Date: 20/8/25 Use various data types, list, Tuples and Dictionary in Python Programming key Terms covered: Datatypes, List, Tuple, Set, Dict Tags -Easy

TASK 4:

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 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 = 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
```

output:

[10, 20, 30]

[10, 30]

[30]

[5, 8, 9, 15, 30, 89]

The maximum value is : 89

The minimum value is : 5

The average is : 26.0

The sum is : 156.0

```
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 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.

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 occurrence Position(1-based):", first_POS)
Print("Morning slots:", morning)

Print ("Afternoon slots:", afternoon)
```

Output

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

is 14 present? True

first occurs 2 time(s)

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 (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 costliest item, and the removed item's price
14. Stop.

Python Programming

```
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
```

Out put:-

```
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 the
  skip):Pen
Available items: ['box', 'pencil']
Prices: [20,0,50]
Costlies item: box at 20.0
Removed 'pen' price (if existed) :10.0
```

```
#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
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("\nAvailable items:", list(prices.keys())) # dict.keys()
print("Prices:", list(prices.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)
```

4.4 Set - 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 withdraw participant from Robotics using discard(), then find participants in both events (intersection()), only in one (difference()), the total unique participants (union()).

Aim:

Algorithm:

1. Start
2. Create two sets: AI_hackathon and Robotics_Challenge with participants IDs.
3. Add the late registration ID to AI_Hackathon using add().
4. Remove the withdraw participant ID from Robotics_challenge using discard().
5. Find participants in both events using intersection()
6. Find Participants in only AI hackathon using AI_Hackathon.difference(Robotics_challenge).
7. Find participants in only Robotics challenge using. Robotics_Challenge.difference(AI_Hackathon).
8. Find the total unique participants using union.
9. Display all results.
10. Stop

Python Program

```
# 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 participants 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()

# Remove a withdraw 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_challenge)
```

VEL TECH - [illegible]	
EX NO.	4
PERFORMANCE (5)	5
RESULT AND ANALYSIS (3)	3
VIVA VOCE (3)	3
RECORD (4)	4
TOTAL (15)	
SIGN WITH DATE	

Result: The programs using lists, tuples, dictionaries are successfully executed.

Task: 6 Date: 03/9/24

Utilizing 'functions' concepts in Python Programming

- You are developing a small python script to analyze and manipulate a list of student grades for a class project. write a python program that satisfies the above requirement using the bold-in function print(), len(), type(), max(), min(), sorted() reversed() and range()

Algorithm:

1. Start the program
2. Print a welcome program: outputs a simple greeting.
3. Determine and print the number of students: uses len() to find the number of elements in the student_names list
4. Print the type of lists: uses type() to show the type of the student_names and student_grade lists.
5. Find and print highest and lowest uses max() and min() to determine the highest and lowest values in student_grades.
6. Print sorted list of grades: uses sorted() to sort the grades.
7. Print reversed list of grades: user reversed() to reverse the sorted list and converts it to a list.
8. Generate and print a range of grades indices: use range() to create a list of indices from 1 to the number of students
9. Stop.

Program:

```
def analyze_student_grades():
# sample data
student_names = ["Alice", "BOB', 'Charlie', "Diana"]
student_grades = [85,92,78,90]
# print a welcome message
print ("welcome to the student Grades analyze !(n")
# 2. determine and print the number of students
num_students = len (student_name)
Print ("number of students":, num_students)
```

```
# 3. Print the type of the student names list and the grad
list.
Print("In Type of student_names list:", type(students_names))
Print(" Type of student_grades list", type(student_grades))
# 4 Find and print the highest and lowest grades.
highest_grade = max(student_grades)
lowest_grade = min(student_grades)
Print("In highest grade:", highest_grade)
print("lowest grade :", lowest_grade)
# 5. Print the list of grades sorted in ascending order
sorted_grades = sorted(student_grades)
Print("In sorted grades:", reversed_grades)
# 6. Print the list of grades in reverse order
reversed_grades = list(reversed(sorted_grades))
Print("Reversed grades", reversed_grades)
# 7. Generate and print range of grade indices from 1 to the
number of students.
grade_indices = list(range(1, num_student +1))
Print("In Grade indices from 1 to number of students:",
      grade_indices)

# Run the analysis
analyze_student_grades()
```

output:-

```
Welcome to the Student grades analyzer!
number of students: 4
type of student_names list: <class 'list'>
type of student_grades list: <class 'list'>
highest grade :92
lowest grade :78
Sorted grades :[78, 85, 90, 92]
Revessed grades :[92, 90, 85, 78]
grades indices, from 1 to number of students :[1, 2, 3, 4]
```

6.2 You are asked with creating a small calculator application to help users perform basic arithmetic operations and greet them with a personalized message. you application should perform the following task: addition, subtractior multiplication and division

Algorithm:-

1. Start the program
2. user input for numbers: The program prompts the user to enter two numbers.
3. user input for operation: The program prompts the user to choose an arithmatic operation (addition, subtraction, multiplicatior division)
4. Perform operation. Based on the user choice, the program performs the choose arithematic operation using the defined functions
5. Display result the program displays the result of the operation.
6. Stop.

Program:

```
def add(a,b):
""" Return the sum of two numbers"""
return a+b
def subtract (a,b):
""" Return the difference between two numbers"""
return a-b
def multiply (a,b):
""" Return the product of two numbers."""
return a*b
def divide(a,b):
""" Return the quotient of two numbers handles division by
    zero"""
if b!=0
   return a/b
else:
    return "Error : Division by zero"
def greet (name):
""" Return a greeting message for the user."""
```

output:

arithmetic operations:

Sum of 10 & 5 : 15

differences b/w 10 & 5 : 5

Product of 10 and 5 = 50

quotient of 10 and 5 = 20

greeting:

hello alice: welcome to the program.

```
return f" Hello, {name}! Welcome to the program".
def main()
# Demonstrating the use of user-defined functions.
# Arithmetic operations
num1 = 10
num2 = 5
print(" Arithmetic operations:")
print(f" Sum of {num1} and {num2}:", add(num1, num2))
print(f" Difference between {num1} and {num2}:", subtract(num1, num2))
print(f" Product of {num1} and {num2}:", multiply(num1, num2))
print(f" Quotient of {num1} and {num2}:", divide(num1, num2))
# Greeting the user
user_name = "Alice"
print("In Greeting:")
print(greet(user_name))
# Run the main function
if __name__ == "__main__":
main()
```

VEL TECH - CSE	
EX NO.	6
PERFORMANCE (5)	5
RESULT AND ANALYSIS (3)	3
VIVA VOCE (3)	3
RECORD (4)	4
TOTAL (15)	
SIGN WITH DATE	15

Result: Thus, the Python program using 'functions' concepts was successfully executed and the output was verified.