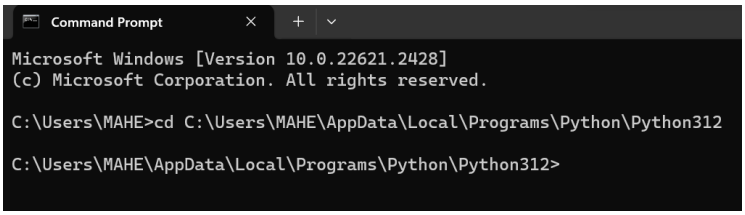
DATA STRUCTURES LABORATORY MANUAL – ICE 2144

III SEMESTER B. TECH

EXPT. 10 DATA STRUCTURES USING *Python*

Precautions:

- Ensure that the codes are saved in *.py* format.
- Save the file(s) in the PATH (i.e. the directory/address) in which the Python .exe file is saved
- Ensure that all PYTHON packages, viz. NumPy, are installed. An example of installing NumPy is shown as follows:
    - Open the COMMAND PROMPT and go to the PATH of the software.



Here, 'cd' means *change [the] directory.*

    - Type *pip3 install numpy*:

```
C:\Users\MAHE\AppData\Local\Programs\Python\Python312> pip3 install numpy
```

    - You may re-try the aforementioned step to ensure the installation of the python3 package. If installed properly, the following message will be displayed.

```
C:\Users\MAHE\AppData\Local\Programs\Python\Python312> pip3 install numpy
Requirement already satisfied: numpy in c:\users\mahe\appdata\local\programs\python\python312\lib\site-packages (1.26.1)
```

- To execute a .py file, open the PYTHON IDLE and press CNTRL + O i.e. open the folder containing the desired .py file. Then, go to the .py file script (opened) and press Fn.5 key to RUN the code.
- WARNING!! ➔ .py files <u>will never be executed</u> unless and until all errors in the <u>indenting</u> of the codes are corrected!!

DATA STRUCTURES LABORATORY MANUAL – ICE 2144

III SEMESTER B. TECH

EXPT. 10 DATA STRUCTURES USING *Python*

1) Write a Python program to add elements to an array.

Program:

```
# Python program to demonstrate Adding Elements to an Array
# importing "array" for array creations
import array as arr
# array with int type
a = arr.array('i', [1, 2, 3])
print("Array before insertion : ", end=" ")
for i in range(0, 3):
    print(a[i], end=" ")
print()
# inserting array using insert() function
a.insert(1, 4)
print("Array after insertion : ", end=" ")
for i in (a):
    print(i, end=" ")
print()
# array with float type
b = arr.array('d', [-2.5, 3.2, -3.3])
print("\nArray before insertion : ", end=" ")
for i in range(0, 3):
    print(b[i], end=" ")
print()
# adding an element using append()
b.append(4.4)
print("Array after insertion : ", end=" ")
```

```
for i in (b):
    print(i, end=" ")
print()
```

2) Write a Python program to access multiple elements in a list (using NumPy function).

Program:

```
# Import libraries required
import numpy as np

# initialize input list and index list
test_list = [9, 4, 5, 8, 10, 14]
index_list = [1, 3, 4]

# print original lists
print("Original list : " + str(test_list))
print("Original index list : " + str(index_list))

# use numpy.take() to retrieve elements from input list at given indices
res_list = np.take(test_list, index_list)
# print resultant list
print("Resultant list : " + str(res_list))
```

3) Write a Python program to define instance variables using a constructor

```
class MAHE:
    # Class Variable
    student = 'MIT'
```

DATA STRUCTURES LABORATORY MANUAL – ICE 2144

III SEMESTER B. TECH

EXPT. 10 DATA STRUCTURES USING *Python*

```
    # The init method or constructor
    def __init__(self, branch, section):

        # Instance Variable
        self.branch = branch
        self.section = section


# Objects of MAHE class
Vaanya = MAHE("EIE", "3A")
Paul = MAHE("CPS", "3B")
print('Vaanya details:\n================')
print('Vaanya is a UG student of: ',Vaanya.student)
print('Branch: ', Vaanya.branch)
print('Section: ', Vaanya.section)
print('\nPaul details:\n================')
print('Vaanya is a UG student of: ', Paul.student)
print('Branch: ', Paul.branch)
print('Section: ', Paul.section)
# Class variables can be accessed using class and name also
print("\nAccessing class variable using class name is :", MAHE.student)
```

