# Practical 2

**a) Implement following operation using python tuple concept.**

**Aim: Tuple operation**

Create tuples with different data types (integer, float, string, and mixed).
Access tuple elements using positive and negative indices.
Perform tuple slicing to extract specific portions of the tuple.
Count occurrences of an element and find the index of an element in a tuple.
Use built-in functions like len(), max(), min(), and sum() with tuples.
Write a program to count and print distinct elements from a tuple.
Convert a list to a tuple and vice versa.
Demonstrate unpacking of tuples into individual variables.

**Code:**

```
# 1. Create tuples with different data types
int_tuple = (1, 2, 3, 4, 5)
float_tuple = (1.1, 2.2, 3.3)
string_tuple = ("apple", "banana", "cherry")
mixed_tuple = (1, "apple", 3.14, True)

print("Integer Tuple:", int_tuple)
print("Float Tuple:", float_tuple)
print("String Tuple:", string_tuple)
print("Mixed Tuple:", mixed_tuple)

# 2. Access tuple elements using positive and negative indices
print("\nAccessing elements:")
print("Positive Index (int_tuple[1]):", int_tuple[1])  # 2
print("Negative Index (string_tuple[-1]):", string_tuple[-1])  #
"cherry"

# 3. Perform tuple slicing to extract specific portions
print("\nSlicing tuples:")
print("int_tuple[1:4]:", int_tuple[1:4])  # (2, 3, 4)
print("mixed_tuple[:3]:", mixed_tuple[:3])  # (1, "apple", 3.14)

# 4. Count occurrences and find the index of an element in a tuple
example_tuple = (1, 2, 3, 1, 1, 4)
print("\nCount and index:")
print("Count of 1 in example_tuple:", example_tuple.count(1))  # 3
print("Index of 3 in example_tuple:", example_tuple.index(3))  # 2

# 5. Use built-in functions with tuples
num_tuple = (10, 20, 30, 40)
```

```python
print("\nBuilt-in functions:")
print("Length of num_tuple:", len(num_tuple))  # 4
print("Maximum in num_tuple:", max(num_tuple))  # 40
print("Minimum in num_tuple:", min(num_tuple))  # 10
print("Sum of num_tuple:", sum(num_tuple))  # 100

# 6. Count and print distinct elements from a tuple
example_tuple = (1, 2, 3, 1, 4, 4, 5)
distinct_elements = set(example_tuple)
print("\nDistinct elements:")
print("Distinct elements in example_tuple:", distinct_elements)

# 7. Convert a list to a tuple and vice versa
example_list = [10, 20, 30]
converted_tuple = tuple(example_list)
converted_list = list(converted_tuple)
print("\nConversions:")
print("List to Tuple:", converted_tuple)  # (10, 20, 30)
print("Tuple to List:", converted_list)  # [10, 20, 30]

# 8. Demonstrate unpacking of tuples into individual variables
person_tuple = ("Alice", 25, "Engineer")
name, age, profession = person_tuple
print("\nTuple unpacking:")
print(f"Name: {name}, Age: {age}, Profession: {profession}")
```

**Output Screenshot:**

```
python3 -u "/Users/debdootmanna/VSCode/Python/2a.py"
compinit:527: no such file or directory: /usr/local/share/zsh/site-functions/_brew_cask
compinit:527: no such file or directory: /usr/local/share/zsh/site-functions/_brew_cask
  ~/VSCode/Python | main ?1                                                    ✔ | 02:18:51 PM
 python3 -u "/Users/debdootmanna/VSCode/Python/2a.py"
Integer Tuple: (1, 2, 3, 4, 5)
Float Tuple: (1.1, 2.2, 3.3)
String Tuple: ('apple', 'banana', 'cherry')
Mixed Tuple: (1, 'apple', 3.14, True)

Accessing elements:
Positive Index (int_tuple[1]): 2
Negative Index (string_tuple[-1]): cherry

Slicing tuples:
int_tuple[1:4]: (2, 3, 4)
mixed_tuple[:3]: (1, 'apple', 3.14)

Count and index:
Count of 1 in example_tuple: 3
Index of 3 in example_tuple: 2

Built-in functions:
Length of num_tuple: 4
Maximum in num_tuple: 40
Minimum in num_tuple: 10
Sum of num_tuple: 100

Distinct elements:
Distinct elements in example_tuple: {1, 2, 3, 4, 5}

Conversions:
List to Tuple: (10, 20, 30)
Tuple to List: [10, 20, 30]

Tuple unpacking:
Name: Alice, Age: 25, Profession: Engineer

  ~/VSCode/Python | main ?1
```

