Python Assignment-1

Name	Aditi Paitandy
Class	MCA Sec- A
Registration No.	PROV/MCA/7/24/020
Github Account	github.com/aditipaitandy/Python

Summary Report

This report summarizes the exercises completed in Assignment 1, which focused on the use of operators, strings, and lists in Python. The assignment consists of various exercises aimed at enhancing programming skills through practical implementation of concepts.

Questions:

Part 1: Operators

Exercise 1: Arithmetic Operators

Write a Python program to perform the following operations:
1. Add, subtract, multiply, and divide two numbers (input by the user).
2. Use the modulus operator to find the remainder of their division.
3. Use the exponentiation operator to raise the first number to the power of the second
number.
4. Perform floor division on the two numbers.

Expected Input:

```
Enter first number: 10
Enter second number: 3
```

Expected Output:

```
Addition: 13
Subtraction: 7
Multiplication: 30
Division: 3.33
Modulus: 1
Exponentiation: 1000
Floor Division: 3
```

Exercise 2: Comparison Operators

Write a Python program that asks for two numbers and checks:

1. If the first number is greater than the second.
2. If the first number is equal to the second.
3. If the first number is less than or equal to the second.

Print the results.

Exercise 3: Logical Operators

Write a Python program that:

1. Takes three boolean values (`True` or `False`) as input.
2. Uses `and`, `or`, and `not` operators to return the result of combining them.

Part 2: Strings

Exercise 4: String Manipulation

1. Take a string input from the user.
2. Display the following:

- The length of the string.
- The first and last character.
- The string in reverse order.
- The string in uppercase and lowercase.

Exercise 5: String Formatting

Write a program that asks for the user's name and age, and displays the message in this format:

```
Hello [Name], you are [Age] years old.
```

Exercise 6: Substring Search

Write a Python program that:

1. Asks for a sentence input from the user.
2. Asks for a word to search in the sentence.
3. Outputs whether the word exists in the sentence and, if it does, at which position (index)

Part 3: Lists

Exercise 7: List Operations

Write a Python program that:
1. Creates a list of 5 numbers (input from the user).
2. Displays the sum of all the numbers in the list.
3. Finds the largest and smallest number in the list.

Exercise 8: List Manipulation

1. Create a list of 5 of your favorite fruits.
2. Perform the following:
   - Add one more fruit to the list.
   - Remove the second fruit from the list.
   - Print the updated list.

Exercise 9: Sorting a List

Write a Python program that:
1. Asks the user to input a list of 5 numbers.
2. Sorts the list in ascending order and displays it.
3. Sorts the list in descending order and displays it.

Exercise 10: List Slicing

Given the list `numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`, perform the following:
1. Print the first 5 elements.
2. Print the last 5 elements.
3. Print the elements from index 2 to index 7.

Bonus Challenge

Exercise 11: Nested List

Write a Python program that:
1. Takes input of 3 students' names and their respective scores in 3 subjects.
2. Stores them in a nested list.
3. Prints each student's name and their average score

.

Solutions:

Part 1: Operators

Exercise 1: Arithmetic Operators

Program:

```
# Arithmetic Operations
# Get input from the user
n1= float(input("Enter first number: "))
n2= float(input("Enter second number: "))

# Perform calculations
addition = n1 + n2
subtraction = n1 - n2
multiplication = n1 * n2
division = n1 / n2
modulus = n1 % n2
exponentiation = n1 ** n2
floor_division = n1 // n2

# Display results
print(f"Addition: {addition}")
print(f"Subtraction: {subtraction}")
print(f"Multiplication: {multiplication}")
print(f"Division: {division:.2f}")
print(f"Modulus: {modulus}")
print(f"Exponentiation: {exponentiation}")
print(f"Floor Division: {floor_division}")
```

Output:

```
Enter first number:  10
Enter second number:  3
Addition: 13.0
Subtraction: 7.0
Multiplication: 30.0
Division: 3.33
Modulus: 1.0
Exponentiation: 1000.0
```

```
Floor Division: 3.0
```

Explanation of the Code:

This program takes two numbers as input from the user and performs a series of
basic arithmetic operations: addition, subtraction, multiplication, divisio n, modulus (remainder), exponentiation (raising the first number to the power of the second),
and floor division (division that rounds down to the nearest whole numb er). The
input numbers are converted to floating-point values to handle both inte gers and
decimals. After calculating the results for each operation, the program di splays them.Notably, the division result is formatted to two decimal plac es for clarity.

