



MALAD KANDIVALI EDUCATION SOCIETY'S
NAGINDAS KHANDWALA COLLEGE OF COMMERCE, ARTS &
MANAGEMENT STUDIES & SHANTABEN NAGINDAS
KHANDWALA COLLEGE OF SCIENCE
MALAD [W], MUMBAI – 64
AUTONOMOUS INSTITUTION
(Affiliated To University Of Mumbai)
Reaccredited 'A' Grade by NAAC | ISO 9001:2015 Certified

CERTIFICATE

Name: Mr./Ms. MITALI PRASHANT VISHWEKAR

Roll No: 367 Programme: BSc CS Semester: III

This is certified to be a bonafide record of practical works done by the above student in the college laboratory for the course **Data Structures (Course Code: 2032UISPR)** for the partial fulfilment of Third Semester of BSc IT during the academic year 2020-21.

The journal work is the original study work that has been duly approved in the year 2020-21 by the undersigned.

External Examiner

Mr. Gangashankar Singh
(Subject-In-Charge)

Date of Examination: (College Stamp)

Class: S.Y. B.Sc. CS Sem- III**Roll No: 367****Subject: Data Structures****INDEX**

| Sr No | Date | Topic | Sign |
|--------------|-------------|--|-------------|
| 1 | 04/09/2020 | Implement the following for Array: a) Write a program to store the elements in 1-D array and provide an option to perform the operations like searching, sorting, merging, reversing the elements. b) Write a program to perform the Matrix addition, Multiplication and Transpose Operation. | |
| 2 | 11/09/2020 | Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate two linked lists. | |
| 3 | 18/09/2020 | Implement the following for Stack: a) Perform Stack operations using Array implementation. b. b) Implement Tower of Hanoi. c) WAP to scan a polynomial using linked list and add two polynomials. d) WAP to calculate factorial and to compute the factors of a given no. (i) using recursion, (ii) using iteration | |
| 4 | 25/09/2020 | Perform Queues operations using Circular Array implementation. | |
| 5 | 01/10/2020 | Write a program to search an element from a list. Give user the option to perform Linear or Binary search. | |
| 6 | 09/10/2020 | WAP to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort. | |
| 7 | 16/10/2020 | Implement the following for Hashing: a) Write a program to implement the collision technique. b) Write a program to implement the concept of linear probing. | |
| 8 | 23/10/2020 | Write a program for inorder, postorder and preorder traversal of tree. | |

Git hub Link: - <https://github.com/mitalivishwekar/DS>

PRACTICAL 1

Aim: - Implement the following for Array:

- a. Write a program to store the elements in 1-D array and provide an option to perform the operations like searching, sorting, merging, reversing the elements.
- b. Write a program to perform the Matrix addition, Multiplication and Transpose Operation.

Theory: -Array is a container which can hold a fix number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms. Following are the important terms to understand the concept of Array.

Element– Each item stored in an array is called an element.

Index – Each location of an element in an array has a numerical index, which is used to identify the element.

Basic Operations

Following are the basic operations supported by an array.

Traverse – print all the array elements one by one.

Insertion – Adds an element at the given index.

Search – Searches an element using the given index or by the value.

Sorting-Means Arranging the Element in particular order. i.e. Ascending or Descending order

Input: -

Practical_1.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical_1.py (3.7.4)

File Edit Format Run Options Window Help

```
# Name :Mitali Vishwekar
# Rollno:4020
# Class :SYCS
from array import *
class Stack():
    def __init__(self):
        self.items = array('i', [4, 3, 2, 4020])

    def end(self, item):
        self.items.append(item)
        print(item)

    def peek(self):
        if self.items:
            return self.items[-1]
        else:
            return None

    def size(self):
        if self.items:
            return len(self.items)
        else:
            return None

    def display(self):
        for i in self.items:
            print(i)

    def start(self, i):
        self.items.insert(0, i)
#searching
    def search(self, a):
        l = self.items
        for i in l:
            if i == a:
                print("found Value : ", a)
                break
            else:
                print("not found")

    def traverse(self):
        a = []
        l = self.items
        for i in l:
            a.append(i)
        print(a)
```

