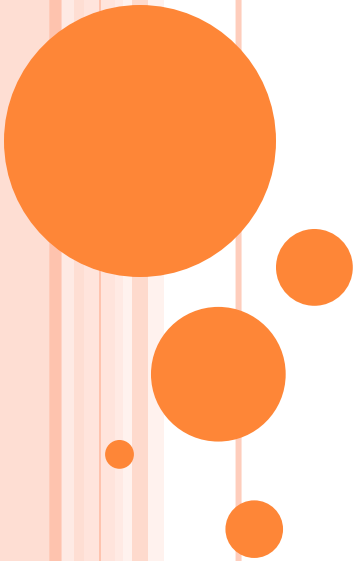


DATA STRUCTURE

Group of data and well defined operations

WHAT IS DATA STRUCTURE?

- Is a named group of data of different data types which is stored in a specific way and can be processed as a single unit. A data structure has well-defined operations, behaviour and properties.

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

DATA TYPE VS. DATA STRUCTURE

- DATA TYPE defines the type of values we can store and operations we can perform. For example in int data type we cannot store decimal values, we cannot multiply two string type values. It also specifies amount of memory it will take.
- DATA STRUCTURE is physical implementation that clearly defines a way of storing, accessing and manipulation of data stored in data structure. Every data structure has a specific way of insertion and deletion like STACK work on LIFO i.e. all operation will take from one end i.e. TOP where as QUEUE works on FIFO i.e. item inserted first will be removed first and new item will always added to the end of QUEUE.
- **In python we will use LIST as a data structure.**

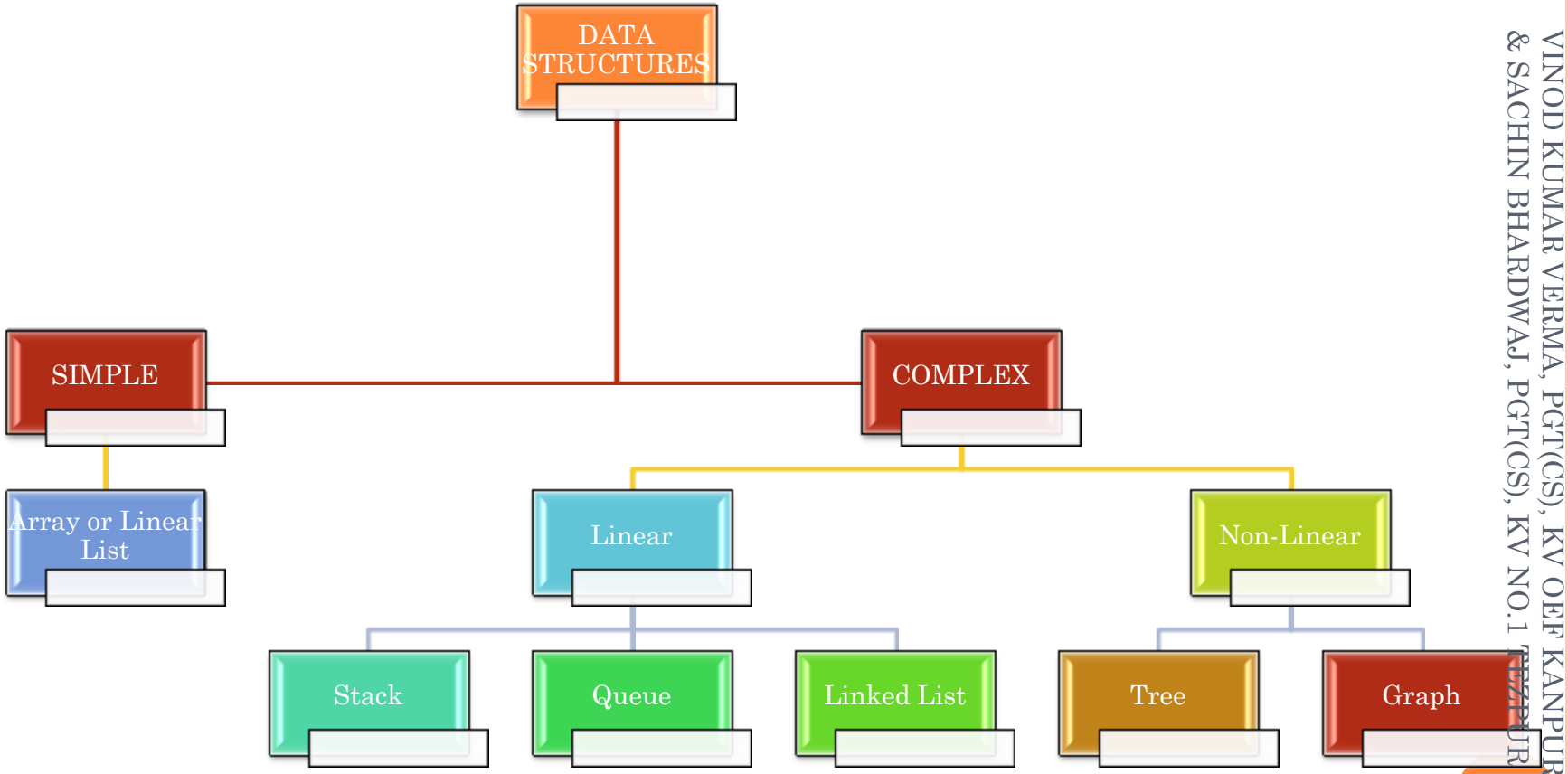
VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

