. It then utilizes `itertools.permutations` to generate all possible rearrangements (anagrams) of the filtered string. Since some rearrangements may be identical, the function employs a `set` to remove duplicate anagrams, ensuring that each anagram is unique. Finally, the unique anagrams are collected into a list, which is returned as the result.****

# 3. Survey

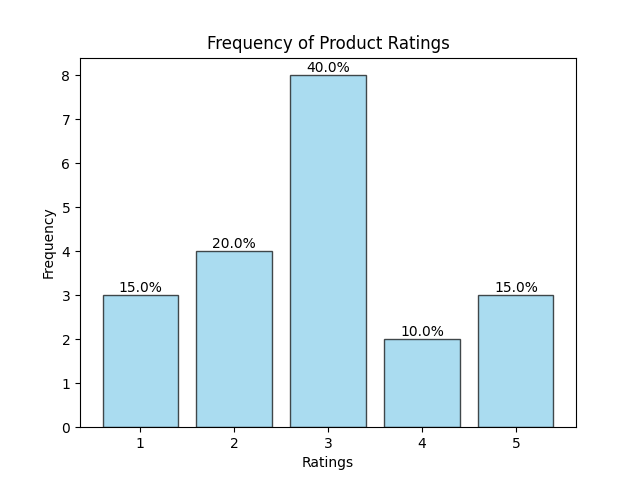
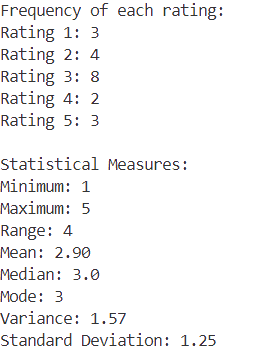
This program aims to analyze and visualize survey data collected from students who were asked to rate the quality of 20 different products in a store. The ratings range from 1 to 5, where:

* **1** represents "awful"
* **5** represents "excellent"

We will use Python to process this data and determine the frequency of each rating. In addition, we will employ built-in functions, the statistics module, and NumPy to calculate various statistical measures, including the minimum, maximum, range, mean, median, mode, variance, and standard deviation. Finally, the program will generate a bar chart to visually represent the frequency of each rating and their percentages of the total responses.

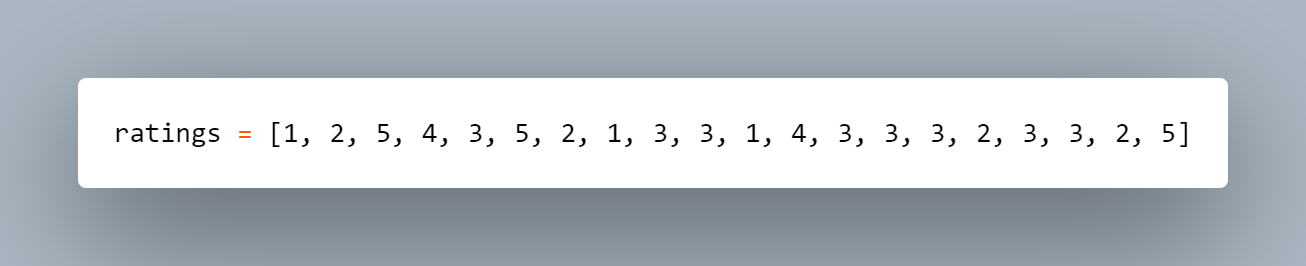


The output from the above program is given below:



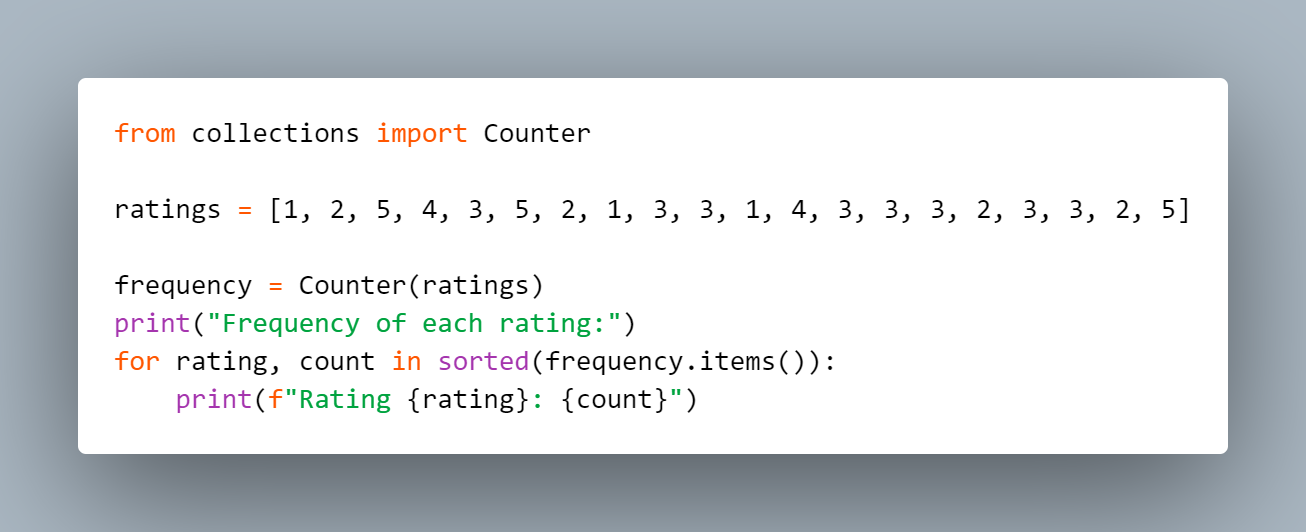
* **Data Collection and Storage**

The purpose of this section is to collect and store ratings provided by students for 20 different products, where each rating is an integer between 1 (awful) and 5 (excellent). The input consists of a list of these integer ratings, which will be used for further analysis. The collected ratings will allow us to perform statistical evaluations and generate visualizations that summarize the overall feedback on the products.

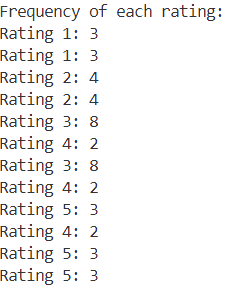


* **Frequency Calculation of Each Rating**

This determines the frequency of each rating (from 1 to 5) in the given list of student responses. The input is a list of ratings provided by students. The process involves using the `Counter` class from the `collections` module to count how often each unique rating appears in the list. The function `Counter(ratings)` creates a frequency dictionary, while `sorted(frequency.items())` sorts the dictionary entries by rating in ascending order. The output is a dictionary (`frequency`) where the keys represent the ratings, and the values indicate their respective counts.

****

Example Output :

