Dated: 27/8/25 Task - 4

Use various data-type, List, Tuples and dictionary in python programming key terms covered: Data types, List, Tuple, Set, Set, Dict.

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. Store these values values in a list, then find the total and average 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 target

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 (use append(), index(), count()):

```
# LIST Scenario.
days = [1, 2, 3, 4, 5, 6, 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()
```

Sample Input/Output

4
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 sale count 589
enter the seven days sales count 98
enter the seven days sales count 348
enter the seven days count 900
enter the seven days sales count 259

Sales (Mon - Sun): 100, 450, 1250, 589, 98, 348, 900, 259.

Total : 3994

Average : 569.71

Best Day : 2 with 1250

Worst Day : 5 with 98

```
worst_day = sales.index(min_val)+1
print("sales (Mon...Sun):", sales
print("total:", total)
print("Average:", round(avg,2))
print("Best:", best_day, "with", max_val)
print("Worst:", worst_day, "with", min_val)
```

Result:- Thus, the python program of Record Calculation using list is executed successfully

4.2 Tuple - lab timetable

Your department has a fixed daily lab schedule represented 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 appears using count() find its first position using index() and display morning and afternoon slots using slicing

Aim:-

To manage and query an immutable daily lab slot schedule 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 existence with query in slots
5. Use count(); if positive, use index() to find the first position.
6) Slice into morning and afternoon
7) Print results.
8) Stop.

Sample output

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

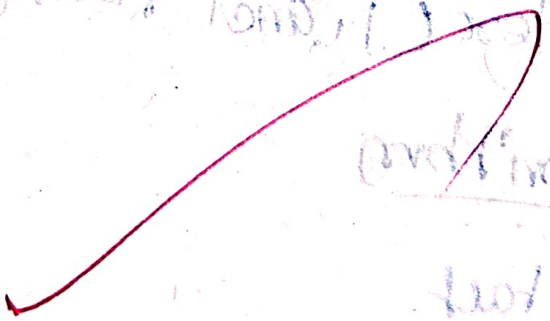
is 14:00 present? true.

14:00 occurs 2 time(s)

first occurrence position (1-based) = 3

Morning slots = (9, 11)

Afternoon slots = (14, 16, 14)



Python Program

```
# TUPLE scenario.
slots = (9,11,14,16,18) # immutable daily schedule
query = 14
exists = (query in slots)
freq = slots.count(query) # tuple.count()
first_pos = slots.index(query) + 1 if exists
            else "N/A" # tuple.index()
morning = slots[:2]
afternoon = slots[2:]
print(f"Is {query} present?", exists)
print("First occurrence position (1-based):",
      first_pos)
Print("Morning slots:", morning)
print("Afternoon slots:", afternoon)
```

Result:- Thus, the python program is manage immutable daily slot is executed successfully

Dictionary - Bookstore Billing.

A. bookstore's price list is stored in a dictionary where keys 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 a live price list and bill a customer using dictionary 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 user for an item to update (or press Enter to skip).
8. Ask the user item and price to prices

9. If the item exists in prices, get the new price and update it.

10. find the costlies item by checking each item's prices

11. Ask the user for an item to remove (or press enter to skip).

12. if given remove that item to that item from prices

13. Show all available items, their prices the costliest 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
```

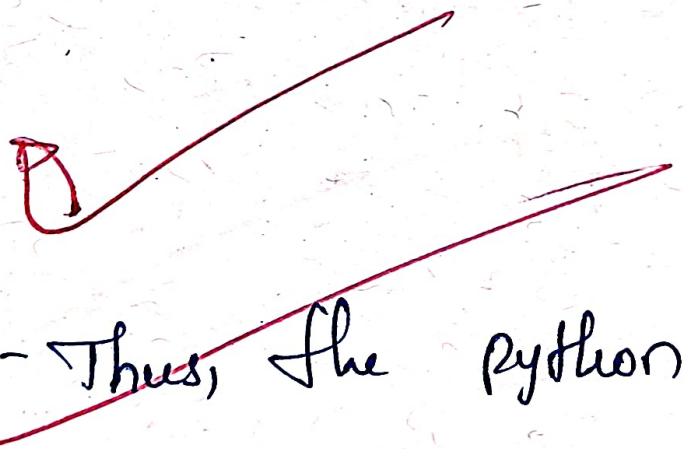
```
# optional price revision
rev_item = input("Enter item to update
    price (or press Enter to skip): ")
if rev_item in prices:
    new_price = float(input(f"Enter new
        price for {rev_item}: "))
    prices.update({rev_item: new_price})
                # dict.update()
# Find costliest item.
costliest_item = None
max_price = 0
for item, price in prices.items():
    if price > max_price:
        max_price = price
        costliest_item = item
# Remove out-of-stock item
removed_item = None
if remove_item:
    removed_price = prices.pop(remove_
                item, None) # dict.pop()
# Display results
```

Sample Input/output:

Enter number of items in price list: 3
Enter item name: box
Enter price of box: 15.
Enter item of name: pen
Enter price of pen: 10
Enter item name: pencil
Enter price of pencil: 5
Enter item to update price for.
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 entered): 10.0

```
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:
    Print (f"Removed {remove_item} price
            (it existed):", removed_price)
```



Result:- Thus, the python program is manage live price bill customer is executed successfully

4.4 Set- Tech Fest Participation.

Two events : AI Hackathon and Robotics.

Challenge, have participants IDs stored into sets Add a late registrant to AI Hackathon remove a withdrawn participant from Robotics using discard() then find participants in both events (intersection()). only-in one (difference()). the total unique participants (union()).

```
# Get AI Hackathon participants
ai_hackathon = set()
n1 = int(input("Enter number of participants in AI Hackathon:"))
for _ in range(n1):
    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_challenges.add(pid)
# Add a late registrant
late_id = input ("Enter late registrant ID
    for AI Hackathon (or press Enter to skip): ")
    if late_id:
    ai_hackathon.add (late_id) # set.add()

# Remove a withdrawn participants
remove_id = input ("Enter withdrawn
    participants ID from Robotics Challenges
        (or press Enter to skip): ")

    if remove_id:
        robotics_challenges.discard (remove
            _id) # set.discard.

# set operations

both_ai_hackathon.intersection (robotics.
                                challenges)
only_ai = ai_hackathon.difference
            (robotics_challenge)
```

Output:-

Print ("\n\n Hackathon :", ai_hackathon)

Print ("Robotics challenge :", robotics_challenge)

Print ("Both events :", both)

Print ("Only AI :", only_ai)

Print ("Only Robotics :", only_robotics)

Print ("Total unique participants:", len (unique_all));

only_robotics = robotics_challenge.
difference (ai_hackathon)
unique = all=ai_hackathon.union
(robotics_challenge)

VEL TECH	
EX NO.	4
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	
TOTAL (20)	
SIGN WITH DATE	15

Result: Thus, the program is variables types are successfully executed.