**“Software Design And Construction”**

**BY:** Bushra Ashraf Bhatti

**Reg\_no:** 2023-BSE-015

**Submitted to:** Engr. Muhammad Shahzad

**Lab 03:**

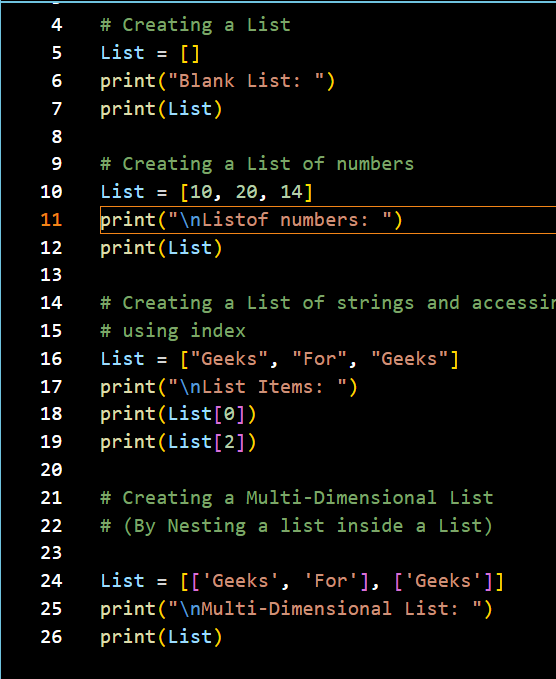
**“Python Lists Tuples Sets Dictionaries”**

**Python Collections (Arrays)**

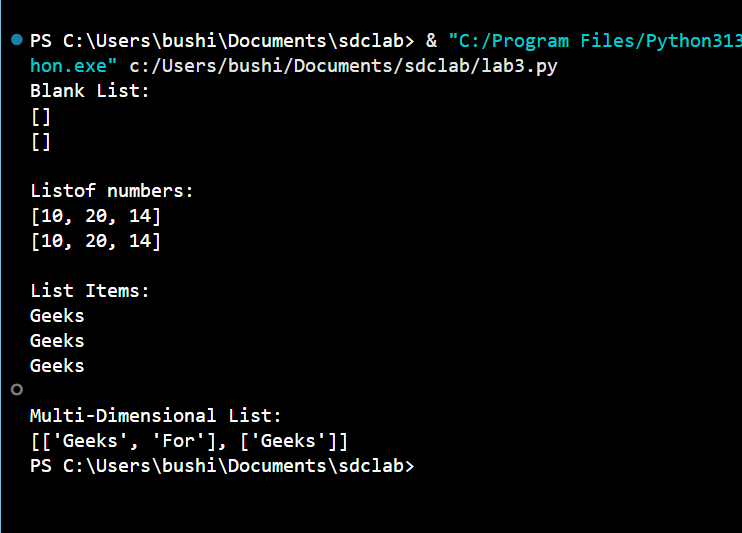
**There are four collection data types in the Python programming language:**

1. **Python Lists: List is a collection which is ordered and changeable. Allows duplicate members**

**Lists in Python can be created by just placing the sequence inside the square brackets [].**

****

**Output:**

****

**Creating a list with multiple distinct or duplicate elements:**

**A list may contain duplicate values with their distinct positions and hence, multiple distinct or**

**duplicate values can be passed as a sequence at the time of list creation.**

list=[1,2,4,4,3,3,3,6,5]

print("\nlist with the use of numbers")

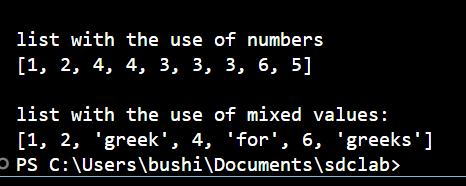
print(list)

list=[1,2,'greek',4,'for',6,'greeks']

print("\nlist with the use of mixed values:")

print(list)

**output:**

****

**Knowing the size of List**

# Creating a List

List1 = []

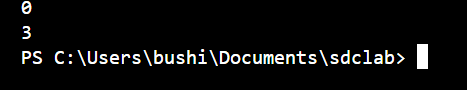
print(len(List1))

#Creating a List of numbers

List2 = [10, 20, 14]

print(len(List2))