**b) Implement following operation using Python List concept.**

**Aim: List Operation**

Create a list of integers, strings, and mixed data types.
Access elements using indices, perform slicing, and update list elements.
Add and remove elements using append(), insert(), remove(), and pop() methods.
Concatenate and repeat lists using operators.
Create a list of squares of the first 10 natural numbers using list comprehension.
Filter even numbers from a list using list comprehension.
Demonstrate sorting, reversing, and copying lists.
Write a program to remove duplicates from a list.

**Code:**

```python
# 1. Create a list of integers, strings, and mixed data types
int_list = [1, 2, 3, 4, 5]
string_list = ["apple", "banana", "cherry"]
mixed_list = [1, "apple", 3.14, True]

print("Integer List:", int_list)
print("String List:", string_list)
print("Mixed List:", mixed_list)

# 2. Access elements using indices, perform slicing, and update list
elements
print("\nAccessing and updating:")
print("int_list[1]:", int_list[1])  # Access element at index 1
print("string_list[-1]:", string_list[-1])  # Access last element
```

```python
# Slicing
print("int_list[1:4]:", int_list[1:4])  # Slice elements

# Updating elements
mixed_list[1] = "orange"
print("Updated mixed_list:", mixed_list)

# 3. Add and remove elements using append(), insert(), remove(), and
pop()
print("\nAdding and removing elements:")
int_list.append(6)
print("After append:", int_list)

int_list.insert(2, 10)
print("After insert at index 2:", int_list)

int_list.remove(10)  # Removes the first occurrence of 10
print("After remove(10):", int_list)

popped_element = int_list.pop()  # Pops the last element
print("After pop(), popped element:", popped_element)
print("After pop:", int_list)

# 4. Concatenate and repeat lists using operators
concat_list = int_list + string_list
print("\nConcatenated List:", concat_list)

repeated_list = string_list * 2
print("Repeated List:", repeated_list)

# 5. Create a list of squares of the first 10 natural numbers using
list comprehension
squares = [x**2 for x in range(1, 11)]
print("\nSquares of the first 10 natural numbers:", squares)

# 6. Filter even numbers from a list using list comprehension
numbers = list(range(1, 21))
even_numbers = [x for x in numbers if x % 2 == 0]
print("Even numbers:", even_numbers)

# 7. Demonstrate sorting, reversing, and copying lists
print("\nSorting, reversing, and copying:")
unsorted_list = [5, 2, 9, 1, 5, 6]
```

```
sorted_list = sorted(unsorted_list)  # Sorting without modifying
original
print("Sorted List:", sorted_list)

unsorted_list.sort()  # Sorting and modifying the original
print("After sort():", unsorted_list)

unsorted_list.reverse()  # Reverse the list
print("After reverse():", unsorted_list)

copied_list = unsorted_list.copy()  # Copy the list
print("Copied List:", copied_list)

# 8. Write a program to remove duplicates from a list
duplicate_list = [1, 2, 2, 3, 4, 4, 5, 5, 6]
unique_list = list(set(duplicate_list))  # Remove duplicates using set
print("\nList with duplicates removed:", unique_list)

# Optional: Maintain order while removing duplicates
unique_list_ordered = []
for item in duplicate_list:
    if item not in unique_list_ordered:
        unique_list_ordered.append(item)
print("List with duplicates removed (order maintained):",
unique_list_ordered)
```