Exercise 2: Comparison Operators

Program:

```
# Comparison Operations
# Get input from the user
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))

# Perform comparisons
greater_than = num1 > num2
equal_to = num1 == num2
less_than_or_equal = num1 <= num2

# Display results
print(f"Is the first number greater than the second? {greater_than}")
print(f"Is the first number equal to the second? {equal_to}")
print(f"Is the first number less than or equal to the second?
{less_than_or_equal}")
```

Output:

```
Enter first number:  30
Enter second number:  20
Is the first number greater than the second? True
Is the first number equal to the second? False
Is the first number less than or equal to the second? False
```

Explanation of the Code:

This program performs comparison operations between two numbers entered by the user. It begins by prompting the user to input two numbers, which are converted to floating-point numbers for precise arithmetic handling. The program then uses comparison operators to evaluate three conditions: whether the first number is greater than the second (>), whether the two numbers are equal (==), and whether the first number is less than or equal to the second (<=). These results are stored in variables (greater_than, equal_to, and less_than_or_equal). Finally, the program prints the outcome of each comparison, displaying whether each condition is true or false. This allows the user to see how their inputted numbers compare to each other in terms of magnitude and equality.

Exercise 3: Logical Operators

Program:

```
# Logical Operations
# Get input from the user
a = input("Enter first boolean value (True/False): ") == 'True'
b = input("Enter second boolean value (True/False): ") == 'True'
c = input("Enter third boolean value (True/False): ") == 'True'

# Combine boolean values using logical operators
and_result = a and b and c
or_result = a or b or c
```

```
not_result1 = not a
not_result2 = not b
not_result3 = not c

# Display results
print(f"AND result: {and_result}")
print(f"OR result: {or_result}")
print(f"NOT first boolean: {not_result1}")
print(f"NOT second boolean: {not_result2}")
print(f"NOT third boolean: {not_result3}")
```

Output:

```
Enter first boolean value (True/False):  True
Enter second boolean value (True/False):  False
Enter third boolean value (True/False):  True
AND result: False
OR result: True
NOT first boolean: True
NOT second boolean: True
NOT third boolean: False
```

Explanation of the Code:

This program takes three boolean values (True/False) as input and applies logical operations on them. It performs an **AND** operation, where the result is True only if all three values are True; an **OR** operation, which is True if at least one value is True; and **NOT** operations, which invert each individual boolean value. The results of these logical operations are then printed, showing how the inputs combine using the and, or, and notoperators to produce the final outcomes.

Part 2: Strings

Exercise 4: String Manipulation

Program:

```
# String Manipulation
# Get string input from the user
user_string = input("Enter a string: ")

# Perform string manipulations
```

```
string_length = len(user_string)
first_character = user_string[0]
last_character = user_string[-1]
reversed_string = user_string[::-1]
uppercase_string = user_string.upper()
lowercase_string = user_string.lower()

# Display results
print(f"Length of string: {string_length}")
print(f"First character: {first_character}")
print(f"Last character: {last_character}")
print(f"Reversed string: {reversed_string}")
print(f"Uppercase string: {uppercase_string}")
print(f"Lowercase string: {lowercase_string}")
```

Output:

```
Enter a string:  Elephant
Length of string: 8
First character: E
Last character: t
Reversed string: tnahpelE
Uppercase string: ELEPHANT
Lowercase string: elephant
```

Explanation of the Code:

This program allows the user to input a string and performs several operations on it. First, it calculates the **length** of the string using len(), which counts the number of characters. It then extracts the **first** and **last characters** using indexing—[0] for the first and [-1] for the last. Next, it **reverses** the string by using slicing with [::-1], which starts from the end and steps backward. It converts the string to **uppercase** using the upper() method and to **lowercase** using lower(). Finally, the results are printed, allowing the user to see the string's length, the first and last characters, the reversed string, and how it looks in uppercase and lowercase.

