Task 4:

Date:- 20/8/25 Cafeteria Sales

4.1: cafeteria series

In your college cafeteria, the sales (in units) of a new snack are recorded 7 days, Monday to Sunday. Store these 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

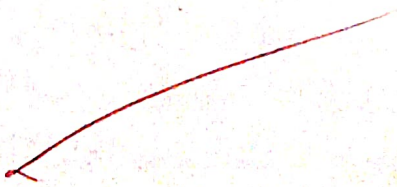
```
# LIST  Scenario
days = 7
Sales = []
target = 500 # target sales for the day
for s in range(8)
    Sample_entries = int(input("enter the seven days Sales Count"))
    Sales.append(Sample_entries) # list.append()
total = sum(sales)
avg = total/days
max_val = max(sales)
min_val = min(sales)
best_day = sales.index(max_val) + 1 # list.index()
```

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

```
worst_day = sales.index(min_val) + 1
print("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)
```



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 slots 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

Python program

```
# TUPLE Scenario
Slots = (9, 11, 14, 16, 14) # 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 ("All lab slots!, slots)
print (f"Is {query?:00 present?; exists)
print (f" {query?:00 occurs ", freq, "time(s)")
```

```
print("first occurance position (1-based):", first_pos)
print("Morning slots:", morning)
print("Afternoon slots:", afternoon)
```

Sample Input /output :-

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

Is 14:00 present ? True

14:00 occurs 2 times(s)

first occurence position (1-based):3

morning slots : (9,11)

Afternoon slots :(14,16,14)

Output :-

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

is 14:00 present? True

14:00 occurs 2 time(s)

first occurance position (1-based): 3

morning slots: (9, 11)

afternoon slots: (14, 16, 14)

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 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(n1)
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
9. if the item exists in prices, get the new price and update it
10. find the costlist item by checking each item's price.
11. Ask the user for an item to remove
12. if given, remove that item from prices
13. Show all available items, their prices, the costliest item, and the remove 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("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.opdate()
# 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
remove_item = input("Enter an item to remove from price list (or
                                    Press enter to skip):")
removed_price = None
if remove_item:
    removed_price = prices.pop(remove_item, None) # dict.pop()
# Display results
print("\n Available items:", list(prices.key())) # dict.key()
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 (if existed):",
                                    removed_price)
```

output:-

```
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):b
Enter new price for box : 20
Enter an item to remove from price list (or pres
Enter to skip): pen
Available Items : ['box', 'pencil']
prices : [20.0, 50]
Costliest item : box at 20.0
Removed 'pen' price (if existed) : 10.0
```

4.4 - TechFest participation

Two events, AI hackathon and Robotics challenge, have participants' IDs stored in two 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 participate (union())

```
# Get AI Hackathon participant
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 participate
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 late registrant ID for AI Hackathon
                                (or press enter to skip):")
if late_id:
    ai_hackathon.add(late_id) # Set.add()
```

output:-

Enter number of participants in AI Hacthon: 4
Enter participant ID: A1
Enter participant ID: A2
Enter participant ID: A8
Enter participant ID: A5
Enter number of participants in Robotics challenge: 4
Enter participant ID: A1
Enter participant ID: A2
Enter participant ID: A3
Enter participant ID: A7
Enter late registrant ID for AI Hacthon (or press Enter to skip): A8
Enter withdraw participant ID from Robotics challenge. (press Enter to skip): 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

```
# Remove a withdrawn participant
remove_id = input("Enter withdraw 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_challenges)
```

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

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