**output:**

****

**Adding Elements to a List**

**Using append() method:**

my\_list = []

print("Initial blank List:")

print(my\_list)

my\_list.append(1)

my\_list.append(2)

my\_list.append(4)

print("\nList after addition of three elements:")

print(my\_list)

for i in range(1, 4):

    my\_list.append(i)

print("\nList after addition of elements from 1-3:")

print(my\_list)

my\_list.append((5, 6))

print("\nList after addition of a Tuple:")

print(my\_list)

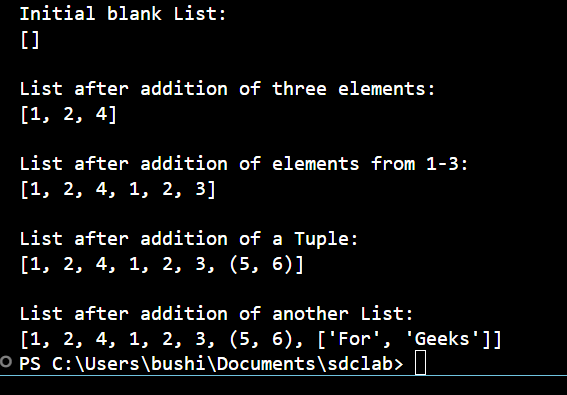
list2 = ['For', 'Geeks']

my\_list.append(list2)

print("\nList after addition of another List:")

print(my\_list)

**output:**

****

**Using insert() method**

List =[1,2,3,4]

print("Initial List: " )

print(List)

# Addition of Element at

# specific Position

# (using Insert Method)

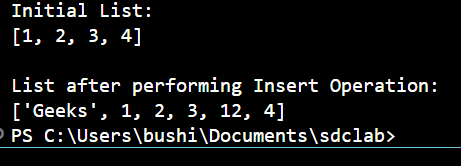
List.insert(3, 12)

List.insert(0, 'Geeks')

print("\nList after performing Insert Operation: ")

print(List)

**output:**

****

**Using extend() method**

#==================================================

list=[1,2,3,4]

print("initial list:")

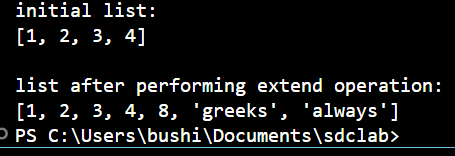
print(list)

list.extend([8,'greeks','always'])

print("\nlist after performing extend operation:")

print(list)

**output:**

****

**Accessing elements from the List**

#========================================================

# Python program to demonstrate

# accessing of element from list

# Creating a List with

# the use of multiple values

List = ["Geeks", "For", "Geeks"]

# accessing a element from the # list using index number

print("Accessing a element from the list" )

print(List[0])

print(List[2])

# Creating a Multi-Dimensional List

# (By Nesting a list inside a List)

List = [['Geeks', 'For'], ['Geeks']]

# accessing an element from the

# Multi-Dimensional List using

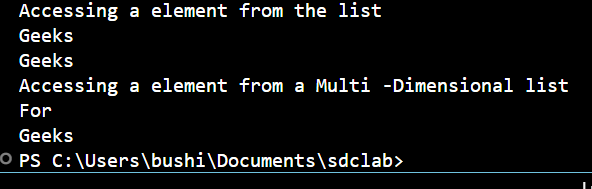
# index number

print("Accessing a element from a Multi -Dimensional list" )

print(List[0][1])

print(List[1][0])

**output:**

****

**Negative indexing**

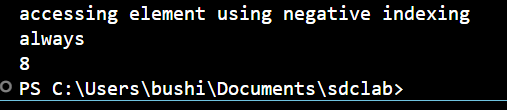
List = [1,2,'Geeks',4, 'For',6, 'Geeks']

print("accessing element using negative indexing")

print(list[-1])

print(list[-3])

**output:**

****

**(Removing Elements from the List)**

**Using remove() method:**

# Python program to demonstrate

# Removal of elements in a List

# Creating a List

List = [1, 2, 3, 4, 5, 6,

7, 8, 9, 10, 11, 12]

print("Initial List: ")

print(List)

# Removing elements from List

# using Remove() method

List.remove(5)

List.remove(6)

print("\nList after Removal of two elements: ")

print(List)