Exercise 5: String Formatting

Program:

```
# String Formatting
# Get user input for name and age
name = input("Enter your name: ")
age = input("Enter your age: ")

# Display message
print(f"Hello {name}, you are {age} years old.")
```

Output:

```
Enter your name:  Aditi
Enter your age:  21
Hello Aditi, you are 21 years old.
```

Explanation of the Code:

This program demonstrates string formatting by taking the user's **name** and **age** as inputs and displaying them in a formatted message. It first p rompts the user to enter their name and age, which are stored as string variables. The print() function then uses an **f-string** (formatted string lit eral) to dynamically insert the user's name and age into the output mess age: "Hello {name}, you are {age} years old.". This allows for easy and r eadable string formatting, creating a personalized greeting based on the user's input.

Exercise 6: Substring Search

Program:

```
# Substring Search
# Get input from the user
sentence = input("Enter a sentence: ")
word_to_search = input("Enter a word to search in the sentence: ")

# Check if the word exists in the sentence
if word_to_search in sentence:
    position = sentence.index(word_to_search)
    print(f"The word '{word_to_search}' exists in the sentence at index 
{position}.")
else:
    print(f"The word '{word_to_search}' does not exist in the sentence.")
```

Output:

```
Enter a sentence:  Hello World!
Enter a word to search in the sentence:  World
The word 'World' exists in the sentence at index 6
```

Explanation of the Code:

This program allows the user to search for a word within a given senten ce. It first prompts the user to input a sentence and the word they want to search for. The program then checks if the word is present in the sent ence using the in operator. If the word is found, it uses the index() method to determine the position (index) of the word within the sentence and displays the result, indicating the exact location. If the word isn't found, a message is printed stating that the word does not exist in the sentence. This simple approach efficiently searches for substrings and provides feedback based on the presence of the word.

Part 3: Lists

Exercise 7: List Operations

Program:

```
# List Operations
# Get a list of 5 numbers from the user
numbers = [ ]
for i in range(5):
    number = float(input(f"Enter number {i + 1}: "))
    numbers.append(number)

# Display the sum, largest, and smallest number
total_sum = sum(numbers)
largest_number = max(numbers)
smallest_number = min(numbers)

print(f"Sum of all numbers: {total_sum}")
print(f"Largest number: {largest_number}")
print(f"Smallest number: {smallest_number}")
```

Output:

```
Enter number 1:  12
Enter number 2:  2
Enter number 3:  34
Enter number 4:  21
Enter number 5:  2
Sum of all numbers: 71.0
Largest number: 34.0
Smallest number: 2.0
```

Explanation of the Code:

This program allows the user to input a list of 5 numbers and performs three key operations: calculating the sum, finding the largest number, and identifying the smallest number. First, the program uses a loop to collect five numbers from the user, which are converted to floats and added to a list called numbers. Once all the numbers are collected, the program uses the sum() function to compute the total of all numbers in the list. It then uses the max() function to determine the largest number and the min() function to find the smallest number. Finally, the program prints the sum, largest, and smallest values, providing a summary of the se basic operations on the user's list of numbers.

Exercise 8: List Manipulation

Program:

```
# List Manipulation
# Create a list of favorite fruits
fruits = ["Apple", "Banana", "Cherry", "Mango", "Orange"]

# Add a new fruit
fruits.append("Pineapple")

# Remove the second fruit
fruits.pop(1)

# Display the updated list
print(f"Updated fruit list: {fruits}")
```

Output:

```
Updated fruit list: ['Apple', 'Cherry', 'Mango', 'Orange', 'Pineapple']
```

Explanation of the Code:

This program demonstrates basic **list manipulation** by creating a list of five favorite fruits and performing several operations on it. First, the list fruits is initialized with five fruit names: Apple, Banana, Cherry, Mango, and Orange. The program then uses the append() method to add a new fruit, "Pineapple," to the end of the list. Next, it removes the second fruit, "Banana," from the list using the pop() method with an index of 1 (since list indexing starts from 0). Finally, the updated list is printed, showing the remaining fruits after the addition and removal operations. This example highlights how to add and remove elements from a list dynamically.