**Output Screenshot:**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  COMMENTS  SQL HISTORY  TASK MONITOR

~/VSCode/Python | main ?1
python3 -u "/Users/debdootmanna/VSCode/Python/2b.py"
Integer List: [1, 2, 3, 4, 5]
String List: ['apple', 'banana', 'cherry']
Mixed List: [1, 'apple', 3.14, True]

Accessing and updating:
int_list[1]: 2
string_list[-1]: cherry
int_list[1:4]: [2, 3, 4]
Updated mixed_list: [1, 'orange', 3.14, True]

Adding and removing elements:
After append: [1, 2, 3, 4, 5, 6]
After insert at index 2: [1, 2, 10, 3, 4, 5, 6]
After remove(10): [1, 2, 3, 4, 5, 6]
After pop(), popped element: 6
After pop: [1, 2, 3, 4, 5]

Concatenated List: [1, 2, 3, 4, 5, 'apple', 'banana', 'cherry']
Repeated List: ['apple', 'banana', 'cherry', 'apple', 'banana', 'cherry']

Squares of the first 10 natural numbers: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
Even numbers: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

Sorting, reversing, and copying:
Sorted List: [1, 2, 5, 5, 6, 9]
After sort(): [1, 2, 5, 5, 6, 9]
After reverse(): [9, 6, 5, 5, 2, 1]
Copied List: [9, 6, 5, 5, 2, 1]

List with duplicates removed: [1, 2, 3, 4, 5, 6]
List with duplicates removed (order maintained): [1, 2, 3, 4, 5, 6]

~/VSCode/Python | main ?2
```

**c) Implementing following operation using python dictionaries concept.**

**Aim: Dictionary Operation**

Create a dictionary to store key-value pairs.
Access, update, and delete dictionary elements using keys.
Use dictionary methods like keys(), values(), and items().
Add a new key-value pair and remove an existing key-value pair.
Create a nested dictionary to store student details (like name, age, and marks).
Access and update elements in a nested dictionary.
Merge two dictionaries using update().
Write a program to sort a dictionary based on its values.

**Code:**

```
# 1. Create a dictionary to store key-value pairs
my_dict = {"name": "Alice", "age": 25, "profession": "Engineer"}
print("Dictionary:", my_dict)

# 2. Access, update, and delete dictionary elements using keys
print("\nAccessing elements:")
print("Name:", my_dict["name"])  # Access element by key

# Updating a value
my_dict["age"] = 26
print("Updated Dictionary:", my_dict)

# Deleting an element
del my_dict["profession"]
print("After deletion:", my_dict)

# 3. Use dictionary methods like keys(), values(), and items()
print("\nDictionary methods:")
print("Keys:", my_dict.keys())  # Returns all keys
print("Values:", my_dict.values())  # Returns all values
print("Items:", my_dict.items())  # Returns all key-value pairs as
tuples

# 4. Add a new key-value pair and remove an existing key-value pair
my_dict["city"] = "New York"  # Add a new key-value pair
print("\nAfter adding a new key-value pair:", my_dict)

removed_value = my_dict.pop("city")  # Remove a key-value pair
print("After removing 'city':", my_dict)
print("Removed Value:", removed_value)

# 5. Create a nested dictionary to store student details
students = {
    "student1": {"name": "John", "age": 20, "marks": {"math": 85,
"science": 90}},
    "student2": {"name": "Emily", "age": 22, "marks": {"math": 78,
"science": 88}},
}
print("\nNested Dictionary (Students):", students)

# 6. Access and update elements in a nested dictionary
print("\nAccessing nested dictionary elements:")
```

```python
print("Student1's Math Marks:", students["student1"]["marks"]["math"])
# Access nested element