# Removing elements from List

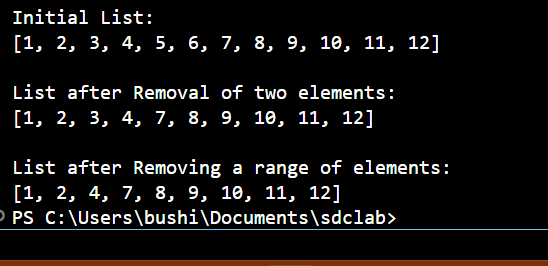
# using iterator method for i in range(1, 5):

List.remove(i)

print("\nList after Removing a range of elements: ")

print(List)

**output:**

****

**Using pop() method**

List = [1,2,3,4,5]

# Removing element from the

# Set using the pop() method

List .pop()

print (" \nList after popping an element: " )

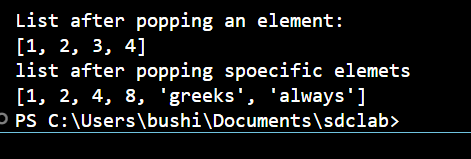
print ( List)

list.pop(2)

print("list after popping spoecific elemets")

print(list)

**output:**

****

**Slicing of a List**

# Creating a List

List = ['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']

print("Initial List:")

print(List)

# Slicing elements in a range 3-8

print("\nSlicing elements in a range 3-8:")

Sliced\_List = List[3:9]

print(Sliced\_List)

# Print elements from a pre-defined point to end

Sliced\_List = List[5:]

print("\nElements sliced from 5th element till the end:")

print(Sliced\_List)

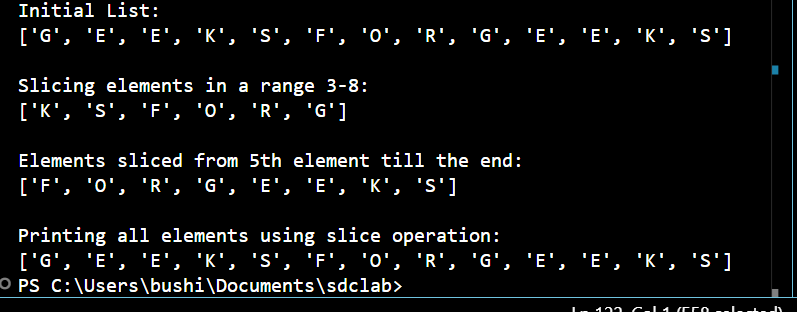
# Printing elements from beginning till end

Sliced\_List = List[:]

print("\nPrinting all elements using slice operation:")

print(Sliced\_List)

**output:**

****

**Negative index List slicing**

#======================================================

List = ['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S']

print("Initial List: ")

print(List)

# Print elements from beginning

# to a pre-defined point using Slice

Sliced\_List = List[:-6]

print("\nElements sliced till 6th element from last: ")

print(Sliced\_List)

# Print elements of a range

# using negative index List slicing

Sliced\_List = List[-6:-1]

print("\nElements sliced from index -6 to -1")

print(Sliced\_List)

# Printing elements in reverse

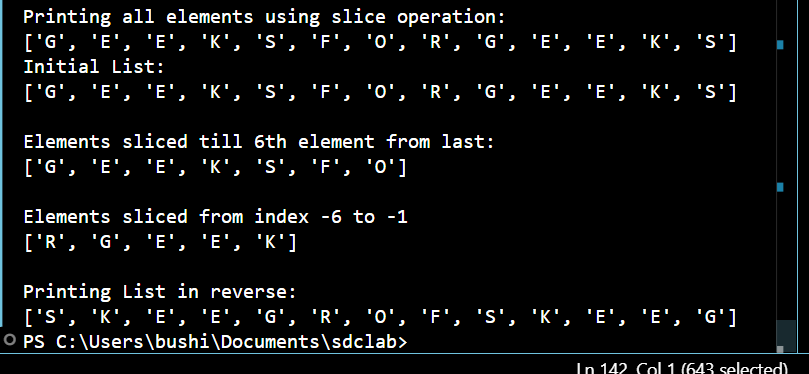
# using Slice operation

Sliced\_List = List[::-1]

print("\nPrinting List in reverse: ")

print(Sliced\_List)

**output:**

****

**Tuple**

Tuples are used to store multiple items in a single variable. A tuple is a collection which is ordered and unchangeable. Tuples are written with round brackets.

