# S. Y. B. Tech (ECE)

**Trimester: VI Subject: Linux Based Python Laboratory (CET2005A)**

# Name: Shreerang Mhatre Class: Electrical and Computer

**Roll No.: 29 Batch: A2**

# Experiment – 05

**Title: Introduction to Advanced Data Structures of Python (Any two)**

# Performed on: 02/11/2022

**Marks**

**Teacher’s Signature with date**

**Submitted on: 02/11/2022**

**Aim**: Introduction to Advanced Data Structures of Python (Any two)

# Objective:

* 1. To know the Advanced Data Structures of Python.
  2. To implement at least two Advanced Data Structures of Python programs.

# Theory:

* **Advanced Data Structures in Python:**

# Tuple

* **Functions**

# List and set

* **Sorting**

# Dictionary Tuple in python

**A tuple is another sequence data type that is similar to the list. The main differences between lists and tuples are:**

# Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated.

**Tuple example**

# TupSub = ( ‘ADC', ‘MC’ , ‘EE322’ )

**TupMob = (‘Iphone6’,Sony‘,’Appo’)**

# Print (TupSub) # Prints complete list

**Print (TupSub[0]) # Prints first element of the list**

# print (TupSub[1:3]) # Prints elements starting from 2nd till 3rd Print (TupSub[2:]) # Prints elements starting from 3rd element Print (TupMob \* 2 ) # Prints list two times

**print (TupSub + TupMob) # Prints concatenated lists Example : Program to merge two unsorted lists**

# A = [100,50,150]

**B = [9,51,20,3]**

# Merged sorted list C=[3,9,20,50,51,100,150] Step 1: Create two user input list.

**Step 2: Final merge list size is (size of the first list + the size of the second list). Step 3: Sort two lists using sort() method.**

# Step 4: Merge two sorted list and store it into a third list.

**Step 5: Merging remaining elements of a[] (if any).Merging remaining elements of b[] (if any).**

# Step 6: Display merged sorted list.

**Step 1: A = [100,50,150] B = [9,51,20,3]**

# Step 2: Final merged list is C and its size is (3+4). Step 3: Sort two lists using sort() method.

**A = [50,100,150] B = [3,9,20,51]**

# Step 4: Merge two sorted list and store it into a third list.

**C=[3,9,20,50,51]**

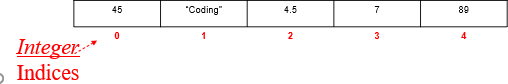
# Step 5: Merging remaining elements of A or B (if any).

**C=[3,9,20,50,51,100,150]**

# Step 6: Display merged sorted list.

**Dictionaries**

# Lists, tuples, and strings hold elements with only integer indices



**In essence, each element has an index (or a key) which can only be an integer**

# What if we want to store elements with non-integer indices (or keys)?

**Python Dictionary**

# Python 's dictionaries are hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs.

**Keys can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.**

# Dictionaries are enclosed by curly braces ( { } ) and values can be assigned and accessed using square braces ( [] ).

**Dictionaries**

# Lists index their entries based on the position in the list Dictionaries are like bags - no order

**So we index the things we put in the dictionary with a “lookup tag”**

# >>> purse = dict()

**>>> purse['money'] = 12**

# >>> purse['candy'] = 3

**>>> purse['tissues'] = 75**

# >>> print purse

**{'money': 12, 'tissues': 75, 'candy': 3}**

# >>> print purse['candy'] 3

**>>> purse['candy'] = purse['candy'] + 2’**

# >>> print purse

**{'money': 12, 'tissues': 75, 'candy': 5}**

# Example of dictionary

**dict = {}**

# dict['one'] = "This is one" dict[2] = "This is two“

**tinydict = {'name': 'john','code':6734, 'dept': 'sales'} print dict['one'] # Prints value for 'one' key print dict[2] # Prints value for 2 key**