Practical_1.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical_1.py (3.7.4)

File Edit Format Run Options Window Help

```

    print(a)
#shorting
    def shoting_element(self):
        #bubble shoting
        nums=self.items
        def sort(nums):
            for i in range(len(nums) - 1, 0, -1):
                for j in range(i):
                    if nums[j] > nums[j + 1]:
                        temp = nums[j]
                        nums[j] = nums[j + 1]
                        nums[j + 1] = temp

            sort(nums)
            print(nums)
        def reverse(self):
            l=self.items
            print(l[::-1])
#class is made to merge two array
class mergel(Stack):
    #inheritance is created to merfe two array
    def __init__(self):
        Stack.__init__(self)
        self.items1 = array('i', [4,3,2,1,6])

    def merge(self):
        l = self.items
        l1=self.items1
        a=(l+l1)
        print(a)

s = Stack()
# Inserting the values
s.end(5)
s.end(6)
s.end(7)
s.start(-1)
s.start(-2)
print("search the specific value : ")
s.search(-2)

print("Display the values one by one :")
s.display()
print("peek (End Value) :", s.peek())
print("treverse the values : ")
s.traverse()

#Shotting element
print("Shotting the values : ")
s.shoting_element()

print("Reversing the values : ")
s.reverse()

s1=mergel()
print("merge")
s1.merge()

```

Output: -

```

>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical_1.py
5
6
7
search the specific value :
found Value : -2
Display the values one by one :
-2
-1
4
3
2
4020
5
6
7
peek (End Value) : 7
treverse the values :
[-2, -1, 4, 3, 2, 4020, 5, 6, 7]
Shotting the values :
array('i', [-2, -1, 2, 3, 4, 5, 6, 7, 4020])
Reversing the values :
array('i', [4020, 7, 6, 5, 4, 3, 2, -1, -2])
merge
array('i', [4, 3, 2, 4020, 4, 3, 2, 1, 6])
>>>

```

PRACTICAL 2

Aim: - Implement Linked List. Include options for insertion, deletion and search of a number, reverse the list and concatenate two linked lists.

Theory: -A linked list is a sequence of data elements, which are connected together via links. Each data element contains a connection to another data element in form of a pointer. Python does not have linked lists in its standard library. We implement the concept of linked lists using the concept of nodes as discussed in the previous chapter. We have already seen how we create a node class and how to traverse the elements of a node. In this chapter we are going to study the types of linked lists known as singly linked lists. In this type of data structure there is only one link between any two data elements. We create such a list and create additional methods to insert, update and remove elements from the list.

Input: -

Practical_2.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical_2.py (3.7.4)
File Edit Format Run Options Window Help

```
# Name :MITALI VISHWEKAR
# Rollno:4020
# Class :SYCS
class Stack():
    def __init__(self):
        self.items = ['4','3','2','1','MITALI']

    def end(self, item):
        self.items.append(item)
        print(item)

    def peek(self):
        if self.items:
            return self.items[-1]
        else:
            return None

    def size(self):
        if self.items:
            return len(self.items)
        else:
            return None

    def display(self):
        for i in self.items:
            print(i)

    def start(self, i):
        self.items.insert(0, i)

    def search(self, a):
        l = self.items
        for i in l:
            if i == a:
                print("found Value : ", a)
                break
        else:
            print("not found")

    def traverse(self):
        a = []
        l = self.items
        for i in l:
            a.append(i)
        print(a)
```

Practical_2.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical_2.py (3.7.4)