Thistuple=("apple", "banana", "cherry") print(thistuple)

**Tuple Items**

Tuple items are ordered, unchangeable, and allow duplicate values. Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

**Ordered**

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

**Unchangeable**

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

**Allow Duplicates**

Since tuples are indexed, they can have items with the same value:

**Example**

Tuples allow duplicate values: thistuple=("apple",

"banana", "cherry", "apple", "cherry") print(thistuple)

**Tuple Length**

To determine how many items a tuple has, use the len() function:

**Example**

Print the number of items in the tuple: thistuple=("apple", "banana", "cherry") print(len(thistuple))

Create Tuple With One Item

Note! To create a tuple with only one item, you have to add a comma after the item,

otherwise Python will not recognize it as a tuple**.**

**Example**

One item tuple, remember the comma:

thistuple=("apple",) print(type(thistuple))

**Tuple Items - Data Types**

Tuple items can be of any data type:

**Example**

String, int, and boolean data types:

tuple1=("apple", "banana", "cherry")

tuple2=(1, 5, 7, 9, 3) tuple3 = (True,

False, False)

A tuple can contain different data types:

**Example**

A tuple with strings, integers, and boolean values:

tuple1 = ("abc", 34, True, 40, "male")

**Set**

A set is a collection which is unordered, unchangeable\*, and unindexed.

**Create a Set:**

thisset={"apple", "banana", "cherry"} print(thisset)

Note: Sets are unordered, so you cannot be sure in which order the items will appear.

**Set Items**

Set items are unordered, unchangeable, and do not allow duplicate values.

**Unordered**

Unordered means that the items in a set do not have a defined order. Set items can appear in a different order every time you use them and cannot be referred to by index or key.

**Unchangeable**

Set items are unchangeable, meaning that we cannot change the items after the set

has been created. Once a set is created, you cannot change its items, but you can remove items and add new items.

**Duplicates Not Allowed**

Sets cannot have two items with the same value.

**Example**

Duplicate values will be ignored:

thisset={"apple", "banana", "cherry", "apple"} print(thisset)

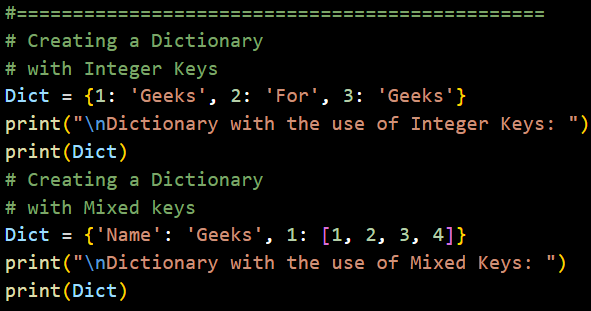
**Dictionary:**

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which, unlike other Data Types that hold only a single value as an element, Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more

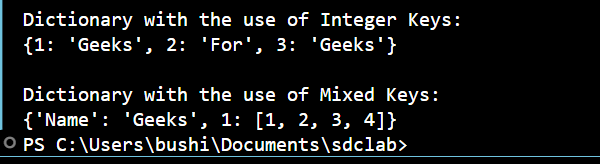
optimized

**Note – Keys in a dictionary don’t allow Polymorphism.**

**Creating a Dictionary:**

****

**Output:**

****

Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing to curly braces{}.

#====================================================

# Creating an empty Dictionary

Dict = {}

print ("Empty Dictionary: " )

print ( Dict )

# Creating a Dictionary

# with dict() method

Dict = dict ({ 1: 'Geeks' , 2: 'For' , 3: 'Geeks' })

print (" \nDictionary with the use of dict(): " )

print ( Dict )

# Creating a Dictionary

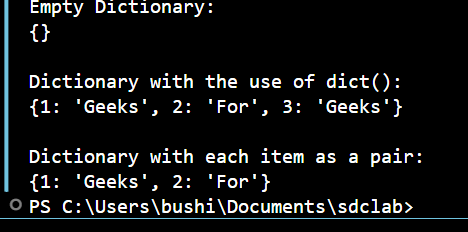
# with each item as a Pair

Dict = dict ([( 1 , 'Geeks') , ( 2, 'For' )])

print (" \nDictionary with each item as a pair: " )

print ( Dict)