# Updating a nested value
students["student2"]["marks"]["math"] = 80
print("Updated Nested Dictionary:", students)

# 7. Merge two dictionaries using update()
dict1 = {"a": 1, "b": 2}
dict2 = {"b": 3, "c": 4}
dict1.update(dict2)  # Merge dict2 into dict1, overwriting common keys
print("\nMerged Dictionary:", dict1)

# 8. Write a program to sort a dictionary based on its values
unsorted_dict = {"apple": 3, "banana": 1, "cherry": 2}
sorted_dict = dict(sorted(unsorted_dict.items(), key=lambda item:
item[1]))
print("\nSorted Dictionary (by values):", sorted_dict)
```

**Output Screenshot:**


**Capstone Project1: College Event Management System**

**Objective:** Apply tuple, list, and dictionary concepts to manage participants and event details.

**Task:**

Store event information as a dictionary where the event name is the key and the value is a list of participant tuples.
Each tuple contains (Participant Name, Contact Number, Department, Participation Status).
Write a program to:
Display the list of participants for a specific event.
Search for a participant by name and display their event details.
Mark a participant as “Attended” or “Not Attended”.
Generate a summary of total participants in each event.

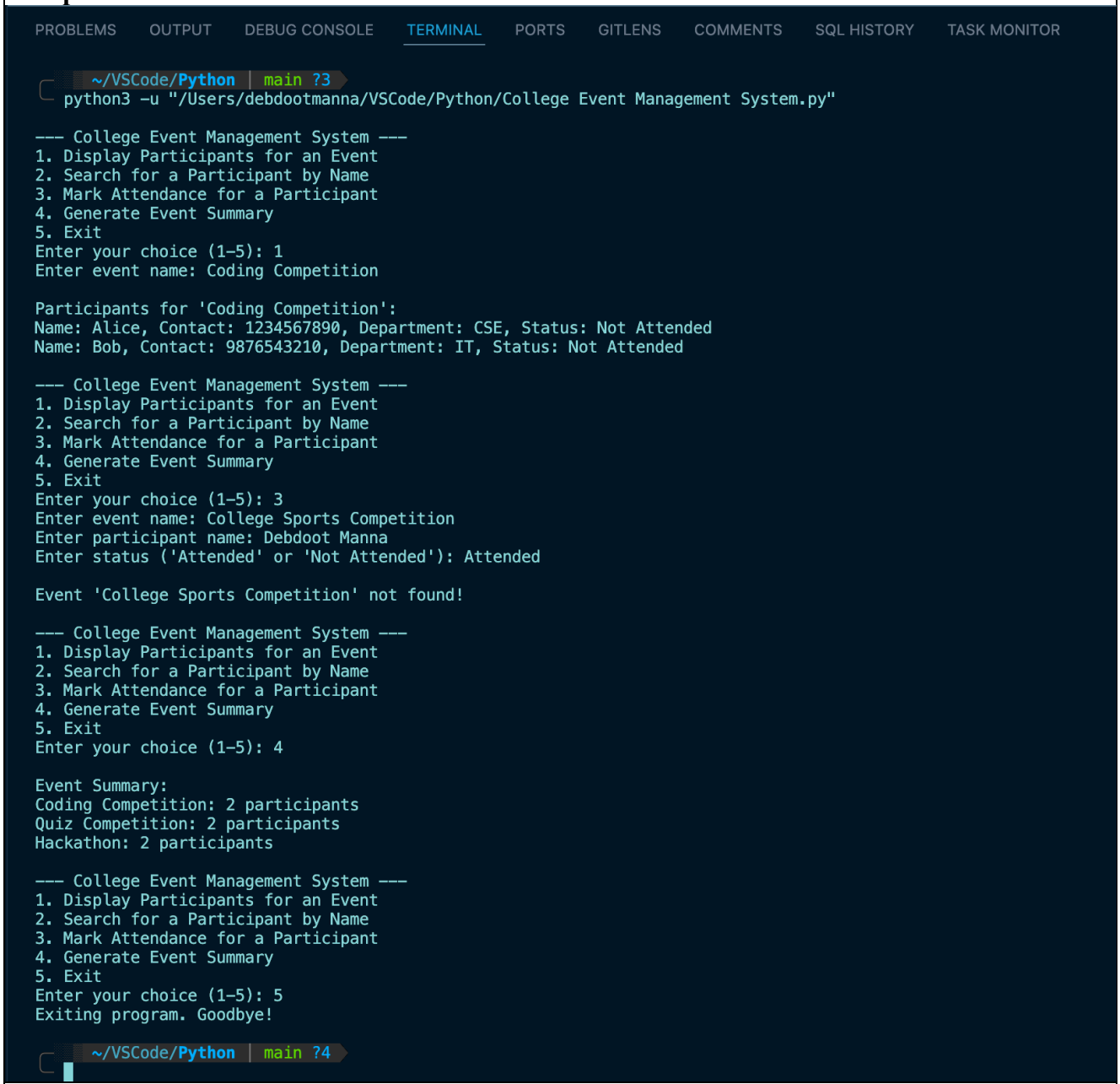
**Code:**

