Sample Input / Output :-

```
enter the seven days sales count 100
enter the seven days sales count 450
enter the seven days sales count 1250
enter the seven days sales count 589
enter the seven days sales count 98
enter the seven days sales count 348
enter the seven days sales count 900
enter the seven days sales count 239
Sales (Mon... Sun) : [100, 450, 1250, 589, 98, 348, 900, 239]
Total : 3974
Average : 567.71
Best day : 3 with 1250
Worst day : 5 with 98
```

4. Use various data types, List Tuples and Dictionary

Task-4: use various types, List, Tuples and Dictionary in phython programming Key Terms covered: Data types List, Tuple, set, Dict Tags - Easy, CO1, S5

4.1 List - cafeteria sales

In your college cafeteria the sales (in units) of a new snack are recorded for 7 days, Monday to sunday. sotre these values in a list, then find the total and aver-age sales, identify the best and worst sales days using index().

Aim: Record a cafeteria's snack sales for 7 days using a list; compute total and average sales, find the best/worst day, and count how many days crossed a

Algorithm:

1. start
2. create an empty list sales = [].
3. For 7 days, append integer sales to the list using append ().
4. compute total = sum(sales) and avg = total/7
5. Find max_val = max (sales), min_val = min (sales).
6. Find corresponding days with index() (add +1 to convert to day number).
7. Count days above a target using count () on a boolean re-map or with a loop.
8. stop

Program (uses append(), index(), count()):

```
# List scenario
days = 7
sales = []
target = 500 # target sales for the day

for s in range(8):
    sample_entries = int(input("enter the seven days sales count"))
```

```
sales.append(sample_entries) # list.append()

total = sum(sales)
avg = total/days

max_val = max(sales)
min_val = min(sales)
best_day = sales.index(max_val)+1 # list.index()
worst_day = sales.index(min_val)+1

Pint("Sales (Mon-Sun):", sales)
Print("Total:", total)
Print("Average:", round(avg,2))
Print("Best Day:", best_day, "with", max_val)
Print("Worst Day:", worst_day, "with", min_val)
```

Sample Input / Output:

All lab slots : (9, 11, 14, 16, 14)

Is 14:00 present? True

14:00 occurs 2 times (s)

First occurrence position (1-based): 3

Morning slots : (9, 11)

Afternoon slots : (14, 16, 14)

4.2 Tuple - Lab Timetable

your department has a fixed daily lab shedule represe-nted by a tuple of starting hours (24-hour format). Write a program to check if a given start time exists in the tuple, count how many times it happears using count(). find its first position using index(), and display morning and afternoon slots using slicing.

Aim:

To manage a query an immutable daily lab slot she-dule using a tuple demonstrating membership checks, count(), index(), and slicing.

Algorithm

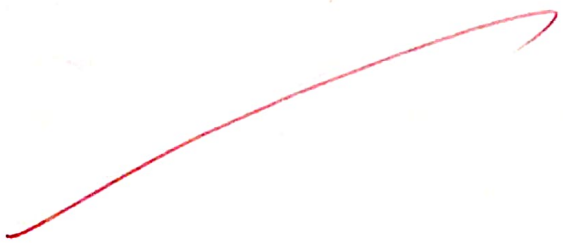
1. start
2. Define slots as a fixed tuple of integers
3. Read query hour.
4. Check ~~con~~existence with query in solts
5. use count(); if positive, use index() to find the first position.
6. slice into morning and afternoon.
7. print results.
8. stop

Python program

```
# TUPLE scenario
slots = (9,11,14,16,14) # immutable daily shedule
query = 14
exists = (query in slots)
freq = slots.count(query)  # tuple.count()
first_pos = solts.index(query) + 1 if exists else "N/A"

morning = solts[:2]
afternoon = solts[2:]
```

```
Print("All lab solts": slots)
Print (f" Is {query} :Do present ?", exists)
Print (f" {query} :oo occurs ", freq, "time(s)")
Print ("First occurrence position (1-based):", first - pos)
Print ("Morning slots:", morning)
Print ("Afternoon slots:", afternoon)
```



Enter number of items in price list : 3.
Enter item name : box
Enter price of box : 15
Enter item name : pen
Enter item name : pencil
Enter price of pen : 10
Enter price of pencil : 5
Enter item to update price (or press Enter to skip): box
Enter new price for box : 20
Enter an item to remove from price list (or press Enter to skip) : pen
Available items : ['box', 'pencil']
prices : (20.0, 5.0)
Costliest item : box at 20.0
Removed 'pen' price (if existed) : 100.0

4.3

Dictionary - Bookstore Billing

A bookstore's price list is stored in a dictionary where keys are item names and values are prices. update the price of an item using update(), find the costliest item using max() on items(), remove an out-of-stock item using pop(), and display all keys, values, and items.