Exercise 9: Sorting a List

Program:

```
# Sorting a List
# Get a list of 5 numbers from the user
numbers = []
for i in range(5):
   number = float(input(f"Enter number {i + 1}: "))  # Convert input to
float
   numbers.append(number)

# Sort in ascending order
numbers.sort()
print(f"Sorted list in ascending order: {numbers}")

# Sort in descending order
numbers.sort(reverse=True)
print(f"Sorted list in descending order: {numbers}")
```

Output:

```
Enter number 1:  2
Enter number 2:  3
Enter number 3:  4
Enter number 4:  8
Enter number 5:  32
Sorted list in ascending order: [2.0, 3.0, 4.0, 8.0, 32.0]
Sorted list in descending order: [32.0, 8.0, 4.0, 3.0, 2.0]
```

Explanation of the Code:

This program demonstrates how to sort a list of numbers in both ascending and descending order. First, it prompts the user to input 5 numbers, which are converted to floats and stored in the list numbers. The program then uses the sort() method to arrange the numbers in **ascending order** (from smallest to largest) and prints the sorted list. After that, the program sorts the list again, this time in **descending order** (from largest to smallest), by passing the argument reverse=True to the sort() method. Finally, it prints the list sorted in descending order. This showcases how Python's built-in sort() function can be used to rearrange lists in different ways.

Exercise 10: List Slicing

Program:

```
# List Slicing
n = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Print the first 5 elements
print(f"First 5 elements: {n[:5]}")

# Print the last 5 elements
print(f"Last 5 elements: {n[-5:]}")

# Print elements from index 2 to index 7
print(f"Elements from index 2 to 7: {n[2:8]}")
```

Output:

```
First 5 elements: [1, 2, 3, 4, 5]
Last 5 elements: [6, 7, 8, 9, 10]
Elements from index 2 to 7: [3, 4, 5, 6, 7, 8]
```

Explanation of the Code:

This program illustrates the concept of **list slicing** in Python, which allows accessing specific portions of a list using index ranges. The list n contains numbers from 1 to 10. First, the program slices the list to display the first five elements by using n[:5], which starts from the beginning (index 0) and ends just before index 5, giving the result [1, 2, 3, 4, 5]. Next, to retrieve the last five elements, n[-5:] is used, where the negative index -5 starts slicing from the fifth element from the end, resulting in [6, 7, 8, 9, 10]. Finally, the slice n[2:8] extracts elements from index 2 up to, but not including, index 8, giving the range [3, 4, 5, 6, 7, 8]. This technique is useful for efficiently accessing and manipulating specific segments of a list without needing loops or manual indexing.

Bonus Exercise: List Comprehension

Program:

```
# Nested List
# Initialize an empty nested list
students = []

# Get input for 3 students
for i in range(3):
    name = input(f"Enter name of student {i + 1}: ")
    scores = []
    for j in range(3):  # For 3 subjects
        score = float(input(f"Enter score for subject {j + 1}: "))
        scores.append(score)
    students.append([name, scores])

# Calculate and print average score for each student
for student in students:
    name = student[0]
    average_score = sum(student[1]) / len(student[1])
    print(f"{name}'s average score: {average_score:.2f}")
```

Output:

```
Enter name of student 1:  Aditi
Enter score for subject 1:  80
Enter score for subject 2:  70
Enter score for subject 3:  50
Enter name of student 2:  Arindam
Enter score for subject 1:  90
Enter score for subject 2:  80
Enter score for subject 3:  40
Enter name of student 3:  Rik
Enter score for subject 1:  60
Enter score for subject 2:  40
Enter score for subject 3:  60
Aditi's average score: 66.67
Arindam's average score: 70.00
Rik's average score: 53.33
```

Explanation of the Code:

This program demonstrates the use of a **nested list** to store and process student names along with their scores in multiple subjects. The program begins by initializing an empty list students, which will hold the names of students and their corresponding scores. It then iterates to gather input for three students. For each student, the user is prompted to input the student's name, followed by their scores for three subjects. These details are stored in a list where the student's name is paired with their scores, and this list is appended to the main students list.

After collecting the data, the program calculates the average score for each student by summing their subject scores and dividing by the number of subjects. It then prints the student's name alongside their average score, formatted to two decimal places. This approach effectively demonstrates how nested lists can be used to handle multiple sets of data (students and their respective scores) in a structured way, simplifying both data storage and computation.