Assignment 2

|  |  |  |
| --- | --- | --- |
| **Use tuples in practical data handling scenarios.** | | |
| **Aim:** Write a function that takes a tuple as an argument and returns the tuple with all duplicates removed.  Create a list of tuples representing student names and marks, and sort the list by marks.  Write a program to count the frequency of elements in a tuple using Counter from the collections module.  Implement a tuple-based record system where each tuple represents a record (ID, Name, and Marks) and perform search operations | | |
| **Code:**  *from* collections *import* Counter  *# Function to remove duplicates from a tuple*  ***def*** remove\_duplicates(input\_tuple):  *return* tuple(set(input\_tuple))  *# Function to create a list of tuples representing students and sort by marks*  ***def*** sort\_students\_by\_marks(student\_data):  *return* sorted(student\_data, key**=*lambda*** x: x[1], reverse**=**True)  *# Function to count the frequency of elements in a tuple using Counter*  ***def*** count\_frequency(input\_tuple):  *return* *Counter*(input\_tuple)  *# Function to implement a tuple-based record system and search operations*  ***def*** search\_record(records, search\_id**=**None, search\_name**=**None):  *if* search\_id:  result **=** [record *for* record *in* records *if* record[0] **==** search\_id]  *elif* search\_name:  result **=** [record *for* record *in* records *if* record[1].*lower*() **==** search\_name.*lower*()]  *else*:  result **=** []  *return* result  *# Main program to demonstrate the functionality*  *if* \_\_name\_\_ **==** "\_\_main\_\_":  *# 1. Removing duplicates from a tuple*  input\_tuple **=** (1, 2, 3, 2, 1, 4, 5, 3, 6)  print("Original Tuple:", input\_tuple)  print("Tuple after removing duplicates:", *remove\_duplicates*(input\_tuple))    *# 2. List of tuples representing student names and marks*  students **=** [("Alice", 85), ("Bob", 90), ("Charlie", 75), ("David", 90), ("Eva", 95)]  print("\nOriginal Student Data:", students)  sorted\_students **=** *sort\_students\_by\_marks*(students)  print("Sorted Student Data by Marks (Descending):", sorted\_students)    *# 3. Counting the frequency of elements in a tuple*  frequency\_tuple **=** (1, 2, 3, 2, 1, 2, 4, 3, 1)  print("\nTuple for Frequency Count:", frequency\_tuple)  print("Frequency Count:", *count\_frequency*(frequency\_tuple))    *# 4. Tuple-based record system*  records **=** [  (101, "Alice", 85),  (102, "Bob", 90),  (103, "Charlie", 75),  (104, "David", 90),  (105, "Eva", 95)  ]  print("\nRecords:", records)  *# Search by ID*  search\_id **=** 102  print(***f***"Search Result for ID {search\_id}:", *search\_record*(records, search\_id**=**search\_id))  *# Search by Name*  search\_name **=** "Eva"  print(***f***"Search Result for Name '{search\_name}':", *search\_record*(records, search\_name**=**search\_name))  **Output Screenshot:** | | |
| **Apply list concepts to real-world scenarios.** | | |
| **Aim:** Implement a program to generate a list of prime numbers within a given range.  Flatten a nested list using recursion.  Write a program to find the second largest element from a list without using built-in functions.  Use a list to manage a task queue, where tasks are added, removed, and processed sequentially. | | |
| **Code:**  *# Function to generate a list of prime numbers within a given range*  ***def*** generate\_primes(start, end):  ***def*** is\_prime(n):  *if* n **<** 2:  *return* False  *for* i *in* range(2, int(n**\*\***0.5) **+** 1):  *if* n **%** i **==** 0:  *return* False  *return* True    *return* [num *for* num *in* range(start, end **+** 1) *if* *is\_prime*(num)]  *# Function to flatten a nested list using recursion*  ***def*** flatten\_nested\_list(nested\_list):  flat\_list **=** []  *for* element *in* nested\_list:  *if* isinstance(element, list):  flat\_list.*extend*(*flatten\_nested\_list*(element))  *else*:  flat\_list.*append*(element)  *return* flat\_list  *# Function to find the second largest element from a list without using built-in functions*  ***def*** find\_second\_largest(numbers):  *if* len(numbers) **<** 2:  *return* None *# Not enough elements for second largest*  largest **=** second\_largest **=** float('-inf')  *for* num *in* numbers:  *if* num **>** largest:  second\_largest, largest **=** largest, num  *elif* num **>** second\_largest **and** num **!=** largest:  second\_largest **=** num  *return* second\_largest *if* second\_largest **!=** float('-inf') *else* None  *# Task queue management using a list*  ***class*** TaskQueue:  ***def*** \_\_init\_\_(self):  *self*.queue **=** []  ***def*** add\_task(self, task):  *self*.queue.*append*(task)  print(***f***"Task '{task}' added to the queue.")  ***def*** remove\_task(self):  *if* *self*.queue:  task **=** *self*.queue.*pop*(0)  print(***f***"Task '{task}' removed from the queue.")  *return* task  *else*:  print("No tasks in the queue to remove.")  *return* None  ***def*** process\_tasks(self):  print("\nProcessing tasks:")  *while* *self*.queue:  task **=** *self*.