Task No-4.1

Date :- 20/8/25

CAFETERIA SALES

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

Algorithm :-

-> start

-> create an empty list sales = []

-> for 7 days, append integer sales to the list using append.

-> compute total = sum(sales) and avg = total/7.

-> find max-val = max(sales), min-val = min(sales).

-> find corresponding days with index()

-> count days above a target using count() on a boolea remap or with a loop.

-> 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 sevendays
                                         sales count"))
    sales. append (sample-entries) # list. append()
total = sum (sales)
avg = total /days

max-val = max (sales)
min-val = min (sales)
```

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 : 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
```

```
best-day = sales.index(max-val)+1 # list.index()
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)
```

Task-4.2

TUPLE - LAB TIMETABLE

Aim :- To manage and query and immutable daily lab slot schedule using a tuple, demostrating membership checks, count(), index(), and slicing.

Algorithm :-

```
-> start
-> Define slots as a fixed tuple of integers.
-> Read query hour.
-> Check existence with query in slots.
-> use count(); it positive, use index () to find the first
   position.
-> slice into monitoring and afternoon
-> print resotts.
-> stop.
```

Program:

```
#1 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 occurence 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 time(s)
first occurence position (1- based):3
morning slots :(9,11)
Afternoon slots: (14,16,14)
```

Task NO - 4.3

DICTIONARY - BOOKSTORE BILLING

Aim :-

To manage a live price list and bill a customer using dictionary methods and views.

Algorithm :-

→ start
→ Create an empty dictionary prices.
→ Ask the user for the number of items in the price list (n1
> Repeat for each item:
> Get the item ~~pri~~name.
> Get the item price.
> Add the item and price to prices.
Ask the user for an item to update.
If the item exists in prices, get the new price and update it.

find the costliest item by checking each item's price.
ASK the user for an item to remove.
If given, remove that item from prices.
show all available items, their prices, the costliest item,
and the removed item's price.
stop.

ram :-

```
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
```

output :-

```
Enter the no.of iten in the List:3
Enter item name =box
enter price of box :15
Enter item name : pen
enter price of pen :10
enter iten nome =pencil
enter price of pencil :15 .
Enter item to update price 'box
enter new price for box :20
Enter an item from price list: pen
avariable items: ['box', 'pencil']
prices :[20.0,5.0]
Costliest item: boxat 20.0
Removed' pen' price :10.0
```

```
#optional price revision
rev-item = input ("Enter item toupdate price (or press
                                          enter toskip):
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
remove-item = input ("Enter an item to remove from
                        Price list (or press Enter toskip):
removed-price = None
 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.keys())) # dict.keys()
~~Print~~ if costliest_item:
       Print ("costliest item!", costliest_item, "at",
                                          max-price)
  if remove-item:
    Print (f"Removed"{remove-item}"price
                   (if existed):", removed-price)
```

VEL TECH

O.	4
FORMANCE (5)	5
ULT AND ANALYSIS (5)	5
VOCE (5)	5
ORD (5)	
AL (20)	15
WITH DATE	

esult: Thus, Various data types, list, Tuples and Dicti python programming was used and verified successfu