TYPES OF DATA STRUCTURE

- Data structure can be classified into following two types:
  - Simple Data structure : these are built from primitive data types like integer, float, string or Boolean. Example: Linear List or Array
  - Complex Data structure : simple data structure can be combined in various way to form more complex data structure. It is of two types:
    - Linear : are single level data structure i.e. in linear fashion to form a sequence. Example : STACK, QUEUE, LINKED LIST
    - Non-Linear: are multilevel data structure. Example: TREE and GRAPH.

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

DIAGRAM TO REPRESENT ALL DATA STRUCTURE



VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

LINEAR LIST ARRAYS

- Refers to named list of finite number of n similar data elements. Each of the data elements can be accessed by its unique index/subscript position usually 0,1,2,3,…
- For example if the list mylist contains 10 elements then first element will be mylist[0], second will be mylist[1] and so on.
- Array can be Single Dimensional or Multidimensional. In Python arrays are implemented through List data types as Linear List or through NumPy arrays.

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

STACK

- Allow to insert and delete items from one end only called TOP.
- Stack works on LIFO principle. It means items added in list will be removed first.
- Real life Example: Stack of Plates, Books, Bangles
- Computer based examples: Undo Operation, Function Call etc.

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

QUEUE

- Allows insertion and deletion from different end i.e. insertion from REAR and deletion from FRONT
- Queue works on FIFO principle i.e. item added first will be removed first.
- Real life example: Queue of People for ticket
- Computer based example: Keyboard typing, Printer etc.

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

LINKED LIST

- It is dynamic data structure i.e. size of Linked List is not pre-known before the execution of program. It takes memory as per requirement during runtime and every item allocated dynamically will be known as NODE.
- NODE contains 2 parts: (1) INFO part stores the actual data to store like roll no. name, marks etc. and (2) LINK holds the address of next allocated node in memory creating chain of linked items
- Singly Linked List hold the address of next node and last Node will point to NULL whereas Doubly Linked List holds the address of next as well as previous node allows both forward and backward movement.

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

TREES

- Are multilevel data structure having hierarchical relationship among its elements called nodes.
- Topmost node is called the root of tree and bottommost nodes are called leaves of tree.
- Each of the node holds the address of nodes below it.
- It will never contains a closed loop

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

OPERATIONS ON DATA STRUCTURE

OPERATIONS	DESCRIPTION
INSERTION	Addition of new data to data structure
DELETION	Removal of data element from data structure
SEARCHING	Finding specific data element in data structure
TRAVERSAL	Processing all data elements one by one
SORTING	Arranging data elements in ascending or descending order
MERGING	Combining elements of two similar data structure to form a new data structure.