*remove\_task*()  print(***f***"Processing task: {task}")  *# Main program to demonstrate the functionality*  *if* \_\_name\_\_ **==** "\_\_main\_\_":  *# 1. Generate a list of prime numbers within a range*  start, end **=** 10, 50  print("Prime numbers between", start, "and", end, ":", *generate\_primes*(start, end))    *# 2. Flatten a nested list using recursion*  nested\_list **=** [1, [2, [3, 4], 5], [6, 7], 8]  print("\nOriginal Nested List:", nested\_list)  print("Flattened List:", *flatten\_nested\_list*(nested\_list))    *# 3. Find the second largest element in a list*  numbers **=** [10, 20, 4, 45, 99, 45, 50]  print("\nList of Numbers:", numbers)  print("Second Largest Element:", *find\_second\_largest*(numbers))    *# 4. Task queue management using a list*  print("\nTask Queue Management:")  task\_queue **=** *TaskQueue*()  task\_queue.*add\_task*("Task 1")  task\_queue.*add\_task*("Task 2")  task\_queue.*add\_task*("Task 3")  task\_queue.*process\_tasks*()  **Output Screenshot:** | | |
| **Apply dictionary concepts to real-world scenarios.** | | |
| **Aim:** Write a program to count the frequency of each word in a string and store it in a dictionary.  Implement a simple phonebook application using a dictionary where users can add, delete, and search for contacts.  Create a dictionary of students and their grades. Write a program to filter students who scored more than a specific mark.  Write a program to convert a list of tuples (key-value pairs) into a dictionary and vice versa. | | |
| **Code:**  *# 1. Count the frequency of each word in a string and store it in a dictionary*  ***def*** count\_word\_frequency(input\_string):  words **=** input\_string.*split*()  frequency **=** {}  *for* word *in* words:  word **=** word.*lower*().*strip*(",.!?") *# Normalize words (lowercase and remove punctuation)*  frequency[word] **=** frequency.*get*(word, 0) **+** 1  *return* frequency  *# 2. Simple phonebook application*  ***class*** Phonebook:  ***def*** \_\_init\_\_(self):  *self*.contacts **=** {}  ***def*** add\_contact(self, name, number):  *self*.contacts[name] **=** number  print(***f***"Contact '{name}' added with number {number}.")  ***def*** delete\_contact(self, name):  *if* name **in** *self*.contacts:  *del* *self*.contacts[name]  print(***f***"Contact '{name}' deleted.")  *else*:  print(***f***"Contact '{name}' not found.")  ***def*** search\_contact(self, name):  *return* *self*.contacts.*get*(name, "Contact not found.")  *# 3. Filter students who scored more than a specific mark*  ***def*** filter\_students\_by\_grade(students, min\_marks):  *return* {name: grade *for* name, grade *in* students.*items*() *if* grade **>** min\_marks}  *# 4. Convert list of tuples (key-value pairs) into a dictionary and vice versa*  ***def*** list\_to\_dict(tuple\_list):  *return* dict(tuple\_list)  ***def*** dict\_to\_list(dictionary):  *return* list(dictionary.*items*())  *# Main program to demonstrate functionality*  *if* \_\_name\_\_ **==** "\_\_main\_\_":  *# 1. Word frequency count*  input\_string **=** "Hello, world! Hello everyone. Welcome to the world of Python."  print("Input String:", input\_string)  word\_freq **=** *count\_word\_frequency*(input\_string)  print("\nWord Frequency:", word\_freq)  *# 2. Phonebook application*  print("\nPhonebook Application:")  phonebook **=** *Phonebook*()  phonebook.*add\_contact*("Alice", "123-456-7890")  phonebook.*add\_contact*("Bob", "987-654-3210")  print("Search for 'Alice':", phonebook.*search\_contact*("Alice"))  phonebook.*delete\_contact*("Bob")  print("Search for 'Bob':", phonebook.*search\_contact*("Bob"))  *# 3. Filter students by grades*  students **=** {"Alice": 85, "Bob": 92, "Charlie": 78, "David": 88, "Eva": 95}  min\_marks **=** 80  print("\nOriginal Student Grades:", students)  filtered\_students **=** *filter\_students\_by\_grade*(students, min\_marks)  print(***f***"Students scoring more than {min\_marks}:", filtered\_students)  *# 4. Convert between dictionary and list of tuples*  tuple\_list **=** [("name", "Alice"), ("age", 25), ("city", "New York")]  print("\nList of Tuples:", tuple\_list)  converted\_dict **=** *list\_to\_dict*(tuple\_list)  print("Converted Dictionary:", converted\_dict)  converted\_list **=** *dict\_to\_list*(converted\_dict)  print("Converted Back to List of Tuples:", converted\_list)  **Output Screenshot:** | | |
| **Conclusion/Summary:**  In this assignment, we explored the versatile applications of Python dictionaries in solving real-world problems. By using dictionaries, we efficiently counted word frequencies in a string, implemented a simple and interactive phonebook application, filtered student records based on criteria, and converted data between dictionaries and list of tuples. These tasks demonstrated the power of dictionaries in organizing, retrieving, and manipulating data efficiently. This exercise highlights the importance of choosing the right data structure to simplify complex operations in Python. | | |
| **Student Signature & Date** | **Marks:** | **Evaluator Signature & Date** |