# print tinydict # Prints complete dictionary print tinydict.keys() # Prints all the keys

**print tinydict.values() # Prints all the values OUTPUT:**

# This is one This is two

**{'dept': 'sales', 'code': 6734, 'name': 'john'} ['dept', 'code', 'name']**

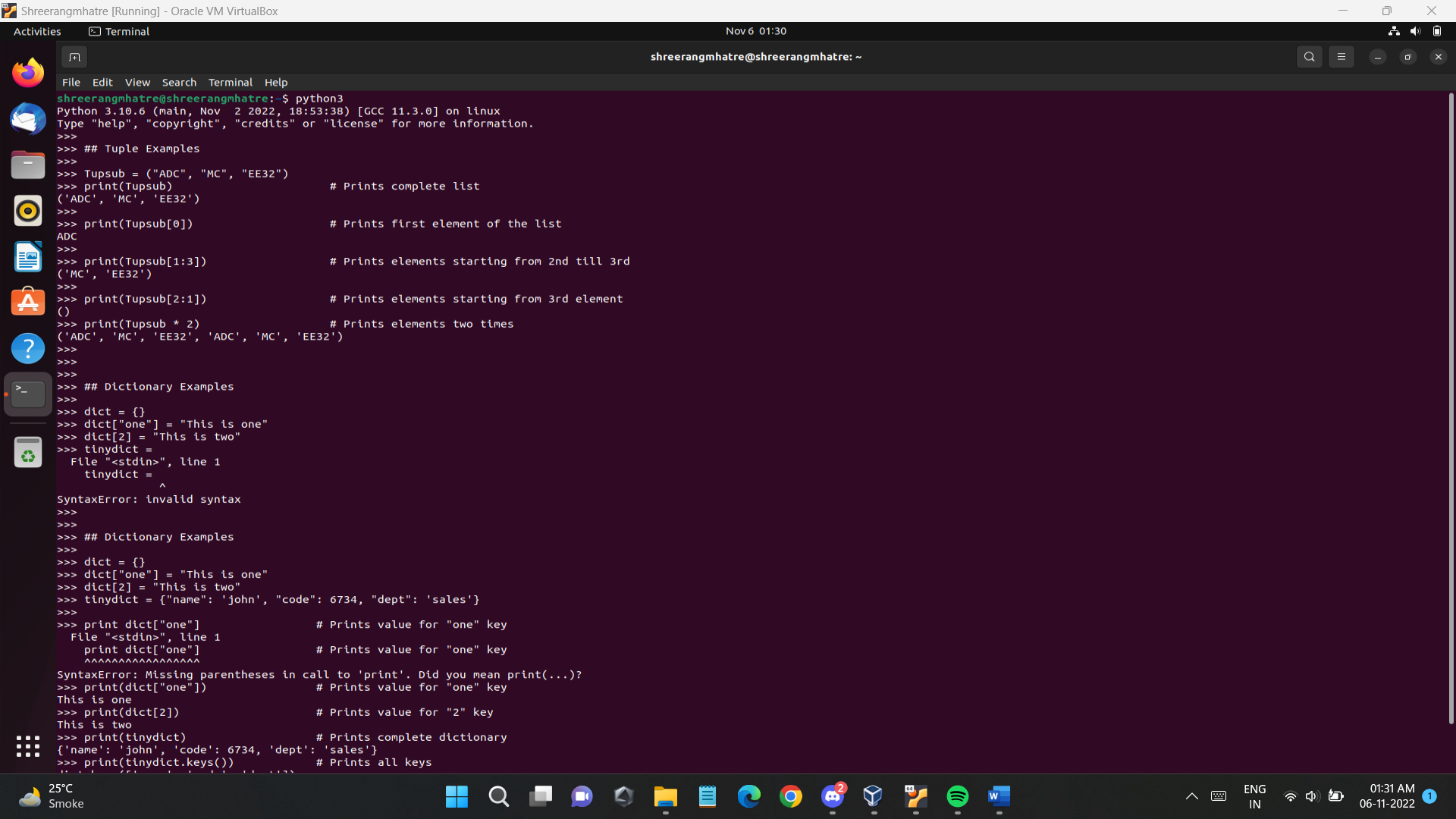
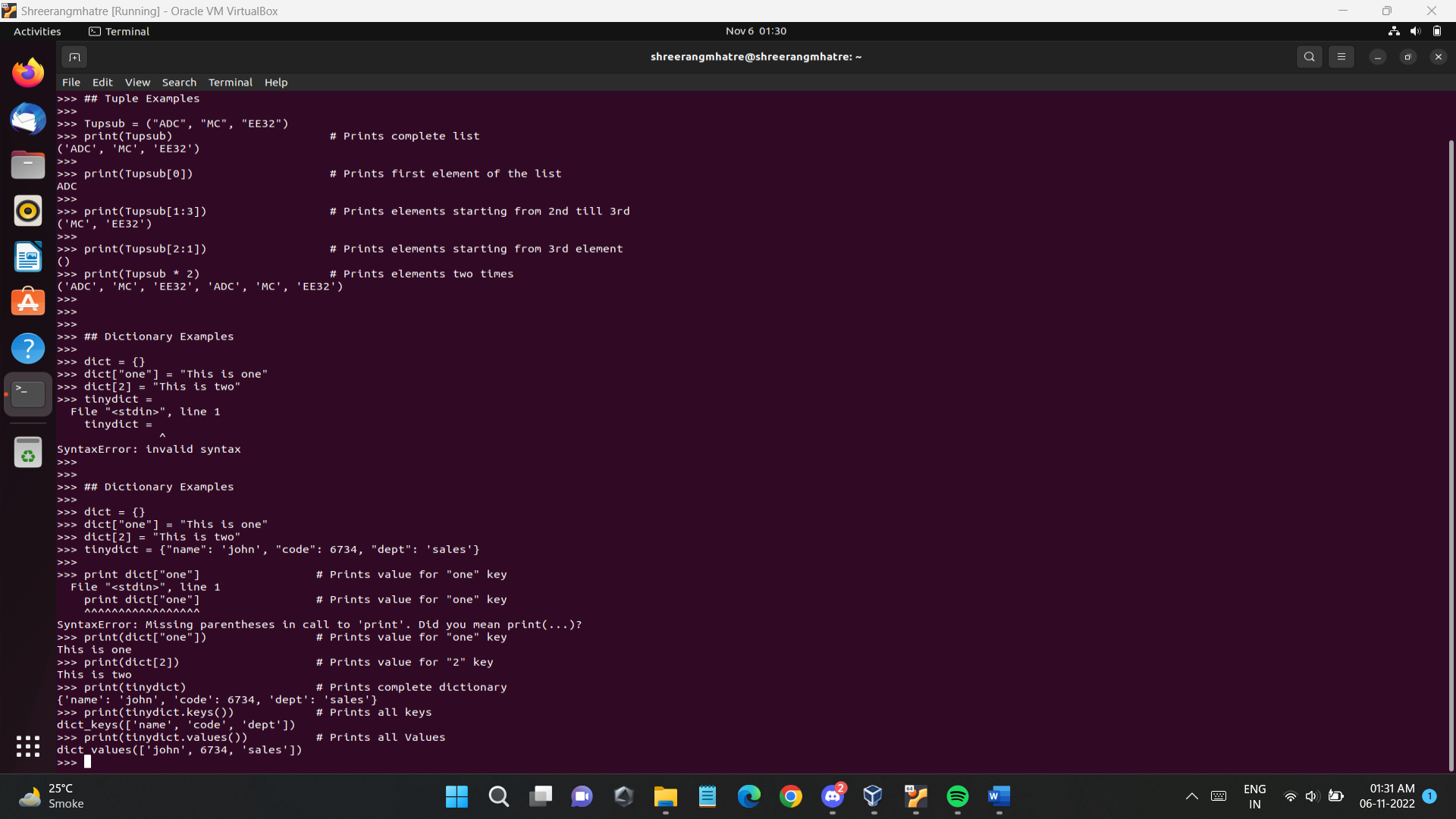
# ['sales', 6734, 'john']