File Edit Format Run Options Window Help

```

def shoting_element(self):
    #bubble shoting
    nums=self.items
    def sort(nums):
        for i in range(len(nums) - 1, 0, -1):
            for j in range(i):
                if nums[j] > nums[j + 1]:
                    temp = nums[j]
                    nums[j] = nums[j + 1]
                    nums[j + 1] = temp

    sort(nums)
    print(nums)
#reverse
def reverse(self):
    l=self.items
    print(l[::-1])

def remove_value_from_particular_index(self,a):
    l=self.items
    l.pop(a)
    print(l)

class mergel(Stack):
    #inheritance
    def __init__(self):
        Stack.__init__(self)
        self.items1 = ['4','3','2','1','6']

    def merge(self):
        l = self.items
        l1=self.items1
        a=(l+l1)
        a.sort()
        print(a)

s = Stack()
# Inserting the values
s.end('-1')
s.start('-2')
s.start('5')
s.end('6')
s.end('7')
s.start('-1')
s.start('-2')
print("search the specific value : ")
s.search('-2')

print("Display the values one by one :")
s.display()
print("peek (End Value) :", s.peek())
print("treverse the values : ")
s.traverse()
#Shotting element
print("Shotting the values : ")
s.shoting_element()
#reversing the list
print("Reversing the values : ")
s.reverse()

print("remove value from particular index which is defined earlier")
s.remove_value_from_particular_index(0)

s1=mergel()
print("merge")
s1.merge()

```

Output: -

```
>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical_2.py
-1
6
7
search the specific value :
found Value : -2
Display the values one by one :
-2
-1
5
-2
4
3
2
1
MITALI
-1
6
7
peek (End Value) : 7
treverse the values :
['-2', '-1', '5', '-2', '4', '3', '2', '1', 'MITALI', '-1', '6', '7']
Shotting the values :
['-1', '-1', '-2', '-2', '1', '2', '3', '4', '5', '6', '7', 'MITALI']
Reversing the values :
['MITALI', '7', '6', '5', '4', '3', '2', '1', '-2', '-2', '-1', '-1']
remove value from particular index which is defined earlier
['-1', '-2', '-2', '1', '2', '3', '4', '5', '6', '7', 'MITALI']
merge
['1', '1', '2', '2', '3', '3', '4', '4', '6', 'MITALI']
>>> |
```


PRACTICAL 3a

Aim: - Implement the following for Stack

Theory: Array is a container which can hold a fix number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms. Following are the important terms to understand the concept of Array.

Element– Each item stored in an array is called an element.

Index – Each location of an element in an array has a numerical index, which is used to identify the element.

Practical 3a.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical 3a.py (3.7.4)

File Edit Format Run Options Window Help

```
# Name :Mitali Vishwekar
# Rollno:4020
# Class :SYCS
from sys import maxsize

def createStack():
    stack = []
    return stack

def isEmpty(stack):
    return len(stack) == 0

def push(stack, item):
    stack.append(item)
    print(item + " pushed to stack ")

def pop(stack):
    if (isEmpty(stack)):
        return str(-maxsize -1)

    return stack.pop()

def peek(stack):
    if (isEmpty(stack)):
        return str(-maxsize -1)
    return stack[len(stack) - 1]

stack = createStack()
push(stack, str(10))
push(stack, str(20))
push(stack, str(30))
print(pop(stack) + " popped from stack")
```

Output: -

```
>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical 3a.py
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
>>> |
```

PRACTICAL 3B

Aim: -Implement Tower of Hanoi

Theory: - Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- 1) Only one disk can be moved at a time.
- 2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- 3) No disk may be placed on top of a smaller disk.