```
# Sample data for events and participants
events = {
    "Coding Competition": [
        ("Alice", "1234567890", "CSE", "Not Attended"),
        ("Bob", "9876543210", "IT", "Not Attended"),
    ],
    "Quiz Competition": [
        ("Charlie", "1231231234", "ECE", "Not Attended"),
        ("Dave", "3213214321", "ME", "Not Attended"),
    ],
    "Hackathon": [
        ("Eve", "1112223334", "CSE", "Not Attended"),
        ("Frank", "5556667778", "IT", "Not Attended"),
    ],
}

# Function to display the list of participants for a specific event
def display_participants(event_name):
    if event_name in events:
        print(f"\nParticipants for '{event_name}':")
        for participant in events[event_name]:
            print(f"Name: {participant[0]}, Contact: {participant[1]},
Department: {participant[2]}, Status: {participant[3]}")
    else:
        print(f"\nEvent '{event_name}' not found!")

# Function to search for a participant by name and display their event
details
def search_participant(participant_name):
    found = False
    for event_name, participants in events.items():
        for participant in participants:
            if participant[0].lower() == participant_name.lower():
                print(f"\nParticipant Found: {participant_name}")
                print(f"Event: {event_name}, Contact: {participant[1]},
Department: {participant[2]}, Status: {participant[3]}")
                found = True
    if not found:
        print(f"\nParticipant '{participant_name}' not found!")

# Function to mark a participant as "Attended" or "Not Attended"
def mark_attendance(event_name, participant_name, status):
```

```
    if event_name in events:
        participants = events[event_name]
        for i, participant in enumerate(participants):
            if participant[0].lower() == participant_name.lower():
                participants[i] = (participant[0], participant[1],
participant[2], status)
                print(f"\nUpdated Status for {participant_name} in
'{event_name}' to '{status}'.")
                return
        print(f"\nParticipant '{participant_name}' not found in
'{event_name}'.")
    else:
        print(f"\nEvent '{event_name}' not found!")

# Function to generate a summary of total participants in each event
def generate_summary():
    print("\nEvent Summary:")
    for event_name, participants in events.items():
        print(f"{event_name}: {len(participants)} participants")

# Menu-driven program
while True:
    print("\n--- College Event Management System ---")
    print("1. Display Participants for an Event")
    print("2. Search for a Participant by Name")
    print("3. Mark Attendance for a Participant")
    print("4. Generate Event Summary")
    print("5. Exit")

    choice = input("Enter your choice (1-5): ")

    if choice == "1":
        event_name = input("Enter event name: ")
        display_participants(event_name)
    elif choice == "2":
        participant_name = input("Enter participant name: ")
        search_participant(participant_name)
    elif choice == "3":
        event_name = input("Enter event name: ")
        participant_name = input("Enter participant name: ")
        status = input("Enter status ('Attended' or 'Not Attended'): ")
        mark_attendance(event_name, participant_name, status)
    elif choice == "4":
```

```
        generate_summary()
    elif choice == "5":
        print("Exiting program. Goodbye!")
        break
    else:
        print("Invalid choice. Please try again!")
```

**Output Screenshot:**




**Capstone project2: Online Food Delivery System**

**Objective:** Design an online food delivery system using list, tuple, dictionary, and set concepts.

**Task:**

Store menu items as a dictionary where item names are keys and (Price, Category) is the value.
Create a list of orders where each order is a tuple (Order ID, Customer Name, Item List, Total Bill).
Use a set to store unique customer names who have placed orders.
Write a program to:
Allow users to place an order by selecting items from the menu.

Generate a bill for the customer and store it in the list of orders.
Display the total revenue generated from all orders.
Display the list of unique customers who have placed orders.

**Code:**

```
# 1. Store menu items as a dictionary where item names are keys and
(Price, Category) is the value.
menu = {
    "Burger": (120, "Fast Food"),
    "Pizza": (250, "Fast Food"),
    "Pasta": (200, "Italian"),
    "Salad": (100, "Healthy"),
    "Ice Cream": (80, "Dessert"),
    "Coffee": (60, "Beverage"),
}

# 2. Create a list of orders where each order is a tuple (Order ID,
Customer Name, Item List, Total Bill).
orders = []

# 3. Use a set to store unique customer names who have placed orders.
unique_customers = set()

# Function to display the menu
def display_menu():
    print("\n--- Menu ---")
    print("{:<20} {:<10} {:<10}".format("Item", "Price", "Category"))
    for item, (price, category) in menu.items():
        print(f"{item:<20} {price:<10} {category:<10}")

# Function to place an order
def place_order(order_id):
    customer_name = input("\nEnter your name: ")
    unique_customers.add(customer_name)  # Add customer to the unique
customers set

    item_list = []
    total_bill = 0

    while True:
        display_menu()
        item_name = input("\nEnter item name to add to your order (or
'done' to finish): ").strip()
```

```
        if item_name.lower() == "done":
            break

        if item_name in menu:
            item_list.append(item_name)
            total_bill += menu[item_name][0]
            print(f"Added '{item_name}' to your order. Current Total:
{total_bill}")
        else:
            print("Item not found in the menu. Please try again.")

    if item_list:
        # Add the order to the list of orders
        orders.append((order_id, customer_name, item_list, total_bill))
        print(f"\nOrder placed successfully! Your Total Bill:
{total_bill}")
    else:
        print("No items selected. Order not placed.")

# Function to display total revenue generated from all orders
def display_total_revenue():
    total_revenue = sum(order[3] for order in orders)  # Sum up the
total bill of all orders
    print(f"\nTotal Revenue Generated: {total_revenue}")

# Function to display the list of unique customers who have placed
orders
def display_unique_customers():
    print("\nUnique Customers:")
    for customer in unique_customers:
        print(customer)

# Menu-driven program
order_id = 1
while True:
    print("\n--- Online Food Delivery System ---")
    print("1. Place an Order")
    print("2. Display Total Revenue")
    print("3. Display Unique Customers")
    print("4. Exit")