**Input:** Pyhton Programs. **Output:** Output of each Program. **Conclusion:**

Thus, we have executed Advanced Data Structures of Python like Tuples and Dictionar

**Additional Reference Links:**

1. [https://www.python.org](https://www.python.org/)
2. [https://www.tutorialspoint.com](https://www.tutorialspoint.com/)
3. <https://www.programiz.com/python-programming>

******Procedure:**

# Post Lab Questions:

1. What are advanced data structures in python? Explain in detail.

Ans)

Advanced Data structures are one of the essential branches of data science which is used for storage, organization and management of data and information for efficient, easy accessibility and modification of data. They are the basic element for creating efficient and effective software design and algorithms.

The broad categories into which advanced data structure are divided are as follows:

**1. PRIMITIVE TYPES**

Primitive types are either a basic building block or are a built-in type support function.

**2. COMPOSITE OR NON-PRIMITIVE TYPE**

It is also known as structure or aggregate data type and can be constructed using a combination of primitive data and other composite data.

**3. ABSTRACT DATA TYPES**

In this data type, the behaviour is analyzed from the point of view of the user.

**4. LINEAR DATA STRUCTURES**

A data structure is linear if the elements of that data structure form a linear pattern or sequence.

**5. TREE TYPES**

Tree types include the main head (parent node) and then the branches (nodes) are divided based on subcategories and are further divided. This continues until all the elements have been allocated properly in their branches.

**6. HASH BASED STRUCTURES**

a) Hash list- It is a list of hashes of data blocks in a file or block for various purposes.

b) Double hashing- This is used to resolve hash collisions.

**7. GRAPHS**

A pictorial representation of data for better understanding purposes.

# Explain in detail Queues in Python

Ans)

A queue is a linear type of data structure used to store the data in a sequentially. The concept of queue is based on the FIFO, which means "First in First Out". It is also known as "first come first severed". The queue has the two ends front and rear. The next element is inserted from the rear end and removed from the front end.

We can perform the following operations in the Queue:

**Enqueue -** The enqueue is an operation where we add items to the queue. If the queue is full, it is a condition of the Queue The time complexity of enqueue is O(1).

**Dequeue -** The dequeue is an operation where we remove an element from the queue. An element is removed in the same order as it is inserted. If the queue is empty, it is a condition of the Queue Underflow. The time complexity of dequeue is O(1).

**Front -** An element is inserted in the front end. The time complexity of front is O(1).

**Rear -** An element is removed from the rear end.. The time complexity of the rear is O(1).

1. Write a program in python to Print the middle element of a given linked list

Ans)

**Program: -**

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

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

self.head = None

self.last\_node = None

def append(self, data):

if self.last\_node is None:

self.head = Node(data)

self.last\_node = self.head

else:

self.last\_node.next = Node(data)

self.last\_node = self.last\_node.next

def print\_middle(llist):

current = llist.head

length = 0

while current:

current = current.next

length = length + 1

current = llist.head

for i in range((length - 1)//2):

current = current.next

if current:

if length % 2 == 0:

print('The two middle elements are {} and {}.'

.format(current.data, current.next.data))

else:

print('The middle element is {}.'.format(current.data))

else:

print('The list is empty.')

a\_llist = LinkedList()

data\_list = input('Please enter the elements in the linked list: ').split()

for data in data\_list:

a\_llist.append(int(data))

print\_middle(a\_llist)

# Remove duplicate elements from a sorted linked list

# Ans)

# Program: -

# Python3 program to remove duplicate

# nodes from a sorted linked list

# Node class

class Node:

# Constructor to initialize

# the node object

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

# Function to initialize head

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

self.head = None

# Function to insert a new node

# at the beginning

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

self.head = new\_node

# Given a reference to the head of a

# list and a key, delete the first

# occurrence of key in linked list

def deleteNode(self, key):

# Store head node

temp = self.head

# If head node itself holds the

# key to be deleted

if (temp is not None):

if (temp.data == key):

self.head = temp.next

temp = None

return

# Search for the key to be deleted,

# keep track of the previous node as

# we need to change 'prev.next'

while(temp is not None):

if temp.data == key:

break

prev = temp

temp = temp.next

# if key was not present in

# linked list

if(temp == None):

return

# Unlink the node from linked list

prev.next = temp.next

temp = None

# Utility function to print the

# linked LinkedList

def printList(self):

temp = self.head

while(temp):

print(temp.data , end = ' ')

temp = temp.next

# This function removes duplicates

# from a sorted list

def removeDuplicates(self):

temp = self.head

if temp is None:

return

while temp.next is not None:

if temp.data == temp.next.data:

new = temp.next.next

temp.next = None

temp.next = new

else:

temp = temp.next

return self.head

# Driver Code

llist = LinkedList()

llist.push(20)

llist.push(13)

llist.push(13)

llist.push(11)

llist.push(11)

llist.push(11)

print ("Created Linked List: ")

llist.printList()

print()

print("Linked List after removing",

"duplicate elements:")

llist.removeDuplicates()

# llist.printList()

# # This code is contributed by

# # Dushyant Pathak.