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

LINEAR LIST

- A linear data structure forms a sequence.
- When elements of linear structure are homogenous and are represented in memory b means of sequential memory location are called arrays.
- An Array stores a list of finite number(s) of homogenous data elements.
- When upper bound and lower bound of linear list are given, its size can be calculated as :
  - Size of List = Upper Bound - Lower Bound + 1

Index	0	1	2	3	4	5	6
values	10	20	30	40	55	70	45

  - Size of above list = 6 – 0 + 1 = 7

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

SEARCHING : LINEAR AND BINARY SEARCHING

- Linear Searching: each element of linear list is compared with the given item to be searched one by one. This method traverse the linear list sequentially to located the given item.
- For Linear Searching item need not be in any order, it can be performed on random list.
- Linear Searching is suitable for small list only.
- Best Case: when item to be search is at first position
- Worst Case: when item to be search is at last position or not present
- Average Case: when item is somewhere in list other than first or last position.

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

LINEAR SEARCH FUNCTION

```
def LSearch(Arr,Item):
        for i in range(len(Arr)):
                if Item == Arr[i]:
                        return True
        return False
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

COMPLETE CODE: LINEAR SEARCH

```
def linearSearch(mylist,item):
        n = len(mylist)
        for i in range(n):
                if item == mylist[i]:
                        return i
        return None
```

```
Enter how many items in List 5
Enter value :10
Enter value :5
Enter value :11
Enter value :20
Enter value :8
Enter Value to Search :20
Found at position  4
```

```
Arr=[]
n = int(input("Enter how many items in List "))
for i in range(n):
        d = int(input("Enter value :"))
        Arr.append(d)
key = int(input("Enter Value to Search :"))

pos = linearSearch(Arr,key)
if pos!=None:
        print("Found at position ",pos+1)
else:
        print("Not Found")
```

```
Enter how many items in List 6
Enter value :1
Enter value :2
Enter value :33
Enter value :44
Enter value :11
Enter value :88
Enter Value to Search :100
Not Found
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

BINARY SEARCHING : *DIVIDE AND RULE*

- **Data must be SORTED either in ascending or descending order to search using binary method**
- **Searching Starts from the middle position. To find middle position**
  - **Mid = (lower_index + higher_index) / 2**
- **If middle value is equal to search element "SEARCH SUCCESSFUL"**
- **If middle value if larger than search value, searching continues towards left of middle position otherwise towards right side of middle position. [for data arranged in ascending order else vice versa]**
- **Takes less time compare to Linear searching**
- **Suitable for LARGE list**
- ***Lets Search…***

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

BINARY SEARCH FUNCTION

```
def BSearch(mylist,item):
        low = 0
        high = len(mylist)-1
        while(low<=high):
                mid = (low+high)//2
                if mylist[mid]==item:
                        return mid
                elif mylist[mid]>item:
                        high = mid - 1
                else:
                        low = mid + 1
        return -1
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

Complete Code: Binary Searching

```
def BSearch(mylist,item):
        low=0
        high=len(mylist)-1
        while low<=high:
                mid = (low+high)//2
                if mylist[mid]==item:
                        return mid
                elif mylist[mid]>item:
                        high=mid-1
                else:
                        low=mid+1
        return -1
```

```
Arr=[]
n=int(input("Enter How many items :"))
print("## Enter Values in Ascending Order ##")
for i in range(n):
        val=int(input("Enter Value :"))
        Arr.append(val)
key=int(input("Enter Item to Search :"))
pos=BSearch(Arr,key)
if pos==-1:
        print("Search Item not found")
else:
        print("Item found @ position :",pos+1)
```

```
Enter How many items :8
## Enter Values in Ascending Order ##
Enter Value :4
Enter Value :8
Enter Value :12
Enter Value :15
Enter Value :21
Enter Value :30
Enter Value :45
Enter Value :99
Enter Item to Search :15
Item found @ position : 4
```