Input: -

```

3b tower of hanoi.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\3b tower of hanoi.py (3.7.4)
File Edit Format Run Options Window Help

#MITALI
#SYCS
#4020
def TowerOfHanoi(n , source, destination, auxiliary):
    if n==1:
        print ("Move disk 1 from source",source,"to destination",destination )
        return
    TowerOfHanoi(n-1, source, auxiliary, destination)
    print ("Move disk",n,"from source",source,"to destination",destination )
    TowerOfHanoi(n-1, auxiliary, destination, source)

n = 4
TowerOfHanoi(n, 'A', 'B', 'C')
```

Output: -

```

>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\3b tower of hanoi.py
Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B
Move disk 3 from source A to destination C
Move disk 1 from source B to destination A
Move disk 2 from source B to destination C
Move disk 1 from source A to destination C
Move disk 4 from source A to destination B
Move disk 1 from source C to destination B
Move disk 2 from source C to destination A
Move disk 1 from source B to destination A
Move disk 3 from source C to destination B
Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B
>>> |
```

Practical 3c

Aim: WAP to scan a polynomial using linked list and add two polynomials.

Theory: A linked list is a sequence of data elements, which are connected together via links. Each data element contains a connection to another data element in form of a pointer. Python does not have linked lists in its standard library. We implement the concept of linked lists using the concept of nodes as discussed in the previous chapter. We have already seen how we create a node class and how to traverse the elements of a node. In this chapter we are going to study the types of linked lists known as singly linked lists. In this type of data structure there is only one link between any two data elements. We create such a list and create additional methods to insert, update and remove elements from the list.

Input: -

Practical 3c.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical 3c.py (3.7.4)

File Edit Format Run Options Window Help

```
#MITALI
#SYCS
#4020
def add(A, B, m, n):

    size = max(m, n);
    sum = [0 for i in range(size)]
    for i in range(0, m, 1):
        sum[i] = A[i]

    for i in range(n):
        sum[i] += B[i]

    return sum
def printPoly(poly, n):
    for i in range(n):
        print(poly[i], end = "")
        if (i != 0):
            print("x^", i, end = "")
        if (i != n - 1):
            print(" + ", end = "")

if __name__ == '__main__':

    A = [5, 0, 10, 6]

    B = [1, 2, 4]
    m = len(A)
    n = len(B)

    print("First polynomial is")
    printPoly(A, m)
    print("\n", end = "")
    print("Second polynomial is")
    printPoly(B, n)
    print("\n", end = "")
    sum = add(A, B, m, n)
    size = max(m, n)

    print("sum polynomial is")
    printPoly(sum, size)
```

Output: -

```
>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical 3c.py
First polynomial is
5 + 0x^ 1 + 10x^ 2 + 6x^ 3
Second polynomial is
1 + 2x^ 1 + 4x^ 2
sum polynomial is
6 + 2x^ 1 + 14x^ 2 + 6x^ 3
>>> |
```

Practical 3d

Aim: -WAP to calculate factorial and to compute the factors of a given no. (i) using recursion, (ii) using iteration.

Input (1): -

 3d using recursion.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\3d using recursion.py (3.7.4)

File Edit Format Run Options Window Help


```
#MITALI
#SYCS
#4020
def recur_factorial(n):
    if n == 1:
        return n
    else:
        return n*recur_factorial(n-1)

num = int(input("Enter a number: "))

if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    print("The factorial of",num,"is",recur_factorial(num))
```

Output (1): -

```
>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\3d using recursion.py
Enter a number: 6
The factorial of 6 is 720
>>> |
```

Input (2): - 3d using iteration.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\3d using iteration.py (3.7.4)

File Edit Format Run Options Window Help

```
#MITALI
#SYCS
#4020
def fact(number):

    fact = 1

    for number in range(number, 1,-1):

        fact = fact * number
    return fact

number = int(input("Enter The Number : "))

factorial = fact(number)
print("Factorial is "+str(factorial))
```

Output (2): -

```
>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\3d using iteration.py
Enter The Number : 5
Factorial is 120
>>> |
```

PRACTICAL 4

Aim: - Perform Queues operations using Circular Array implementation.

Theory: - A Circular Queue is a queue data structure but circular in shape, therefore after the last position, the next place in the queue is the first position. We recommend you to first go through the Linear Queue tutorial before Circular queue, as we will be extending the same implementation. In case of Linear queue, we did not had the head and tail pointers because we used python List for implementing it. But in case of a circular queue, as the size of the queue is fixed, hence we will set a maxSize for our list used for queue implementation.

