Python Assignment 1 Report

Name: Saurabh Sen

Course: BCA (Sec A)

Registration Number: PROV/BCA/7/24/001

GitHub Repository Link: https://github.com/saurabhsen848/Python-Assignment-1

Part 1: Operators

Exercise 1: Arithmetic Operators

Problem: 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.

Solution:

```
a = float(input())

b = float(input())

print(a + b, a - b, a * b, round(a / b, 2), a % b, a ** b, a // b)
```

Explanation:

In this code, two numbers are input as floats to handle any decimal values. The code performs all basic arithmetic operations in a single line for each operation and prints the results in a concise manner. Functions like round() are used to format the division output to two decimal places, ensuring a clean result for division operations. Floor division is done using //, and exponentiation uses ** to raise the first number to the power of the second.

Exercise 2: Comparison Operators

Problem: 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.

Solution:

```
a = float(input())

b = float(input())

print(a > b, a == b, a <= b)
```

Explanation:

This code prompts the user to input two numbers and checks the relationship between them using comparison operators. The greater-than (>) operator checks if the first number is larger, while the equality (==) and less-than-or-equal-to (<=) operators check for equality or if the first number is smaller, respectively. The results are printed directly as either True or False based on the comparison outcomes.

Exercise 3: Logical Operators

Problem: 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.

Solution:

```
a = bool(int(input()))

b = bool(int(input()))

c = bool(int(input()))

print(a and b and c, a or b or c, not a)
```

Explanation:

The program takes three boolean inputs from the user, converts them from integers (1 or 0) to True or False. It then applies the and, or, and not logical operators. The and operator checks if all values are True, the or operator checks if at least one value is True, and not negates the first input. This gives a simple understanding of how different logical conditions work.

Exercise 4: String Manipulation

Problem: Take a string input from the user and display the following:

1. The length of the string.

2. The first and last character.

3. The string in reverse order.

4. The string in uppercase and lowercase.

Solution:

```
s = input()

print(len(s), s[0], s[-1], s[::-1], s.upper(), s.lower())
```

Explanation:

In this program, the user provides a string, and the code performs multiple manipulations on it. The len() function calculates the total number of characters, and slicing (s[0] and s[-1]) extracts the first and last characters. Using [::-1] reverses the string. The methods upper() and lower() are applied to convert the string to uppercase and lowercase, respectively.

Exercise 5: String Formatting

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

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

Solution:

```
name = input()

age = input()

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

Explanation:

The program captures the user's name and age as inputs. It then uses f-string formatting, a modern way to embed variables directly into a string, to generate a friendly message. The result is a clean output that includes the user's name and age in a single sentence.

Exercise 6: Substring Search

Problem: 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).

Solution:

```
sentence = input()

word = input()

print(sentence.find(word))
```

Explanation:

This program takes a sentence and a word as inputs from the user. The find() method is used to locate the position of the word within the sentence. If the word exists, the method returns its index; otherwise, it returns -1, indicating the word is not found.

Part 3: Lists

Exercise 7: List Operations

Problem: 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.

Solution:

```
numbers = [float(input()) for _ in range(5)]

print(sum(numbers), max(numbers), min(numbers))
```

Explanation:

This program collects 5 numbers from the user and stores them in a list. The sum() function calculates the total of all elements, while max() and min() find the largest and smallest values in the list. Using list comprehensions simplifies the input collection process and makes the code cleaner.

Exercise 8: List Manipulation

Problem: 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.

Solution:

```
fruits = ["Apple", "Banana", "Cherry", "Mango", "Grapes"]

fruits.append("Orange")

del fruits[1]

print(fruits)
```

Explanation:

The program starts with a predefined list of five fruits. It adds a new fruit using the append() method, which places the new element at the end of the list. The del keyword is used to remove the second fruit by specifying its index (1). The updated list is then printed, reflecting the changes.

Exercise 9: Sorting a List

Problem: 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.

Solution:

```
numbers = [float(input()) for _ in range(5)]

print(sorted(numbers), sorted(numbers, reverse=True))
```

Explanation:

This program accepts five numbers from the user and stores them in a list. The sorted() function is used to sort the list in ascending order by default. To sort in descending order, the reverse=True argument is passed to sorted(). The results are displayed for both ascending and descending orders.

Exercise 10: List Slicing

Problem: 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.

Solution:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(numbers[:5], numbers[-5:], numbers[2:8])
```

Explanation:

This program demonstrates list slicing. The first slice (:5) selects the first 5 elements, the second slice (-5:) grabs the last 5 elements, and the final slice (2:8) extracts the elements from index 2 to 7. This shows how to efficiently access specific sections of a list.

Exercise 11: Nested List

Problem: 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.

Solution:

```
students = [[input(), [float(input()), float(input()), float(input())]] for _ in range(3)]

for s in students:
    print(s[0], sum(s[1]) / 3)
```

Explanation:

This program collects the names and scores of three students in three subjects. It calculates and displays each student's average score after all input has been gathered. The use of nested lists allows for organized storage of related data, making it easier to perform calculations on each student's scores.