**output**:



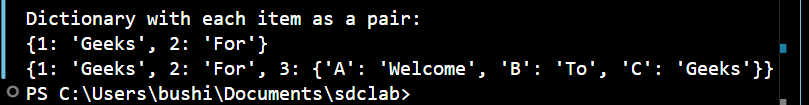
# Creating a Nested Dictionary

# as shown in the below image

Dict = {1: 'Geeks', 2: 'For',

3:{'A' : 'Welcome', 'B' : 'To', 'C' : 'Geeks'}}

**Output:**

****

**Adding elements to a Dictionary:**

# Creating an empty Dictionary

Dict = {}

print("Empty Dictionary: ")

print(Dict)

# Adding elements one at a time

Dict[0] = 'Geeks'

Dict[2] = 'For'

Dict[3] = 1

print("\nDictionary after adding 3 elements:")

print(Dict)

# Adding set of values

# to a single Key

Dict['Value\_set'] = 2, 3, 4

print("\nDictionary after adding 3 elements: ")

print(Dict)

# Updating existing Key's Value

Dict[2] = 'Welcome'

print("\nUpdated key value: ")

print(Dict)

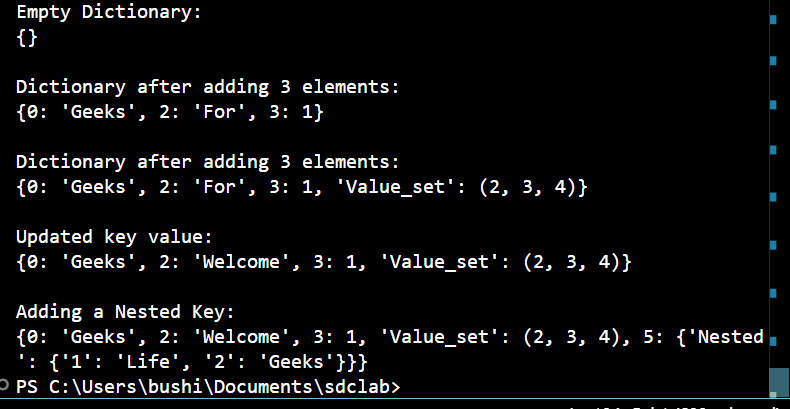
# Adding Nested Key value to Dictionary

Dict[5] = {'Nested' :{'1' : 'Life', '2' : 'Geeks'}}

print ( "\nAdding a Nested Key: ")

print ( Dict)

**output:**

****

**Accessing elements from a Dictionary:**

# Python program to demonstrate

# accessing a element from a Dictionary

Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

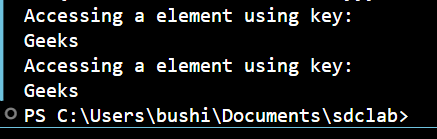
print("Accessing a element using key:")

print(Dict[1])

print("Accessing a element using key:")

print(Dict[1])

**output:**

****

#===============================================

# Creating a Dictionary

Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

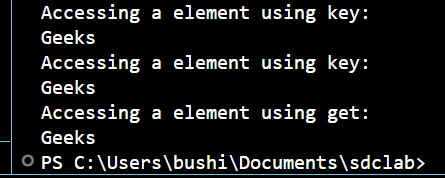
# accessing a element using get()

# method

print("Accessing a element using get:")

print(Dict.get(3))

**output:**

****

**Accessing an element of a nested dictionary:**

# Creating a Dictionary

Dict = {

    'Dict1': {1: 'Geeks'},

    'Dict2': {'Name': 'For'}

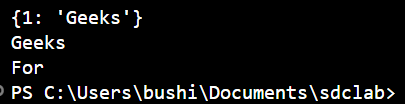
}

print(Dict['Dict1'])

print(Dict['Dict1'][1])

print(Dict['Dict2']['Name'])

**output:**

****

**Removing Elements from Dictionary**

**Using del keyword:**

# Initial Dictionary

Dict = { 5 : 'Welcome', 6 : 'To', 7 : 'Geeks',

'A' : {1 : 'Geeks', 2 : 'For', 3 : 'Geeks'},

'B' : {1 : 'Geeks', 2 : 'Life'}}

print("Initial Dictionary: ")

print(Dict)

# Deleting a Key value del

Dict[6]

print("\nDeleting a specific key: ")

print(Dict)

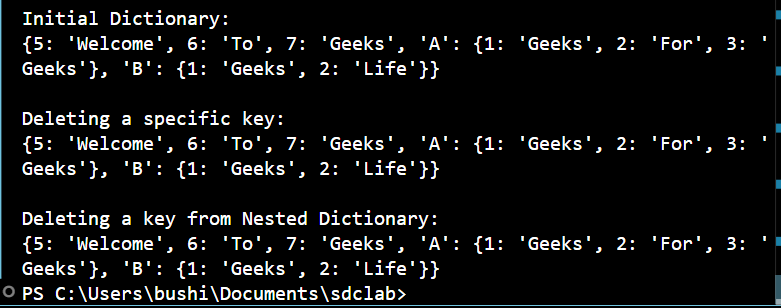
# Deleting a Key from

# Nested Dictionary del Dict['A'][2]

print("\nDeleting a key from Nested Dictionary: ")

print(Dict)

**output:**

****

**Using pop() method:**

# Creating a Dictionary

Dict = { 1: 'Geeks' , 'name': 'For' , 3 :

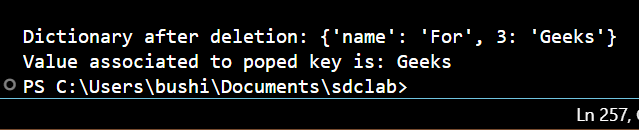
'Geeks' }

pop\_ele = Dict .pop( 1)

print ( '\nDictionary after deletion: ' + str( Dict ))

print ( 'Value associated to poped key is: ' + str (pop\_ele))

**output:**

****

**Using popitem() method:**

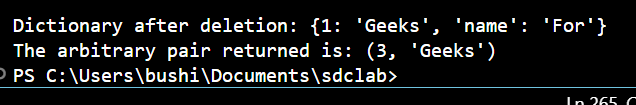
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

pop\_ele = Dict.popitem()

print("\nDictionary after deletion: " + str(Dict))

print ("The arbitrary pair returned is: " + str (pop\_ele))

**output:**

****

**Using clear() method:**

#---------------------

# Creating a Dictionary

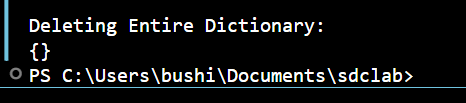
Dict = { 1: 'Geeks' , 'name': 'For' , 3 : 'Geeks' }

Dict .clear()

print (" \nDeleting Entire Dictionary: " )

print ( Dict)

**output:**

****

**Task 1.**

Accept two lists from user and display their join.

# Accept two lists from the user and display their join

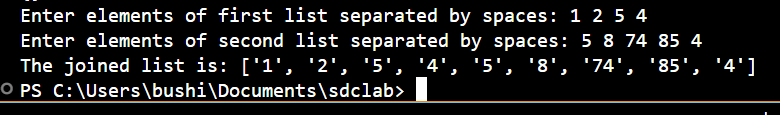
list1 = input("Enter elements of first list separated by spaces: ").split()

list2 = input("Enter elements of second list separated by spaces: ").split()

joined\_list = list1 + list2

print("The joined list is:", joined\_list)

**output:**

****

**Task 2.**

Imagine two matrices given in the form of 2D lists as under; a = [[1, 0, 0], [0, 1, 0], [0, 0, 1] ] b =

[[1, 2, 3], [4, 5, 6], [7, 8, 9] ]. Write a Python code that finds another matrix/2D list that is a product of and b, i.e., C=a\*b

a = [[1, 0, 0],

     [0, 1, 0],

     [0, 0, 1]]

b = [[1, 2, 3],

     [4, 5, 6],

     [7, 8, 9]]

c = [[0 for \_ in range(len(b[0]))] for \_ in range(len(a))]

for i in range(len(a)):          # rows of a

    for j in range(len(b[0])):   # columns of b

        for k in range(len(b)):  # rows of b

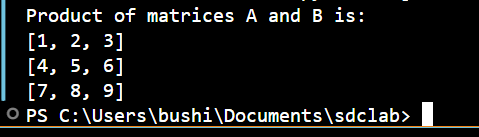
            c[i][j] += a[i][k] \* b[k][j]

print("Product of matrices A and B is:")

for row in c:

    print(row)

**output:**

****

**Task 3.**

Create a Python program that contains a dictionary of names and phone numbers. Use a tuple of separate first and last name values for the key field. Initialize the dictionary with at least three names and numbers. Ask the user to search for a phone number by entering a first and last name. Display the matching number if found, or a message if not found.