Input: -

Practical 4.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical 4.py (3.7.4)

File Edit Format Run Options Window Help

```
#MITALI VISHWEKAR
#SYCS
#4020
class Stack():
    def __init__(self):
        self.items = [1,2,3,4,5]

    def enqueue(self,item):
        self.items.append(item)
        print(item)

    def deque(self):
        b= self.items
        b.pop()
        print(b)

    def traverse(self):
        a = []
        l = self.items
        for i in l:
            a.append(i)
        print(a)

s=Stack()

print("Adding the element in the queue : ")
s.enqueue(6)
print("initial queue : ")
s.traverse()

print("After removing an element from the queue : ")
s.deque()
|
```

Output: -

```
>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical 4.py
Adding the element in the queue :
6
initial queue :
[1, 2, 3, 4, 5, 6]
After removing an element from the queue :
[1, 2, 3, 4, 5]
>>> |
```

PRACTICAL 5

Aim: -Write a program to search an element from a list. Give user the option to perform Linear or Binary search.

Theory: - Searching is a very basic necessity when you store data in different data structures. The simplest approach is to go across every element in the data structure and match it with the value you are searching for. This is known as Linear search. It is inefficient and rarely used, but creating a program for it gives an idea about how we can implement some advanced search algorithms.

Linear Search:

In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data structure.

Binary Search:

Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Input: -

```
practical_5.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\practical_5.py (3.7.4)
File Edit Format Run Options Window Help

#MITALI VISHWEKAR
#SYCS
#4020
a = str(input("Enter the string 1 for Linear Search , b For Binary Search: "))
pos = -1
list = [0,1,2,3,4,5,6,7,8,45,20]
if a == 'b':
    def search(list,n):
        l = 0
        u = len(list)-1
        while l <=u:
            mid = (l+u)//2

            if list[mid] == n:
                globals()['pos']= mid
                return True
            else:
                if list[mid]<n:
                    l = mid+1
                else:
                    u = mid-1
        return False

    list.sort()
    n= int(input("Enter the numbers for binary search : "))
    if search(list, n):
        print("Number Found ")
    else:
        print("Not Found ")
elif a == 'l':
    def search(list ,n):
        i = 0

        while i < len(list):
            if list[i] == n:
                return True
            i = i+1

        return False

    list.sort()
    n= int(input("Enter the numbers for linear search : "))
    if search(list ,n):
        print("Number found ")
    else:
        print("not found")
else:
    print("enter valid input")
```

Output: -

```
>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\practical_5.py
Enter the string l for Linear Search , b For Binary Search: l
Enter the numbers for linear search : 2
Number found
>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\practical_5.py
Enter the string l for Linear Search , b For Binary Search: b
Enter the numbers for binary search : 72
Not Found
>>> |
```

PRACTICAL 6

Aim: - WAP to sort a list of elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.

Theory: - Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order. The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats. Below we see five such implementations of sorting in python.

Bubble Sort

It is a comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

Insertion Sort

Insertion sort involves finding the right place for a given element in a sorted list. So in beginning we compare the first two elements and sort them by comparing them. Then we pick the third element and find its proper position among the previous two sorted elements. This way we gradually go on adding more elements to the already sorted list by putting them in their proper position.

Selection Sort

In selection sort we start by finding the minimum value in a given list and move it to a sorted list. Then we repeat the process for each of the remaining elements in the unsorted list. The next element entering the sorted list is compared with the existing elements and placed at its correct position. So at the end all the elements from the unsorted list are sorted.

Input: -

```

Practical_6.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical_6.py (3.7.4)
File Edit Format Run Options Window Help

#MITALI VISHWEKAR
#SYCS
#4020
nums = [5,4,4020,-1]
a = str(input("enter the string i for insertion sort , b for bubble sort , s for selection sort : "))
if a=='i' or a=='I':

    def insertion_sort(nums):
        for i in range(1, len(nums)):
            j = i-1
            nxt_element = nums[i]

            while (nums[j] > nxt_element) and (j >= 0):
                nums[j+1] = nums[j]
                j=j-1
            nums[j+1] = nxt_element

        insertion_sort(nums)
        print(nums)
elif a == 'b' or a == 'B':

    def sort(nums):
        for i in range(len(nums)-1,0,-1):
            for j in range(i):
                if nums[j]>nums[j+1]:
                    temp = nums[j]
                    nums[j]=nums[j+1]
                    nums[j+1] = temp

        sort(nums)
        print(nums)
elif a == 's' or a == 'S':
    def sort(nums):
        for i in range(len(nums)):
            minpos = i
            for j in range(i,len(nums)):
                if nums[j] < nums[minpos]:
                    minpos=j
            temp = nums[i]
            nums[i] = nums[minpos]
            nums[minpos] =temp

        sort(nums)
        print(nums)
else:
    print("Enter valid input")

```

Output: -

```

>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical_6.py
enter the string i for insertion sort , b for bubble sort , s for selection sort : i
[-1, 4, 5, 4020]
>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical_6.py
enter the string i for insertion sort , b for bubble sort , s for selection sort : b
[-1, 4, 5, 4020]
>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\Practical_6.py
enter the string i for insertion sort , b for bubble sort , s for selection sort : s
[-1, 4, 5, 4020]
>>> |

```

PRACTICAL 7


Aim: - Implement the following for Hashing:

- a. Write a program to implement the collision technique.

Theory: - Hash tables are a type of data structure in which the address or the index value of the data element is generated from a hash function. That makes accessing the data faster as the index value behaves as a key for the data value. In other words Hash table stores key-value pairs but the key is generated through a hashing function. So the search and insertion function of a data element becomes much faster as the key values themselves become the index of the array which stores the data. In Python, the Dictionary data types represent the implementation of hash tables. The Keys in the dictionary satisfy the following requirements.

- The keys of the dictionary are hash able i.e. the are generated by hashing function which generates unique result for each unique value supplied to the hash function.
- The order of data elements in a dictionary is not fixed.

Input (a): -

 practical7a.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\practical7a.py (3.7.4)

File Edit Format Run Options Window Help

```
#MITALI VISHWEKAR
#SYCS
#4020
class Hash:
    def __init__(self, keys, lowerrange, higherrange):
        self.value = self.hashfunction(keys, lowerrange, higherrange)

    def get_key_value(self):
        return self.value

    def hashfunction(self, keys, lowerrange, higherrange):
        if lowerrange == 0 and higherrange > 0:
            return keys%(higherrange)

if __name__ == '__main__':
    list_of_keys = [23, 43, 1, 87]
    list_of_list_index = [None, None, None, None]
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:
        list_index = Hash(value, 0, len(list_of_keys)).get_key_value()
        if list_of_list_index[list_index]:
            print("Collission detected")
        else:
            list_of_list_index[list_index] = value
    print("After: " + str(list_of_list_index))
```

Output (a): -

```
>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\practical7a.py
Before : [None, None, None, None]
Collision detected
Collision detected
After: [None, 1, None, 23]
>>> |
```

B. Write a program to implement the concept of linear probing.

Theory: - Hashing is an important Data Structure which is designed to use a special function called the Hash function which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends of the efficiency of the hash function used.

In Open Addressing, all elements are stored in the hash table itself. So at any point, size of table must be greater than or equal to total number of keys (Note that we can increase table size by copying old data if needed).

