Lab Report 6 Date:2081/03/28

Experiment 1: Data Structure

Title: Programming to learn about python data structures.

Objective:The objective of this lab-work is to efficiently store, organize, and manipulate data in a way that allows optimal performance for various operations, such as searching, insertion, deletion and updating.

Theory: In Python, data structures are built in or can be implemented using classes and libraries. Python provides several built in data structures that are easy to use and efficient for common tasks. Some common data structure in python are: List, Tuple, Set, Dictionary, Queue,Stack.

1. WAP to implement stack using list.

*class Stack:*

*def \_\_init\_\_(self):*

*self.stack=[]*

*def push(self,item):*

*self.stack.append(item)*

*def pop(self):*

*if not self.is\_empty():*

*return self.stack.pop()*

*else:*

*return "Stack is Empty"*

*def peek(self):*

*if not self.is\_empty():*

*return self.stack[-1]*

*else:*

*return "Stack is empty"*

*def is\_empty(self):*

*return len(self.stack)==0*

*def size(self):*

*return len(self.stack)*

*stack=Stack()*

*stack.push(10)*

*stack.push(13)*

*stack.push(14)*

*print("Top item: ",stack.peek())*

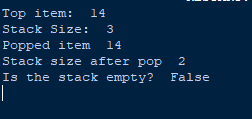
*print("Stack Size: ",stack.size())*

*print("Popped item ",stack.pop())*

*print("Stack size after pop ",stack.size())*

*print("Is the stack empty? ",stack.is\_empty())*

Output:



1. Create a program to implement queue.

*class Queue:*

*def \_\_init\_\_(self):*

*self.queue = []*

*def enqueue(self, item):*

*self.queue.append(item)*

*def dequeue(self):*

*if not self.is\_empty():*

*return self.queue.pop(0)*

*else:*

*return "Queue is empty!"*

*def peek(self):*

*if not self.is\_empty():*

*return self.queue[0]*

*else:*

*return "Queue is empty!"*

*def is\_empty(self):*

*return len(self.queue) == 0*

*def size(self):*

*return len(self.queue)*

*queue = Queue()*

*queue.enqueue(10)*

*queue.enqueue(20)*

*queue.enqueue(30)*

*print("Front item:", queue.peek())*

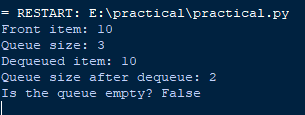
*print("Queue size:", queue.size())*

*print("Dequeued item:", queue.dequeue())*

*print("Queue size after dequeue:", queue.size())*

*print("Is the queue empty?", queue.is\_empty())*

Output:



1. Write a program to find duplicates in list.

*def find\_duplicates(input\_list):*

*duplicates = []*

*for i in range(len(input\_list)):*

*if input\_list[i] in input\_list[:i]:*

*duplicates.append(input\_list[i])*

*return duplicates*

*input\_list = [1, 2, 3, 4, 2, 5, 3, 6, 7, 8, 6]*

*duplicates = find\_duplicates(input\_list)*

*print("The original list is ",input\_list)*

*print("Duplicates in the list:", duplicates)*

Output:



1. Implement a Python program to sort a dictionary by the values.

*my\_dict = {'a': 10, 'b': 30, 'c': 20, 'd': 40}*

*sorted\_dict = dict(sorted(my\_dict.items(), key=lambda item: item[1]))*

*print("The original dictionary ",my\_dict)*

*print("Dictionary sorted by values (ascending):", sorted\_dict)*

Output:



1. WAP to count the occurrence of each element in tuple.

*def count\_occurrences(tup):*

*count\_dict = {}*

*for item in tup:*

*if item in count\_dict:*

*count\_dict[item] += 1*

*else:*

*count\_dict[item] = 1*

*return count\_dict*

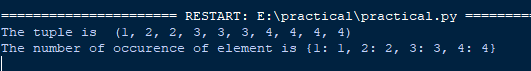
*my\_tuple = (1, 2, 2, 3, 3, 3, 4, 4, 4, 4)*

*print("The tuple is ",my\_tuple)*

*result = count\_occurrences(my\_tuple)*

*print("The number of occurence of element is",result)*

Output:



Conclusion: In the above page we have done the Data Structure programming with its output .