****

* **Statistical Measures Calculation**

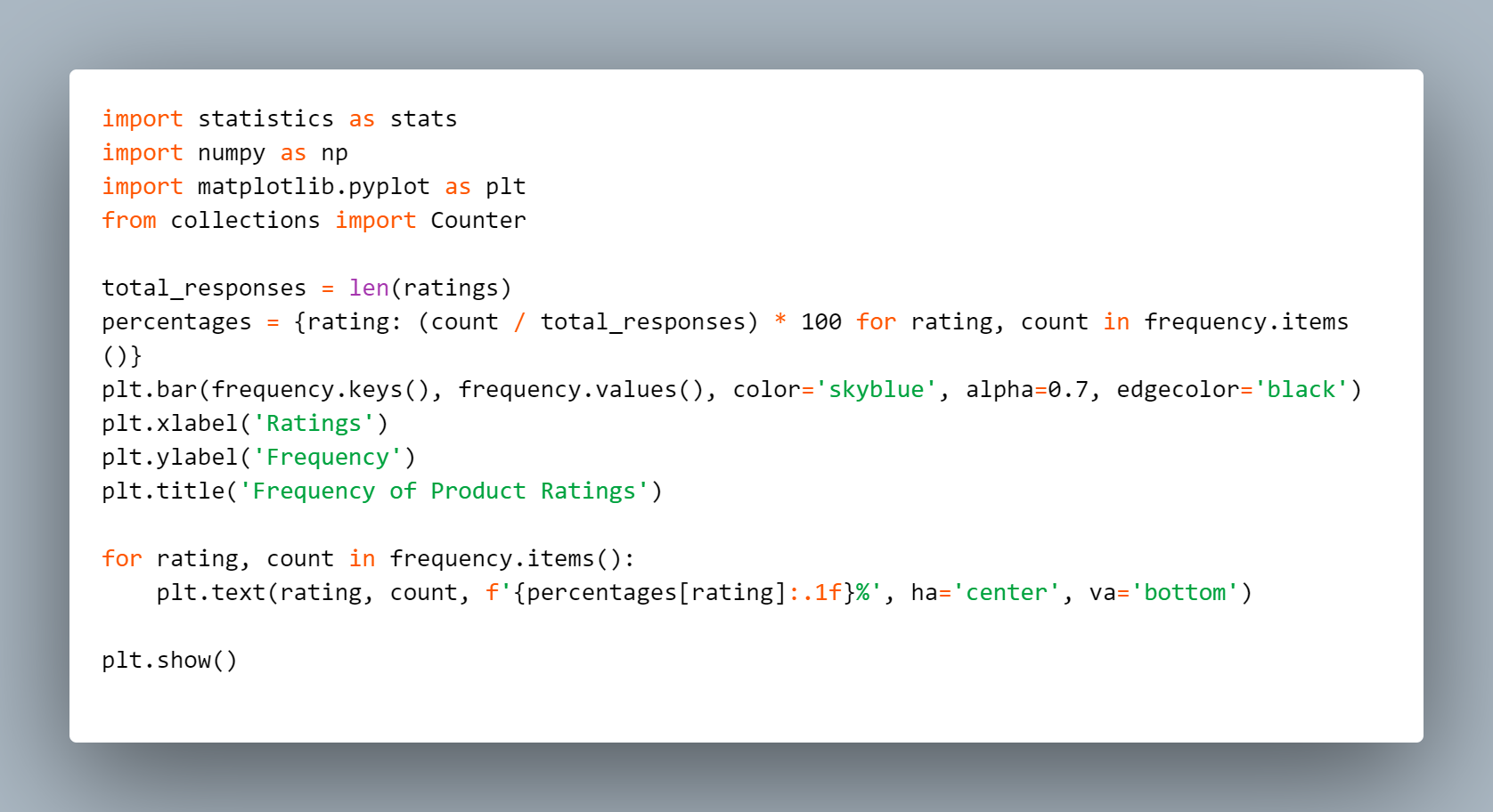
****

This part uses the statistics and NumPy modules to calculate various statistical measures:

1. min(ratings) and max(ratings): Find the smallest and largest ratings. Output: 1 (Minimum) and 5 (Maximum).
2. max(ratings) - min(ratings): Calculate the range of ratings. Output: 4 (Range).
3. np.mean(ratings): Compute the mean (average) of the ratings. Output: 3.05 (Mean).
4. np.median(ratings): Find the median, the middle value in the sorted list. Output: 3 (Median).
5. stats.mode(ratings): Identify the most frequently occurring rating. Output: 3 (Mode).
6. np.var(ratings, ddof=1): Calculate the variance, showing how spread out the ratings are. Output: 1.21 (Variance).
7. np.std(ratings, ddof=1): Compute the standard deviation, which measures the average variation from the mean. Output: 1.10 (Standard Deviation).

* **Displaying the Bar Chart of Response Frequencies and Percentages**

Here we visually display the frequency of each rating using a bar chart, along with the percentage of total responses. The input includes the frequency dictionary and the total number of responses. The process involves calculating the percentage for each rating and creating a bar chart where the x-axis represents ratings and the y-axis represents their frequencies. The chart is styled with colors, and percentages are displayed above each bar. Finally, plt.show() renders the chart, providing a clear visual summary of the rating distribution.