Input (b): -

```
practical7b.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\practical7b.py (3.7.4)
File Edit Format Run Options Window Help

#MITALI VISHWEKAR
#SYCS
#4020
class Hash:
    def __init__(self, keys, lowerrange, higherrange):
        self.value = self.hashfunction(keys, lowerrange, higherrange)

    def get_key_value(self):
        return self.value

    def hashfunction(self, keys, lowerrange, higherrange):
        if lowerrange == 0 and higherrange > 0:
            return keys % (higherrange)
if __name__ == '__main__':
    linear_probing = True
    list_of_keys = [23, 43, 1, 87, 32, 34, 67, 77, 45, 54]
    list_of_list_index = [None]*len(list_of_keys)
    print("Before : " + str(list_of_list_index))
    for value in list_of_keys:
        # print(Hash(value,0,len(list_of_keys)).get_key_value())
        list_index = Hash(value, 0, len(list_of_keys)).get_key_value()
        print("hash value for " + str(value) + " is : " + str(list_index))
        if list_of_list_index[list_index]:
            print("Collision detected for " + str(value))
            if linear_probing:
                old_list_index = list_index
                if list_index == len(list_of_list_index)-1:
                    list_index = 0
                else:
                    list_index += 1
            list_full = False
            while list_of_list_index[list_index]:
                if list_index == old_list_index:
                    list_full = True
                    break
                if list_index+1 == len(list_of_list_index):
                    list_index = 0
                else:
                    list_index += 1
            if list_full:
                print("List was full . Could not save")
            else:
                list_of_list_index[list_index] = value
        else:
            list_of_list_index[list_index] = value

    print("After: " + str(list_of_list_index))
```

Output (b): -

```
///
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\practical7b.py
Before : [None, None, None, None, None, None, None, None, None, None]
hash value for 23 is :3
hash value for 43 is :3
Collision detected for 43
hash value for 1 is :1
hash value for 87 is :7
hash value for 32 is :2
hash value for 34 is :4
Collision detected for 34
hash value for 67 is :7
Collision detected for 67
hash value for 77 is :7
Collision detected for 77
hash value for 45 is :5
Collision detected for 45
hash value for 54 is :4
Collision detected for 54
After: [54, 1, 32, 23, 43, 34, 45, 87, 67, 77]
>>> |
```

PRACTICAL 8


Aim: - Write a program for inorder, postorder and preorder traversal of tree.

Theory: - Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.

Depth First Traversals:

- (a) Inorder (Left, Root, Right)
- (b) Preorder (Root, Left, Right)
- (c) Postorder (Left, Right, Root)

Input: -

 practical8.py - C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\practical8.py (3.7.4)

File Edit Format Run Options Window Help

```
#MITALI VISHWEKAR SYCS 4020
import random
random.seed(23)
class Node:
    def __init__(self, val):
        self.val = val
        self.leftChild = None
        self.rightChild = None
def insert(root, key):
    if root is None:
        return Node(key)
    else:
        if root.val == key:
            return root
        elif root.val < key:
            root.rightChild = insert(root.rightChild, key)
        else:
            root.leftChild = insert(root.leftChild, key)
    return root
def PrintInorder(root):
    if root:
        PrintInorder(root.leftChild)
        print(root.val, end=" ")
        PrintInorder(root.rightChild)
def printPreorder(root):
    if root:
        print(root.val, end=" ")
        printPreorder(root.leftChild)
        printPreorder(root.rightChild)
def printPostorder(root):
    if root:
        printPostorder(root.leftChild)
        printPostorder(root.rightChild)
        print(root.val, end=" ")
tree = Node(20)
for i in range(10):
    insert(tree, random.randint(2, 100))
if __name__ == "__main__":
    print("inorder")
    PrintInorder(tree)
    print("\n")
    print("preorder")
    printPreorder(tree)
    print("\n")
    print("postorder")
    printPostorder(tree)
```

Output: -

```
>>>
RESTART: C:\Users\Mitali Vishwekar\Desktop\Mitali_FYCS_20\DS CODES\practical8.py
inorder
4 12 18 20 39 41 47 50 56 69 77

preorder
20 12 4 18 39 77 41 56 50 47 69

postorder
4 18 12 47 50 69 56 41 77 39 20
>>> |
```