    choice = input("Enter your choice (1-4): ")
```

```
    if choice == "1":
        place_order(order_id)
        order_id += 1
    elif choice == "2":
        display_total_revenue()
    elif choice == "3":
        display_unique_customers()
    elif choice == "4":
        print("Exiting the system. Goodbye!")
        break
    else:
        print("Invalid choice. Please try again.")
```

**Output Screenshot:**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  COMMENTS  SQL HISTORY  TASK MONITOR

~/VSCode/Python | main ?4
python3 -u "/Users/debdootmanna/VSCode/Python/Online Food Delivery System.py"

--- Online Food Delivery System ---
1. Place an Order
2. Display Total Revenue
3. Display Unique Customers
4. Exit
Enter your choice (1-4): 1

Enter your name: Debdoot

--- Menu ---
Item                Price      Category
Burger              120        Fast Food
Pizza               250        Fast Food
Pasta               200        Italian
Salad               100        Healthy
Ice Cream           80         Dessert
Coffee              60         Beverage

Enter item name to add to your order (or 'done' to finish): Pizza
Added 'Pizza' to your order. Current Total: 250

--- Menu ---
Item                Price      Category
Burger              120        Fast Food
Pizza               250        Fast Food
Pasta               200        Italian
Salad               100        Healthy
Ice Cream           80         Dessert
Coffee              60         Beverage

Enter item name to add to your order (or 'done' to finish): Salad
Added 'Salad' to your order. Current Total: 350

--- Menu ---
Item                Price      Category
Burger              120        Fast Food
Pizza               250        Fast Food
Pasta               200        Italian
Salad               100        Healthy
Ice Cream           80         Dessert
Coffee              60         Beverage

Enter item name to add to your order (or 'done' to finish): done

Order placed successfully! Your Total Bill: 350

--- Online Food Delivery System ---
1. Place an Order
2. Display Total Revenue
3. Display Unique Customers
4. Exit
Enter your choice (1-4): 2

Total Revenue Generated: 350

--- Online Food Delivery System ---
1. Place an Order
2. Display Total Revenue
3. Display Unique Customers
4. Exit
Enter your choice (1-4): 3
```

```
Unique Customers:
Debdoot

--- Online Food Delivery System ---
1. Place an Order
2. Display Total Revenue
3. Display Unique Customers
4. Exit
Enter your choice (1-4): 4
Exiting the system. Goodbye!

~/VSCode/Python | main ?5
```

**Conclusion/Summary:**

The practical implemented—covering Python concepts like tuples, lists, dictionaries, and sets—successfully demonstrated their versatility in solving real-world problems. Here's a summary of each project:

1. Tuple Operations:
   - Showcased tuple creation, indexing, slicing, and usage of built-in functions (e.g., len(), max(), min(), sum()).
   - Demonstrated distinct element counting, tuple unpacking, and conversion between tuples and lists.
2. List Operations:
   - Included list creation, slicing, updating, and manipulation using methods like append(), insert(), remove(), and pop().
   - Covered list comprehensions for generating squares and filtering even numbers, along with sorting, reversing, and duplicate removal.
3. Dictionary Operations:
   - Demonstrated key-value pair creation, access, and updates.
   - Used dictionary methods (keys(), values(), items()) and implemented sorting by values.
   - Explored nested dictionaries for managing structured data like student details and dictionary merging.
4. Capstone Project 1: College Event Management System:
   - Utilized dictionaries to manage events, tuples for participant data, and lists for event participants.
   - Implemented features like searching, attendance marking, and participant summary generation.
5. Capstone Project 2: Online Food Delivery System:
   - Used dictionaries for menus, lists for orders, and sets for unique customers.
   - Built a system for placing orders, generating bills, calculating total revenue, and identifying unique customers.

Key Takeaways:

- Tuples ensure data integrity with immutability.
- Lists provide flexibility for dynamic data manipulation.
- Dictionaries enable fast key-based access for structured data.
- Sets ensure uniqueness in collections.

| | | |
|---|---|---|
| These projects emphasized the importance of Python's data structures in real-world applications like event management and e-commerce systems. They also provide a strong foundation for more advanced software development, including database integration and analytics. | | |
| **Student Signature & Date** | **Marks:** | **Evaluator Signature & Date** |