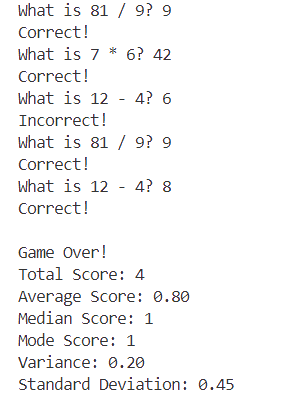


# 4. Game

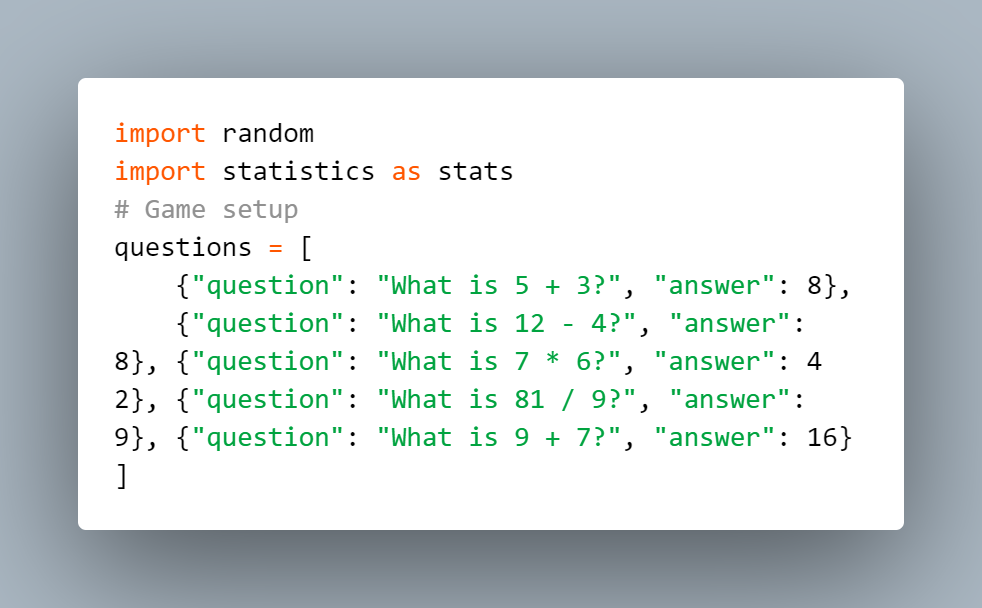
This game project motive is to develop an engaging and educational game that incorporates elements of words or numbers and integrates descriptive statistics. The final game will combine the most effective features from individual contributions to create a fun and stimulating arcade or single-player experience. By combining entertainment with cognitive challenges, the game aims to offer players both relaxation and an opportunity to enhance their brain skills through interactive play. This approach ensures a comprehensive and enjoyable gaming experience that is both mentally stimulating and entertaining.



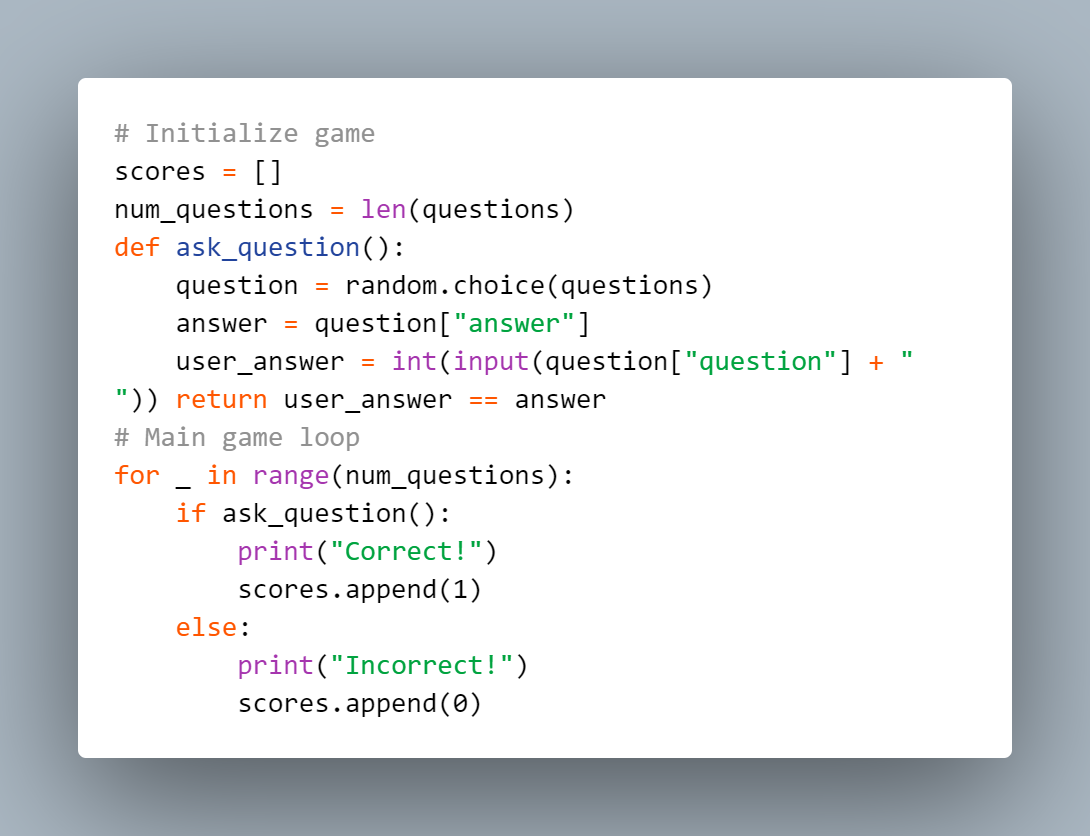
The output from the above program is given below:

****

* **Setup:** Define questions and answers. Initialize game state and statistics collection.



* **Gameplay Loop:** Ask questions, evaluate answers, and track scores.



* **Descriptive Statistics: :** Compute and display statistics based on the collected scores.