```
Enter How many items :5
## Enter Values in Ascending Order ##
Enter Value :20
Enter Value :30
Enter Value :40
Enter Value :50
Enter Value :60
Enter Item to Search :61
Search Item not found
```

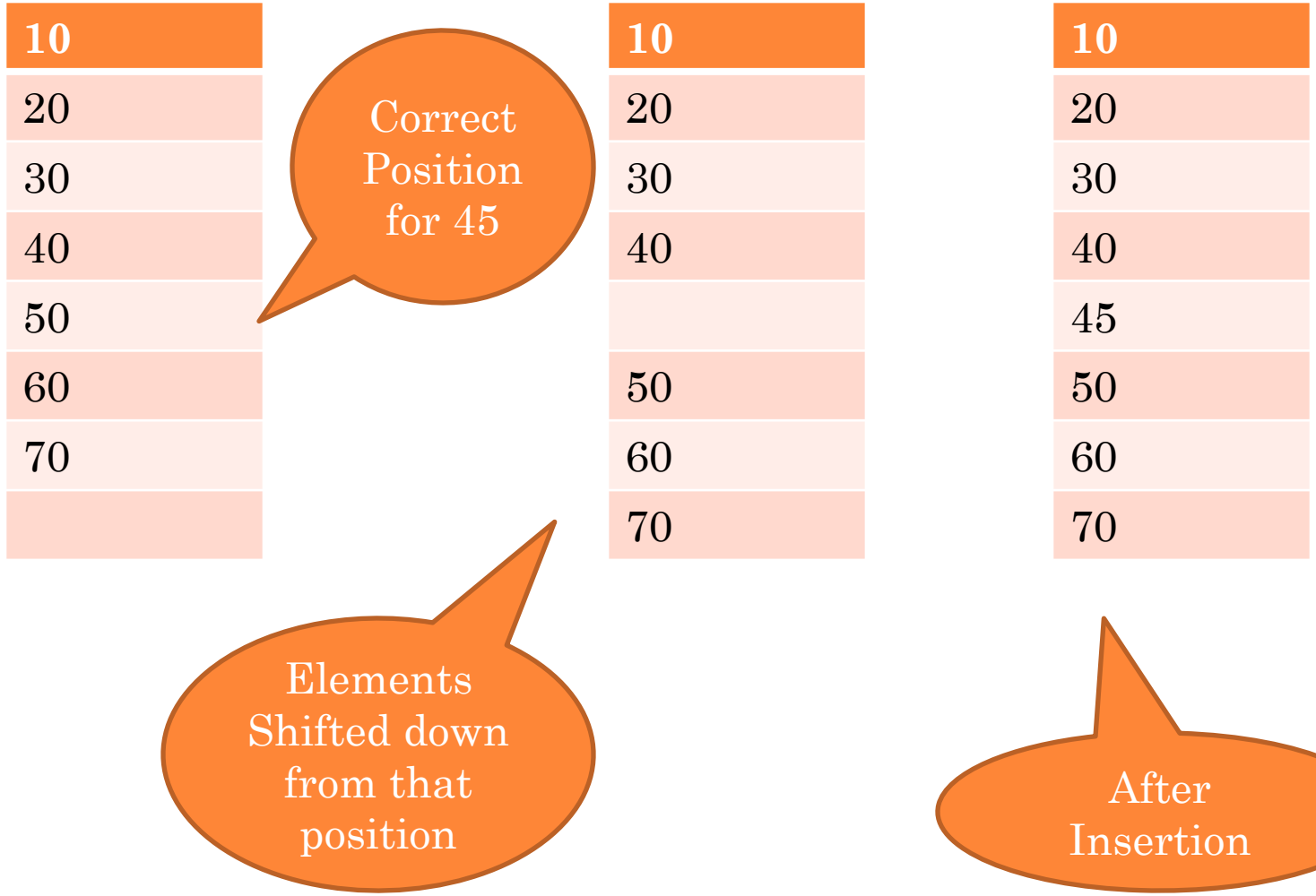
VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

INSERTION IN LINEAR LIST

- Insertion of new item can be done by 2 ways:
  - In unsorted list we can add item directly at the end of list
  - For sorted array, we have to first find the position where new items is to be added, then shift all the elements from that position one place down and create place for new item and then insert the new item to that place.
  - Let us take an example…

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

Insertion in Sorted Array: insert 45



VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

INSERTION USING PYTHON APPROACH

- As we can observe Insertion of new item in sorted array requires shifting of elements to make room and this is very time consuming when list grows.
- It should be avoided using better approach like bisect, available in bisect module.
  - **bisect.insort(list,newItem)**
- The insort() function of bisect module inserts an item in the sorted array, keeping it sorted.
- Bisect module offers another function bisect() which return the appropriate index where new item can be inserted to keep the order at it is.
  - **bisect.bisect(list,element)**
- ***Note: bisect module works only on sequence arranged in Ascending Order only.***

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

PROGRAM: USING BISECT MODULE