# Dictionary with tuple keys (first name, last name)

phone\_book = {

    ('ali', 'khan'): '0300-1234567',

    ('sara', 'ahmed'): '0321-9876543',

    ('bilal', 'hussain'): '0333-4567890'

}

while True:

    # Ask user for input

    first = input("Enter first name: ").strip().lower()

    last = input("Enter last name: ").strip().lower()

    key = (first, last)

    # Search in the dictionary

    if key in phone\_book:

        print(f"✅ Phone number of {first.title()} {last.title()} is: {phone\_book[key]}")

        break

    else:

        print(f"Sorry, phone number for {first.title()} {last.title()} not found.")

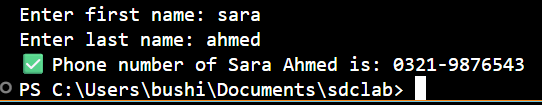
        again = input("Do you want to try again? (yes/no): ").strip().lower()

        if again != "yes":

            print("Goodbye!")

            break

**Output:**

****

**Task 4:**

Create two lists based on the user values. Merge both the lists and display in sorted order.

# Create two empty lists

list1 = []

list2 = []

n1 = int(input("How many elements in the first list? "))

for i in range(n1):

    value = int(input(f"Enter element {i+1} for first list: "))

    list1.append(value)

# Take input for second list

n2 = int(input("How many elements in the second list? "))

for i in range(n2):

    value = int(input(f"Enter element {i+1} for second list: "))

    list2.append(value)

# Merge the two lists

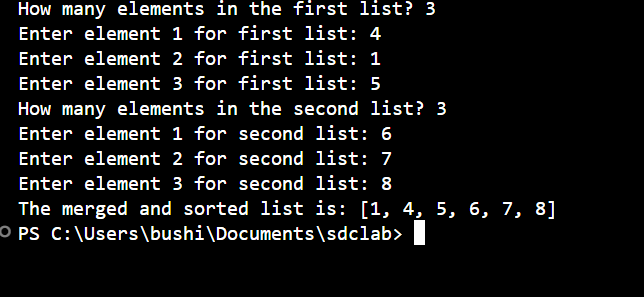
merged\_list = list1 + list2

# Sort the merged list

merged\_list.sort()

print("The merged and sorted list is:", merged\_list)

**output:**

****

**Task 5.**

Repeat the above activity to find the smallest and largest element of the list. (Suppose

all the elements are integer values)

list1 = []

list2 = []

# Take input for first list

n1 = int(input("How many elements in the first list? "))

for i in range(n1):

    value = int(input(f"Enter element {i+1} for first list: "))

    list1.append(value)

# Take input for second list

n2 = int(input("How many elements in the second list? "))

for i in range(n2):

    value = int(input(f"Enter element {i+1} for second list: "))

    list2.append(value)

# Merge both lists

merged\_list = list1 + list2

# Display merged list

print("\nMerged list:", merged\_list)

# Find smallest and largest

smallest = merged\_list[0]

largest = merged\_list[0]

for num in merged\_list:

    if num < smallest:

        smallest = num

    if num > largest:

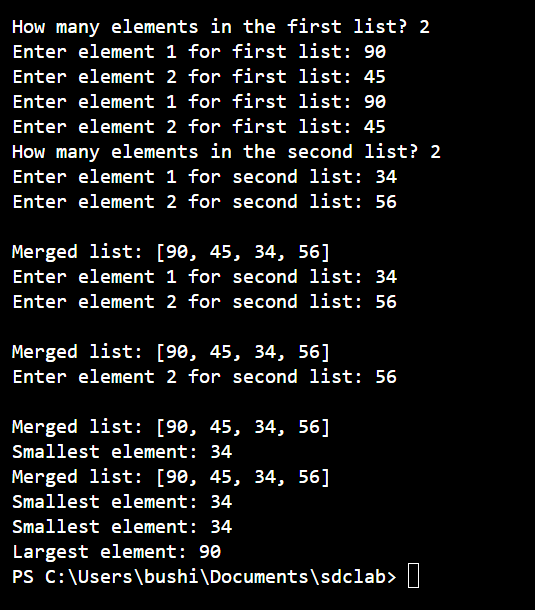
        largest = num

# Display results

print("Smallest element:", smallest)

print("Largest element:", largest)

**output:**

****

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*