Aim:

To manage live prices list and bill a customer using dictonary methods and views.

Algorithm:

1. start
2. create an empty dictionary prices.
3. Ask the user for the number of items in the price list (n).
4. Repeat for each item;
5. Get the item name.
6. Get the item price.
7. Add the item and price to prices.
8. Ask the user for an item to update (or press Enter to skip)
9. If the item exists in prices, get the new price and update it.
10. Find the costliest item by checking each item's price.
11. Ask the user for an item to remove (or press enter to skip).
12. If given, remove that item from prices.
13. show all available items, their prices, the costilest item, and the removed item's price
14. stop.

Python Program

```
Prices = {}
n1 = int(input("Enter number of items in price
                                   list: "))
for _ in range (n1):
  item = input ("Enter item name: ")
  price = float (input (f"Enter price of{item}:"))
  prices [item] = price
# optional price version revision
rev_item = input ("Enter item to update price (or
                         press Enter to skip): ")
if rev_item in prices:
   new_price = float (input (f"Enter new price for
                                    {rev_item}:"))
   prices.update ({rev_item: new_price) # dict.update()

# Find costliest item
 costliest_item = None
 max_price = 0
 for item, price in price.items():
 if price > max_price:
     max_price = price
     costliest_item = item

# Remove out-of-stock item
 if remove_item:
    removed_price = prices.pop(remove_item, None) # dict.pop()

# Display results
Print("\n Available items:", list (Prices.keys())) # dict.keys()
Print ("Prices:", list(Prices.values()))   #dict.values()
if costliest_item:
   Print ("costliest item:", costliest_item, "at", max_price)
 if remove_item:
```

Enter the no. of participants in AI Hackthon : 4

enter participant ID : A1

A2

Enter

A4

enter

A5

enter

enter number of participants in robotics challenges : 4

enter participant ID : A1

: A2

: A3

: A7

Enter late registrant ID for AI Hackthon : A8

Enter withdraw participant ID from robotics challenge : A1

AI Hackathon : {'A1', 'A4', 'A5', 'A8', 'A2'}

Robotics challenge : {'A2', 'A7', 'A3'}

Both events : {'A2'}

Only AI : {'A4', 'A8', 'A5', 'A1'}

only Robotics : {'A7', 'A3'}

Total unique participants : 7

4.4

Two events, AI Hackthon and Robotics Challenge, have Participants' IDs stored in two sets. Add a late registration to AI Hackthon, remove a withdrawn Participant from Robotics using discard(). then find participants in both events (intersection()), only in one (difference()), the total unique participants (union()).

Aim :- To implement the python program for the participants IDs storing.

Algorithm :-

1. Start
2. get AI hackathon participants details.
3. Done the set operations.
4. Print the Hackathon details of the Participants.

```
# Get AI Hackthon participants
ai_hackathon = set()
n1 = int(input("Enter number of participants in AI
    Hackathon:"))
for _ in range(n1):
    pid = input("Enter number of participants in
        AI Hackathon:")
    Pid = input("Enter participant ID:")
    ai_hackathon.add(pid)
```

```
# Get Robotics challenge participants
robotics_challenge = set()
n2 = int(input("Enter number of participants in
        Robotics challenge:"))
for _ in range(n2):
    pid = input("Enter participant ID:")
    robotics_challenge.add(pid)

# Add a late registrant
late_id = input("Enter number of participants in
        Robotics challenge):
late_id = input("Enter late registrant ID for AI
    Hackthon (or press Enter to skip):")
if late_id:
    ai_hackathon.add(late_id) # set add()

# Remove a withdrawn participant
remove_id = input("Enter withdrawn participant ID
    from Robotics challenge (or press Enter to skip):")
if remove_id:
    robotics_challenge.discard(remove_id) # set discard()

# Set operations
both = ai_hackathon.intersection(robotics_challenge)
only_ai = ai_hackathon.difference(robotics_challenge)
only_robotics = robotics_challenge.difference(ai_hackathon)
unique_all = ai_hackathon.union(robotics_challenge)
```

```
# Output
print("\n AI Hackathon:", ai_hackathon)
print("Robotics Challenge:", robotics_challenge)
only
Print("Both events:", both)
Print("Only AI:", only_ai)
Print("Only Robotics:", only_robotics)
Print("Total unique participants:", len(unique_all))
```

VEL TECH	
EX NO.	4
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	
TOTAL (20)	
SIGN DATE	14

Result:- Thus the implementation of various tuples, list, data types and dictionary was successfully executed.