```
import bisect
marks=[10,20,30,40,50,60,70,80]
print("\nCurrent list is :")
print(marks)
Item = int(input("Enter item to insert:"))
pos = bisect.bisect(marks,Item)
bisect.insort(marks,Item)
print("\nItem Inserted at Index :",pos)
print("\nUpdated List is :")
print(marks)
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

BISECT MODULE

- As we know bisect module works only on Ascending order, hence to make it for descending order:
  - ***First reverse the list arranged in descending in ascending order using reverse()***
  - ***Perform the insert operation***
  - ***Again reverse the list.***

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

DELETION FROM SORTED LIST: TRADITIONAL APPROACH

- To Delete item from sorted list we have to first find the element position in List using Binary Searching
- Delete the item from list if search successful
- Shift all the items upwards to keep the order of list undisturbed
- Reduce the List size by 1

- *However in Python*, we have to just delete the item and rest of the work is automatically handled by Python i.e. Shifting, reducing size etc.
- In Python we will use either **remove(), del or pop()** for this purpose

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

DELETION OF AN ELEMENT FROM LINEAR LIST : SORTED

```
def BSearch(mylist,item):
    low = 0
    high = len(mylist)-1
    while(low<=high):
        mid = (low+high)//2
        if mylist[mid]==item:
            return mid
        elif mylist[mid]>item:
            high = mid - 1
        else:
            low = mid + 1
    return -1

marks = [10,20,30,40,50,60]
print("\nCurrent List :")
print(marks)
ans ="y"
while ans=="y":
    Item = int(input("Enter item to delete :"))
    pos = BSearch(marks,Item)
    if pos!=-1:
        del marks[pos]
        print("\nList after deletion :")
        print(marks)
    else:
        print("Sorry Item not found :")
    ans = input("delete more ?")
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

ANOTHER METHOD – SIMPLE AND SHORT

```
Items=[10,20,30,40,50,60,70]
print("List is :",Items)
ans="y"
while ans=="y":
        item = int(input("Enter Item to Delete :"))
        try:
                pos=Items.index(item)
                Items.pop(pos)
                print("Item Deleted")
                print("List is :",Items)
        except:
                print("Item not in the list")
        ans=input("Delete More ?(y/n)")
```

```
List is : [10, 20, 30, 40, 50, 60, 70]
Enter Item to Delete :30
Item Deleted
List is : [10, 20, 40, 50, 60, 70]
Delete More ?(y/n)y
Enter Item to Delete :10
Item Deleted
List is : [20, 40, 50, 60, 70]
Delete More ?(y/n)y
Enter Item to Delete :35
Item not in the list
Delete More ?(y/n)n
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

Traversal of Linear List

- It involves processing of all elements one by one like:
  - Printing of all items
  - Searching item one by one
  - Double each value of list etc.
- In nutshell, if we are accessing all elements one by one for any reason, it is traversing.

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

EXAMPLE – 1 (PROGRAM TO DOUBLE THE LIST ELEMENT)

```
def DoubleIt(Arr):
        for i in range(len(Arr)):  #traversing all elements
                Arr[i] *= 2

Arr=[10,20,30,40,50]
print("Current List :")
for i in Arr:                          #traversing all elements
        print(i)

DoubleIt(Arr)
print("After Update List :")
for i in Arr:
        print(i)
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

Few functions: Traversal (function to find sum of every alternate elements)

```
#Function to add alternate elements

def AddAlternate(Arr):
        sum=0
        for i in range(len(Arr)):
            if i % 2 == 0:
                sum+=Arr[i]
        return sum
```

```
#Function to add element ending with digit 7

def AddEnding7(Arr):
        sum=0
        for i in range(len(Arr)):
            if Arr[i] % 10 == 7:
                sum+=Arr[i]
        return sum
```

```
#Function to count how many even values in list

def CountEven(Arr):
        count=0
        for i in range(len(Arr)):
            if Arr[i] % 2 == 0:
                count+=1
        return count
```

```
#Function to swap adjacent elements

def SwapAdjacent(Arr):
    for i in range(0,len(Arr),2):
        Arr[i],Arr[i+1]=Arr[i+1],Arr[i]
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