DATA STRUCTURES LABORATORY MANUAL – ICE 2144

III SEMESTER B. TECH

EXPT. 10 DATA STRUCTURES USING *Python*

4) Write a program to create a List of Tuples without using a built-in function

```
# Function to create a list of tuples
def create_list_of_tuples(lst1, lst2):
    result = []  # Empty list to store the tuples
    for i in range(len(lst1)):
        # Create a tuple from corresponding elements
        tuple_element = (lst1[i], lst2[i])
        result.append(tuple_element)  # Append the tuple to the list
    return result
# Example usage
list1 = [1, 2, 3]
list2 = ['a', 'b', 'c']
list_of_tuples = create_list_of_tuples(list1, list2)
print(list_of_tuples)
```

5) Write a Python program to remove nodes from a given Linked List.

```
# Create a Node class to create a node
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

# Create a LinkedList class
class LinkedList:
    def __init__(self):
```

```
        self.head = None

    # Method to add a node at begin of LL
    def insertAtBegin(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        else:
            new_node.next = self.head
            self.head = new_node

    # Method to add a node at the end of LL
    def insertAtEnd(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        current_node = self.head
        while(current_node.next):
            current_node = current_node.next
        current_node.next = new_node

# Update node of a linked list at given position
    def updateNode(self, val, index):
        current_node = self.head
        position = 0
```

```
        if position == index:
                current_node.data = val
        else:
                while(current_node != None and position != index):
                        position = position+1
                        current_node = current_node.next

                if current_node != None:
                        current_node.data = val
                else:
                        print("Index not present")

# Method to remove first node of linked list

def remove_first_node(self):
        if(self.head == None):
                return
        self.head = self.head.next

# Method to remove last node of linked list
def remove_last_node(self):
        if self.head is None:
                return
        current_node = self.head
        while(current_node.next.next):
                current_node = current_node.next
```

```
        current_node.next = None


# Method to remove at given index
def remove_at_index(self, index):
        if self.head == None:
                return
        current_node = self.head
        position = 0
        if position == index:
                self.remove_first_node()
        else:
                while(current_node != None and position+1 != index):
                        position = position+1
                        current_node = current_node.next
                if current_node != None:
                        current_node.next = current_node.next.next
                else:
                        print("Index not present")


# Method to remove a node from linked list
def remove_node(self, data):
        current_node = self.head
        while(current_node != None and current_node.next.data != data):
                current_node = current_node.next
        if current_node == None:
                return
        else:
```

```
                current_node.next = current_node.next.next
        # Print the size of linked list
        def sizeOfLL(self):
                size = 0
                if(self.head):
                        current_node = self.head
                        while(current_node):
                                size = size+1
                                current_node = current_node.next
                        return size
                else:
                        return 0
        # print method for the linked list
        def printLL(self):
                current_node = self.head
                while(current_node):
                        print(current_node.data)
                        current_node = current_node.next


# create a new linked list
llist = LinkedList()
# add nodes to the linked list
llist.insertAtEnd('a-->')
llist.insertAtEnd('-->b-->')
llist.insertAtEnd('-->c-->')
llist.insertAtEnd('-->d-->')
llist.insertAtEnd('-->e-->')
```

```
llist.insertAtEnd('-->f')
# print the original linked list
print("\nOriginal Node Data of SLL")
llist.printLL()
print("\nSize of original linked list:", end=" ")
print(llist.sizeOfLL())
print('\n')
llist.insertAtBegin('X<--')
llist.insertAtEnd('-->Y')
#llist.insertAtIndex('g', 2)
# print the new linked list
print("\nUpdated Node Data of SLL")
llist.printLL()
print("\nSize of linked list before Updation:", end=" ")
print(llist.sizeOfLL())print('\n')
# remove a nodes from the linked list
llist.remove_first_node()
print("\nRemoved the First Node.\n#######")
llist.remove_last_node()
print("\nRemoved the Last Node. \n#######")
# print the linked list again
print("\nLinked list after removing a node:")
llist.printLL()
print("\nSize of linked list after Updation:", end=" ")
print(llist.sizeOfLL())
```

DATA STRUCTURES LABORATORY MANUAL – ICE 2144

III SEMESTER B. TECH

EXPT. 10 DATA STRUCTURES USING *Python*

Exercise:

1. Write a program (in Python) to search for a key in the given B.S. Tree and delete the node containing the key.

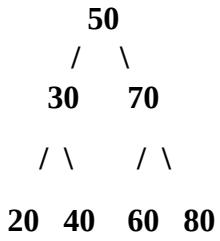


Fig: 1 Binary Search Tree

2. Write a program to create/generate the following tree structures (Figures 2 and 3). Also, display the max and min values in each tree.

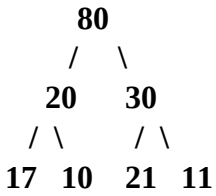


Fig.2 Max Heap

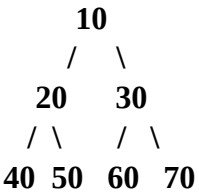


Fig.3 Min Heap