Practical 2

|  |  |  |
| --- | --- | --- |
| 1. **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)  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:** | | |
| **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:**  *# 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*  *# 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:** | | |
| **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:")  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:** | | |
|  | | |
| **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** |