Sorting a Linear List

- Sorting means arranging the list items in Ascending or Descending order.
- Many Sorting algorithm which we can use for this purpose like: Bubble, Selection, Insertion, Shell, Quick, Heap, Merge etc.
- In Class XI, we have already studied about 2 sorting methods: Bubble and Insertion (PDF available in Class XI option of website)
- Bubble Sorting
- Insertion Sorting
- Comparison between Bubble and Insertions

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

List Comprehensions

- We have already studied how to create a list i.e. either by assigning comma separated values in square bracket to list variable or by using append() in loop.

  e.g. mylist = [4,8,12,16,20] Or by loop
- There is another method of creating list called **list comprehensions.**
- A list comprehension is a shorthand for list creating for loop in the form of single statement.
- Example 1: to create list using list comprehension:

```
mylist = [i*4 for i in range(1,6)]
print(mylist)
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

EXAMPLE - 2

```
mylist = [i for i in range(1,101) if i%2==0]
print(mylist)
```

```
mylist=[]
for i in range(1,101):
        if i % 2 == 0:
                mylist.append(i)
```

EXAMPLE - 3

```
lst2 = [i if i%2==0 else '@' for i in range(1,101) ]
print(lst2)
```

EXAMPLE – 4 : FOR CLARITY

```
lst2 = [(i if i%2==0 else '@') for i in range(1,101) ]
print(lst2)
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

LIST COMPREHENSION FOR NESTED FOR LOOP

Example: Nested Loop

```
Arr = [ ]
for i in [10,20,30]:
        for j in [5,10,15]:
                Arr.append(i+j)
print(Arr)
```

Example : using List Comprehension

```
Arr = [ i + j for i in [10,20,30] for j in [5,10,15]]
print(Arr)
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

Advantages of List Comprehensions

- Code reduction : A code of 3 or more lines gets reduced to single line.
- Faster code procession : list comprehensions are executed faster than their equivalent for loops because:
  - Python will allocate the list's memory before adding the elements to it, instead of having to resize on runtime
  - Also call to append() function gets avoided thus reducing the function overheads.

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

Nested/Two Dimensional List In Python

- In Class XI, we have already studied Nested List that a List can have another list as a value. For e.g.

```
List1 = [1,2,3]
List2 = [10,20,List1]
print(List2)
```

Output: [10,20,[1,2,3]]

List2

[0]	[1]	[2][0]	[2][1]	[2][2]
10	20	1	2	3

```
print(List2)          # [10,20,[1,2,3]]
print(List2[0])       # 10
print(List2[2][0])    # 1
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

Two Dimensional List

- Is a list having all elements as list of same shape for e.g.
  - Mylist=[[20,40,60],[5,10,15],[9,18,27]]

20	40	60
5	10	15
9	18	27

  - It is a 2 – D List with 3 rows and 3 columns
  - First value will be at index [0][0] and last will be [2][2]

```
mylist = [[1,2,3],[4,5,6],[7,8,9]]
print(len(mylist))          # no. of rows
print(len(mylist[0]))       # no. of columns
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

Two Dimensional List

- Ragged List: sometimes list contains another list of different shapes for e.g.

  mylist = [[20,40,60],[4,5]]

  Here first row contains 3 columns and 2 row contains 2 columns

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

CREATING AND PRINTING OF 2-D LIST

```
print("\nWelcome to 2-D List Management System")
r = int(input("Enter How many rows :"))
c = int(input("Enter how many columns "))
Mat=[]
for i in range(r):
    R = []
    for j in range(c):
        val = int(input("Enter value to store at index ["+str(i)+","+str(j)+"]"))
        R.append(val)
    Mat.append(R)
print("*** Your Matrix is ***")
for i in range(r):
    for j in range(c):
        print(Mat[i][j],"\t",end=' ')
    print()
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

Slice of 2-d List

```
mylist=[[20,30,40],[21,22,23],[100,150,200],[19,21,40]]
print(mylist[:2])
print(mylist[2:])
print(mylist[2:][:1])
```

```
[[20, 30, 40], [21, 22, 23]]
[[100, 150, 200], [19, 21, 40]]
[[100, 150, 200]]
```

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR
& SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR