CHAPTER - IX
DATA STRUCTURES LINEAR LISTS

INTRODUCTION

INTRODUCTION

What is Data Structure?

In computer science, a data structure is a particular way of organizing data in a computer so that it can be used

Data structures provide a means to manage large amounts of data efficiently for uses such as large databases and internet indexing services. Usually, efficient data structures are key to designing efficient algorithms.

DATA STRUCTURE OPERATIONS

Data are processed by means of certain operations which appearing in the data structure. Following are the most common types of Data structure operations.

1. TRAVERSING
2. SEARCHING
3. INSERTING
4. DELETING
5. SORTING
6. MERGING

DATA STRUCTURE OPERATIONS

1. TRAVERSING

Accessing each records exactly once so that certain items in the record may be processed.

2. SEARCHING

Finding the location of a particular record with a given key value, or finding the location of all records which satisfy one or more conditions.

DATA STRUCTURE OPERATIONS

3. INSERTING

Adding a new record to the structure.

4. DELETING

Removing the record from the structure.

DATA STRUCTURE OPERATIONS

5. SORTING

Managing the data or record in some logical order(Ascending or descending order).

6. MERGING

Combining the record in two different sorted files into a single sorted file.

ELEMENTRAY DATA REPRESENTATION

ELEMENTRAY DATA REPRESENTATION

Data can be in two forms:

1. RAW DATA

2. DATA ITEM

1. RAW DATA

These are facts. In simplest form these are values.

2. DATA ITEM

These represent single unit of values of certain type.

DATA TYPE VS DATA STRUCTURES

DATA TYPE VS DATA STRUCTURES

DATA TYPE refers to the type i.e integer or float or character or string for the data which is represented in the memory location of the computer system.

DATA STRUCTURE refers to the physical implementation or arrangement of data in the memory I other words it defines a way of storing, accessing,manipulating od data stored in the data structure.

DIFFERENT DATA STRUCTURES

Data structures are broadly classified in to two types.

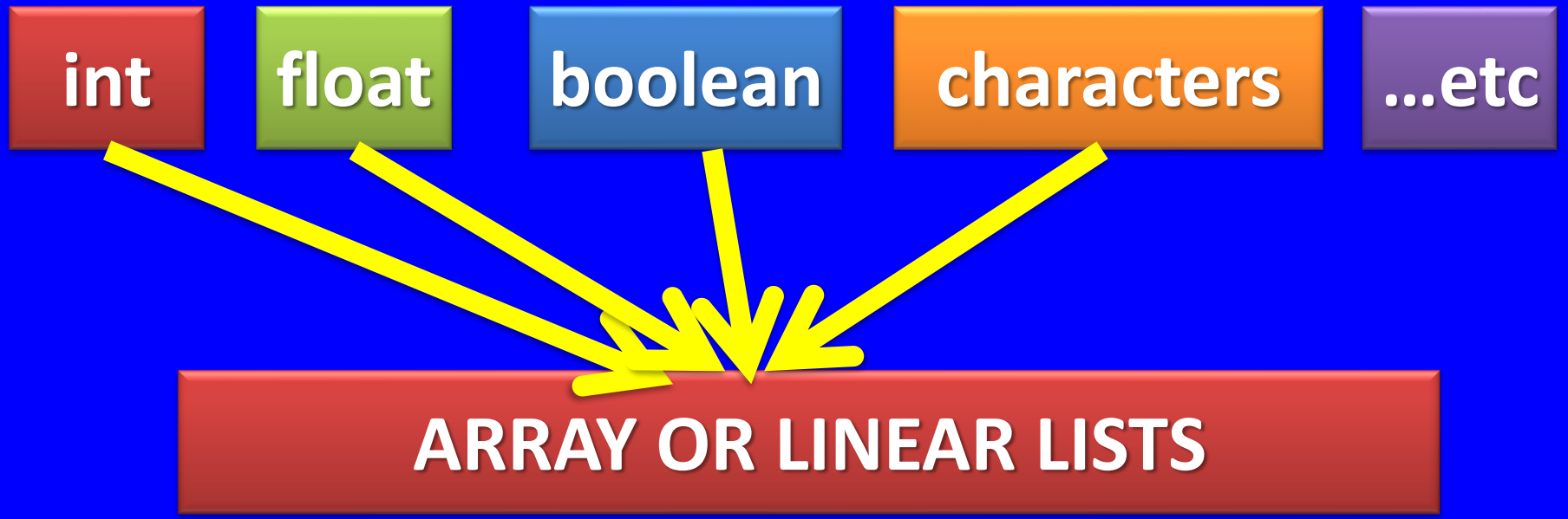
1. SIMPLE DATA STRUCTURE

2. COMPOUND DATA STRUCTURE

1. SIMPLE DATA STRUCTURE

1. SIMPLE DATA STRUCTURE

Simple Data structures are also called as primitive data structures and are built from basic data types viz int, float, boolean, characters are called simple data structures.



ARRAY or LINEAR LISTS

A linear array is a list of finite numbers of elements stored in the memory. In a linear array, we can store only homogeneous data elements. Elements of the array form a sequence or linear list that can have the same type of data. Each element of the array is referred by an index set.

Note: Arrays can be implemented through LIST or through NumPy arrays.

2. COMPOUND DATA STRUCTURE

2. COMPOUND DATA STRUCTURE

Compound Data Structure are complex in nature and are classified in to two types

1. LINEAR DATA STRUCTURES

2. NON LINEAR DATA STRUCTURES

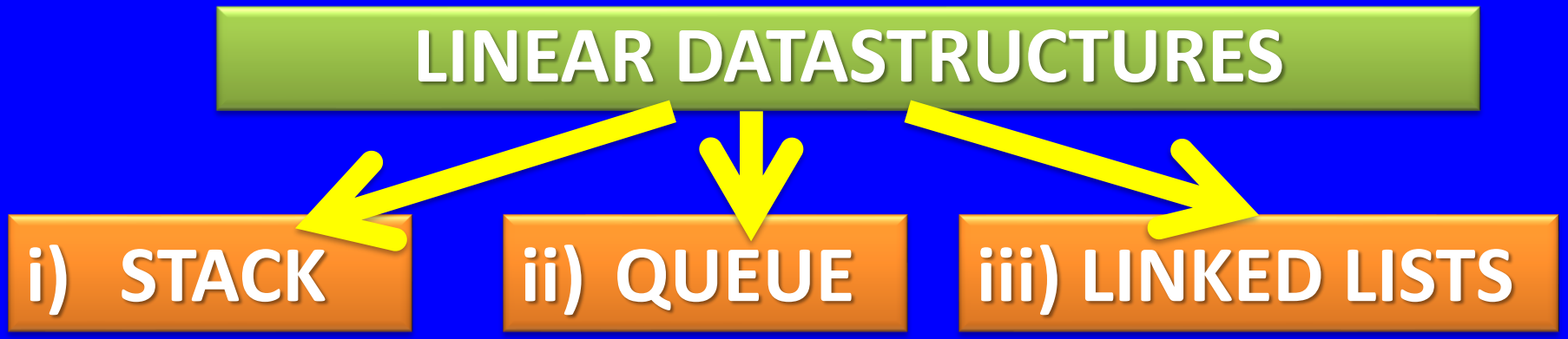
1. LINEAR DATASTRUCTURES

1. LINEAR DATASTRUCTURES

What is Linear Data Structure?

A data structure is said to be linear data structure if its elements form a sequence.

meaning sequential arrangement of data is the linear data structure. For Example

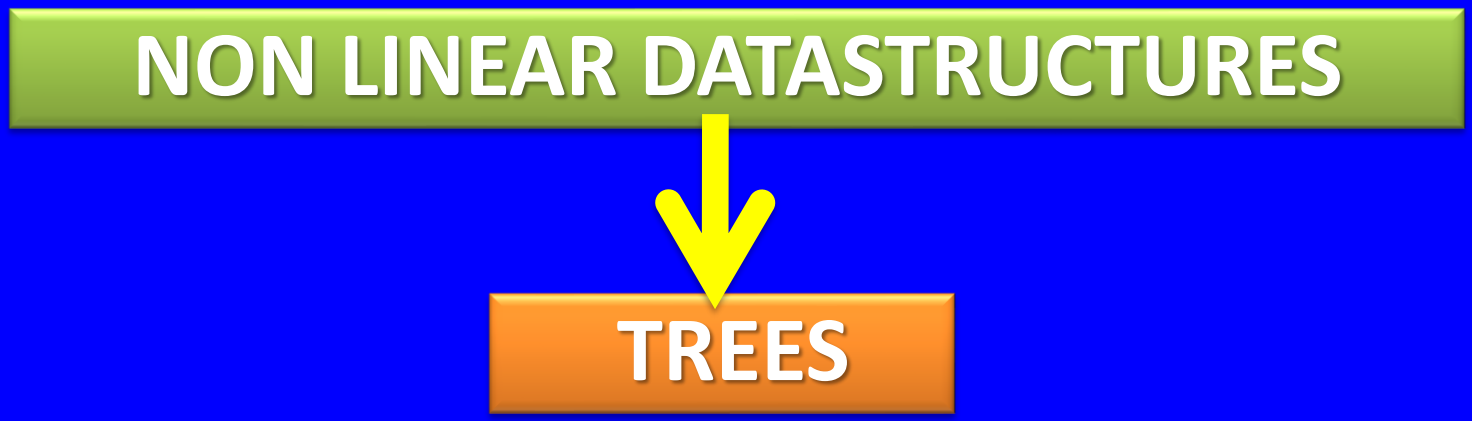


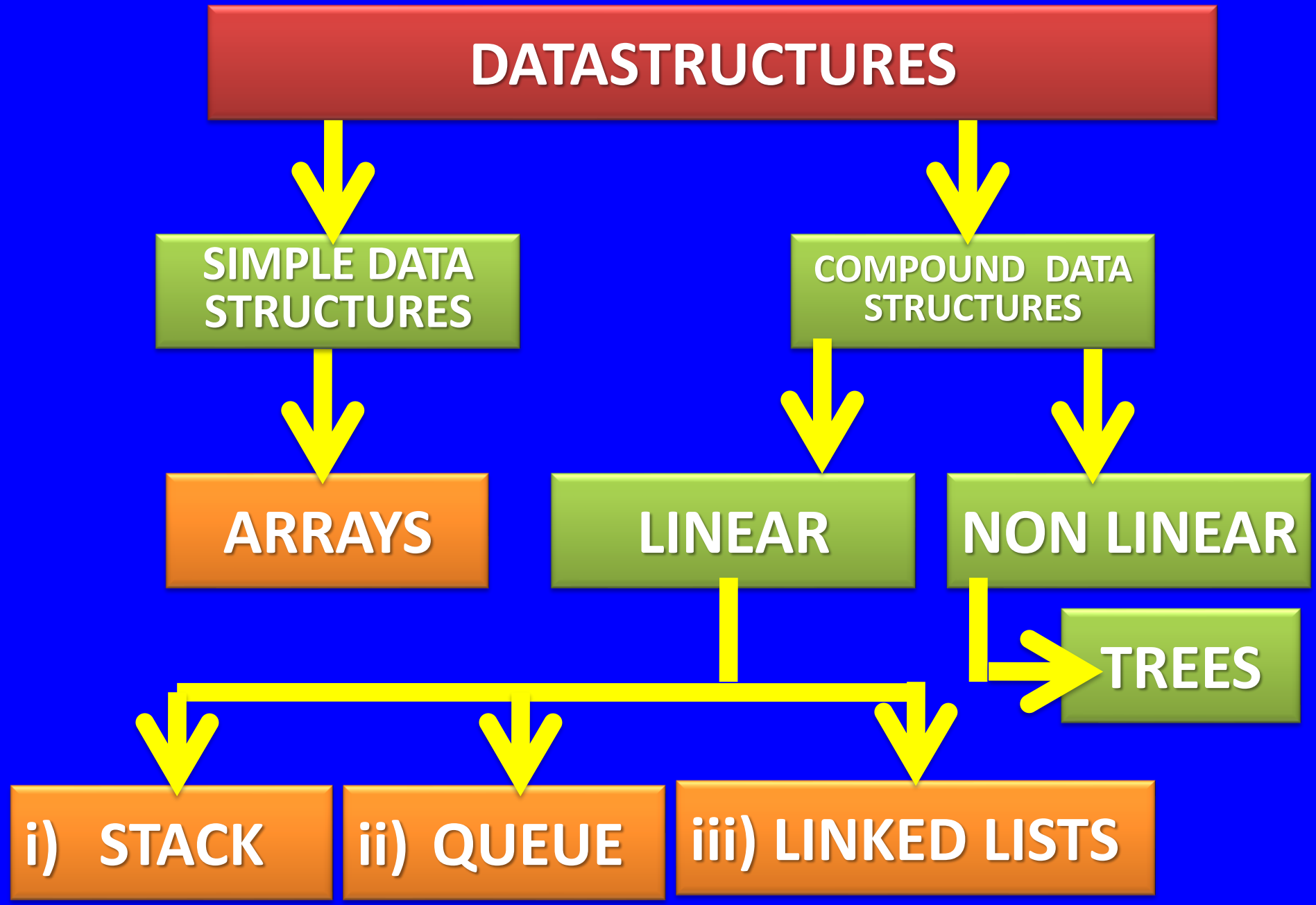
NON LINEAR DATASTRUCTURES

2. NON LINEAR DATASTRUCTURES

What is Non Linear Data Structure?

A non-linear data structure is a data structure in which a data item is connected to several other data items. So that a given data item has the possibility to reach one-or-more data items.




DATASTRUCTURES
SIMPLE DATA STRUCTURES
COMPOUND DATA STRUCTURES
ARRAYS
LINEAR
NON LINEAR
TREES
i) STACK
ii) QUEUE
iii) LINKED LISTS

STACKS

STACKS

What is STACK?

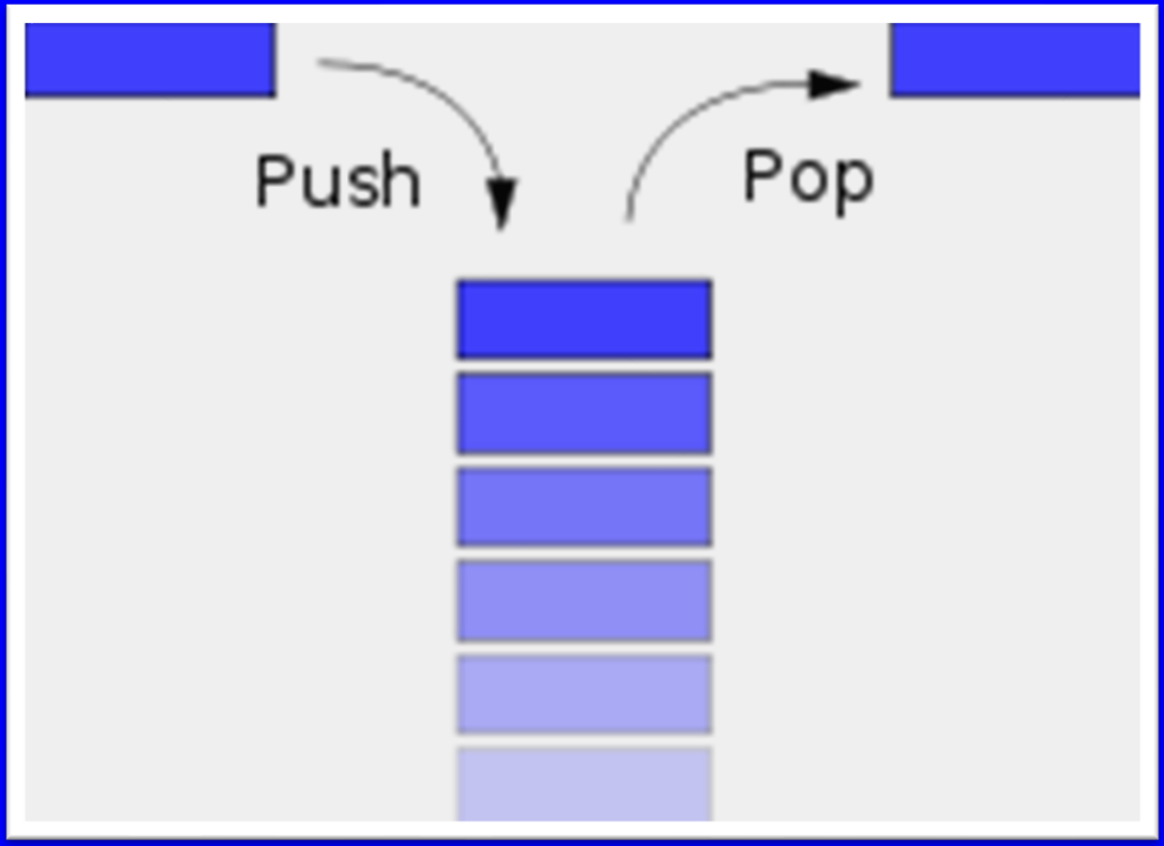
In computer science, a stack is a last in, first out (LIFO) data structure. A stack is characterized by only two fundamental operations and they are:

1. PUSH
2. POP

The **PUSH** operation adds an item to the top of the stack.

The **POP** operation removes an item from the top of the stack.

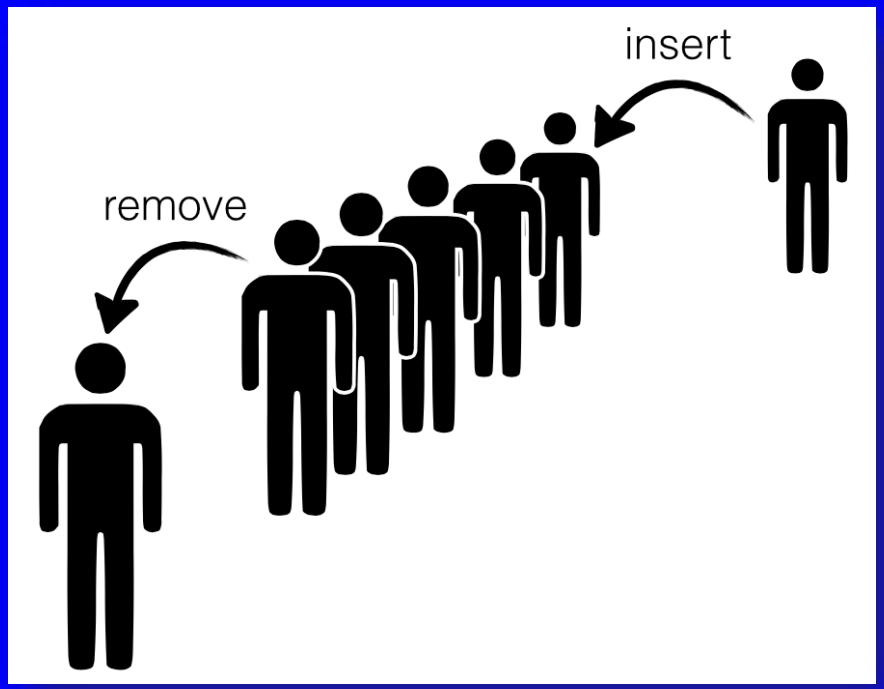
STACKS



QUEUE

QUEUE

What is Queue?



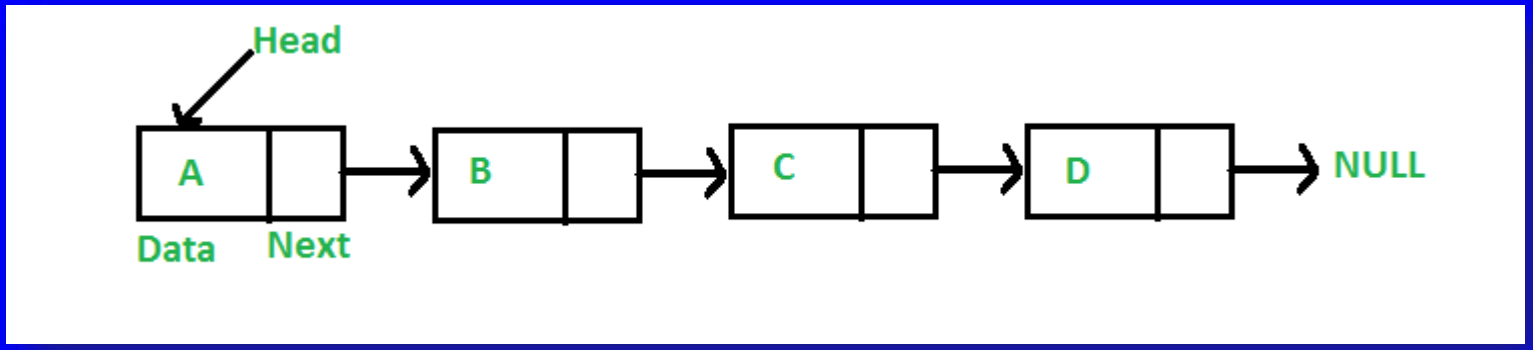
Queue is a linear data structure which follows First In First Out (FIFO) rule in which a new item is added at the rear end and deletion of item is from the front end of the queue. In a FIFO data structure, the first element added to the queue will be the first one to be removed.

LINKED LISTS

LINKED LISTS

What is Linked List?

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:



TREES

TREES

What is Tree?

Tree is a non-linear data structure which organizes data in a hierarchical structure and this is a recursive definition.

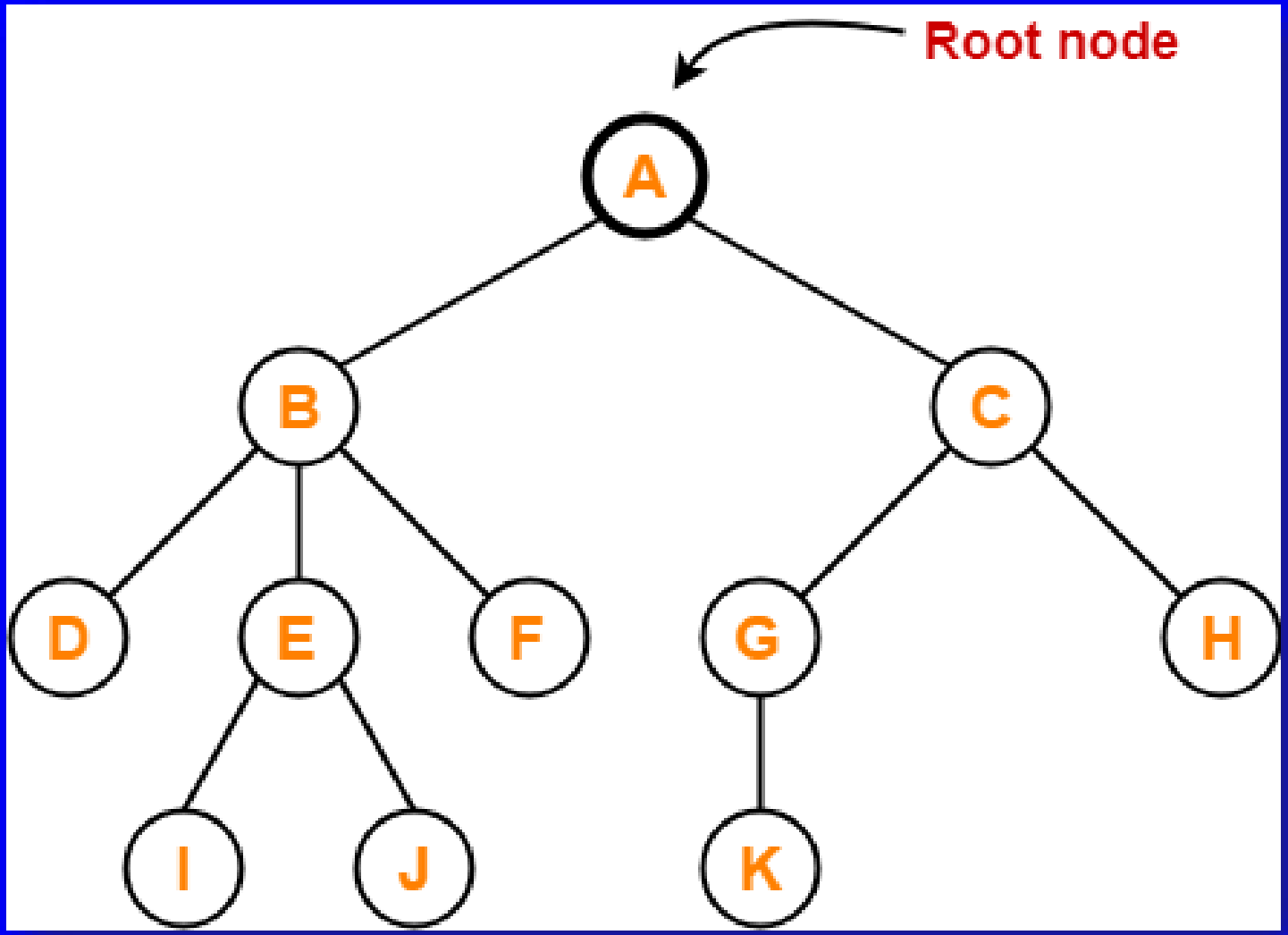
OR

A tree is a connected graph without any circuits.

OR

If in a graph, there is one and only one path between every pair of vertices, then graph is called as a tree.

TREES



SEARCHING

SEARCHING

What is Searching?

Searching is an operation or a technique that helps finds the place of a given element or value in the list.

There are various algorithms , most common searching algorithms are:-

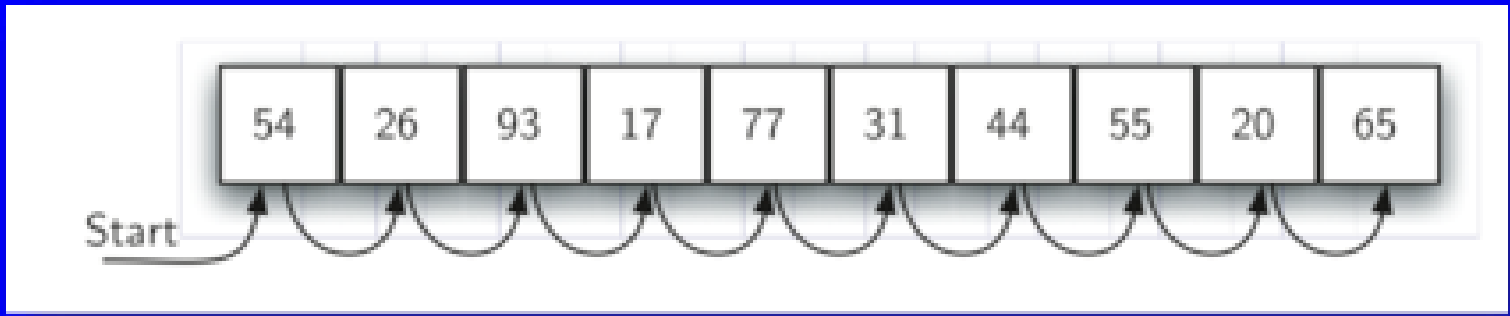
1. LENEAR SEARCH OR SEQUENTIAL SEARCH
2. BINARY SEARCH

1. LINEAR SEARCH OR SEQUENTIAL SEARCH

1. LINEAR SEARCH OR SEQUENTIAL SEARCH

What is Linear Search or Sequential Search?

In computer science, a **linear search or sequential** search is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.



1. LINEAR SEARCH OR SEQUENTIAL SEARCH

WAPP to search an element in a given list using Linear Search Method using NumPy

```
import numpy as np
def Linear_Search():
    L1=[9,32,67,72,899,390,879]
    a=np.array(L1)
    Search_Val=int(input("Enter the Searching Element"))
    found=0
    pos=1
```

1. LINEAR SEARCH OR SEQUENTIAL SEARCH

WAPP to search an element in a given list using Linear Search Method using NumPy

```
for i in a:
        if i==Search_Val:
            found=1
            break
        pos= pos+1
    if found:
        print("Element is found at location ",pos)
    else:
        print("Element is not found!")
Linear_Search()
```

1. LINEAR SEARCH OR SEQUENTIAL SEARCH

WAPP to search an element in a given list using Linear Search Method

```
def Sequential_Search(dlist, item):
        pos = 0 found = False
        while pos < len(dlist) and not found:
                if dlist[pos] == item:
                        found = True
                else:   pos = pos + 1
        return found, pos
```

1. LINEAR SEARCH OR SEQUENTIAL SEARCH

Linear Search or Sequential Search Analysis

ITEM PRESENT:

Best Case	:	1
Average Case	:	½
Worst Case	:	n

ITME NOT PRESENT:

Best Case	:	n
Average Case	:	n
Worst Case	:	n

2. BINARY SEARCH

2. BINARY SEARCH

What is Binary Search?

In computer science, binary search, also known as half-interval search, logarithmic search, or binary chop, is a search algorithm that finds the position of a target value within a sorted array.

Binary Search. Binary search is a fast search algorithm with run-time complexity of O(log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.

2. BINARY SEARCH

```
def binary_search(alist, key):
    start = 0
    end = len(alist)
    while start < end:
        mid = (start + end)//2
        if alist[mid] > key:
            end = mid
        elif alist[mid] < key:
            start = mid + 1
else:
            return mid
    return -1
```

2. BINARY SEARCH

```
alist = input('Enter the sorted list of numbers: ')
alist = alist.split()
alist = [int(x) for x in alist]
key = int(input('The number to search for: '))

index = binary_search(alist, key)
if index < 0:
    print('{} was not found.'.format(key))
else:
    print('{} was found at index {}.'.format(key,
index))
```

2. BINARY SEARCH

```
alist = input('Enter the sorted list of numbers: ')
alist = alist.split()
alist = [int(x) for x in alist]
key = int(input('The number to search for: '))

index = binary_search(alist, key)
if index < 0:
    print('{} was not found.'.format(key))
else:
    print('{} was found at index {}.'.format(key,
index))
```

INSERTION IN LINEAR LIST

INSERTION IN LINEAR LIST

Insertion of new element in array can be done in two ways:-

1) If the array is unordered it can be inserted at the end of array.
2) If the array is sorted then new element is added at appropriate position without altering the order of the list or array.

INSERTION IN LINEAR LIST

```
# Function to insert element
def insert(list, n):

    # Searching for the position
    for i in range(len(list)):
        if list[i] > n:
            index = i
            break
```

INSERTION IN LINEAR LIST

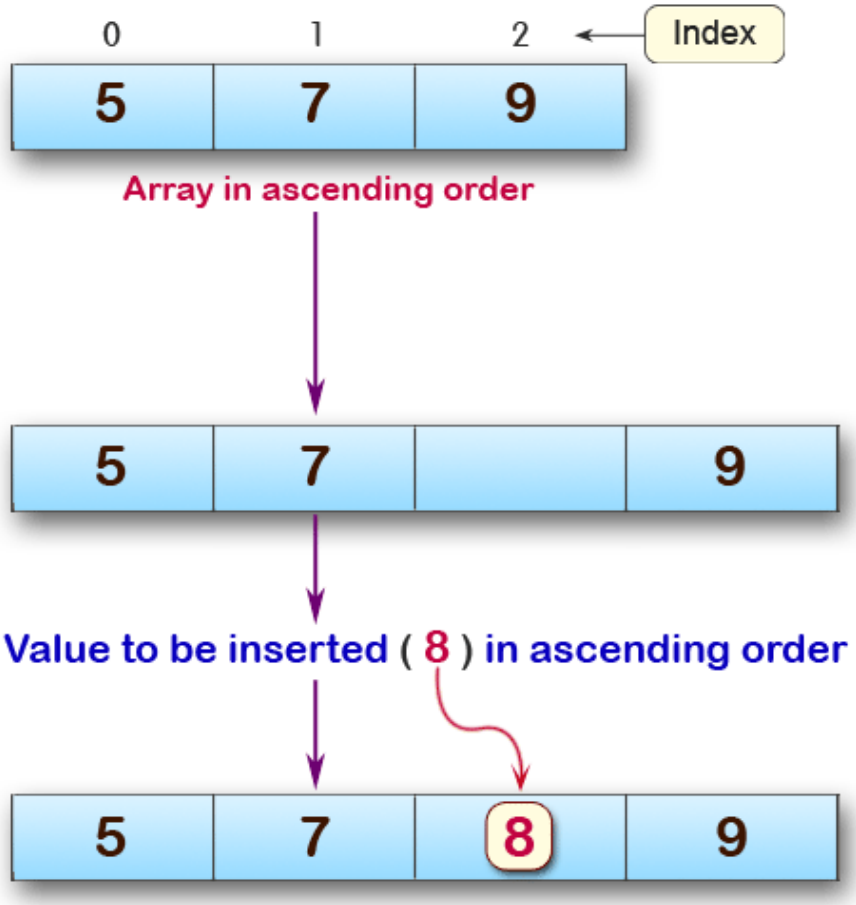
```
# Inserting n in the list
    list = list[:i] + [n] + list[i:]
    return list

# Driver function
list = [1, 2, 4]
n = 3

print(insert(list, n))
```

INSERTING AN ELEMENT IN SORTED ARRAY

INSERTING AN ELEMENT IN SORTED ARRAY



Inserting new element in sorted list requires the searching the right position and shifting the elements. As shown in fig.

bisect MODULE

bisect MODULE

Inserting new element in sorted list requires the searching the right position and shifting the elements. This is the complex procedure.

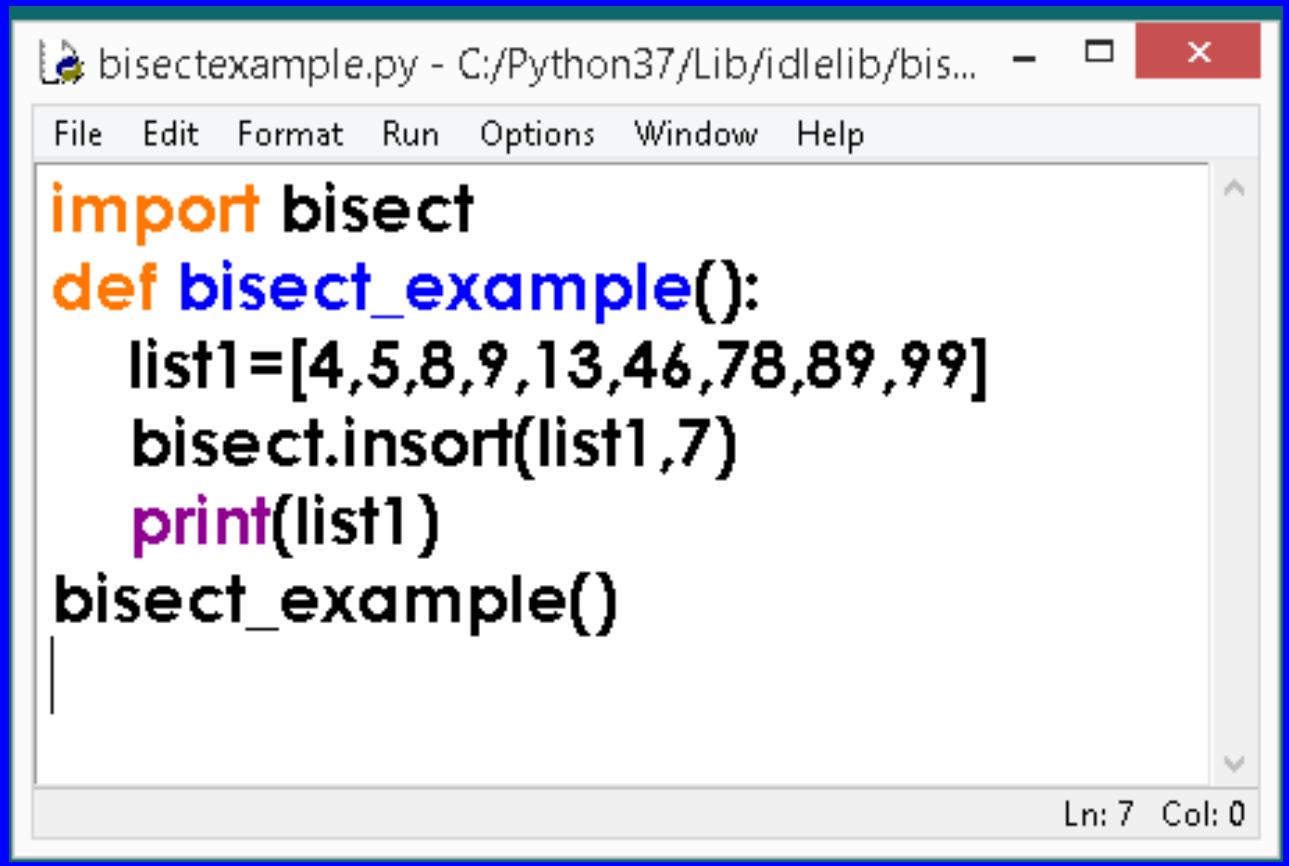
Python provides **bisect algorithm** to solve this problem

bisect MODULE

What is bisect?

Python in its definition provides the **bisect algorithms** using the module **“bisect”** which allows to keep the list in sorted order after insertion of each element.

bisect MODULE



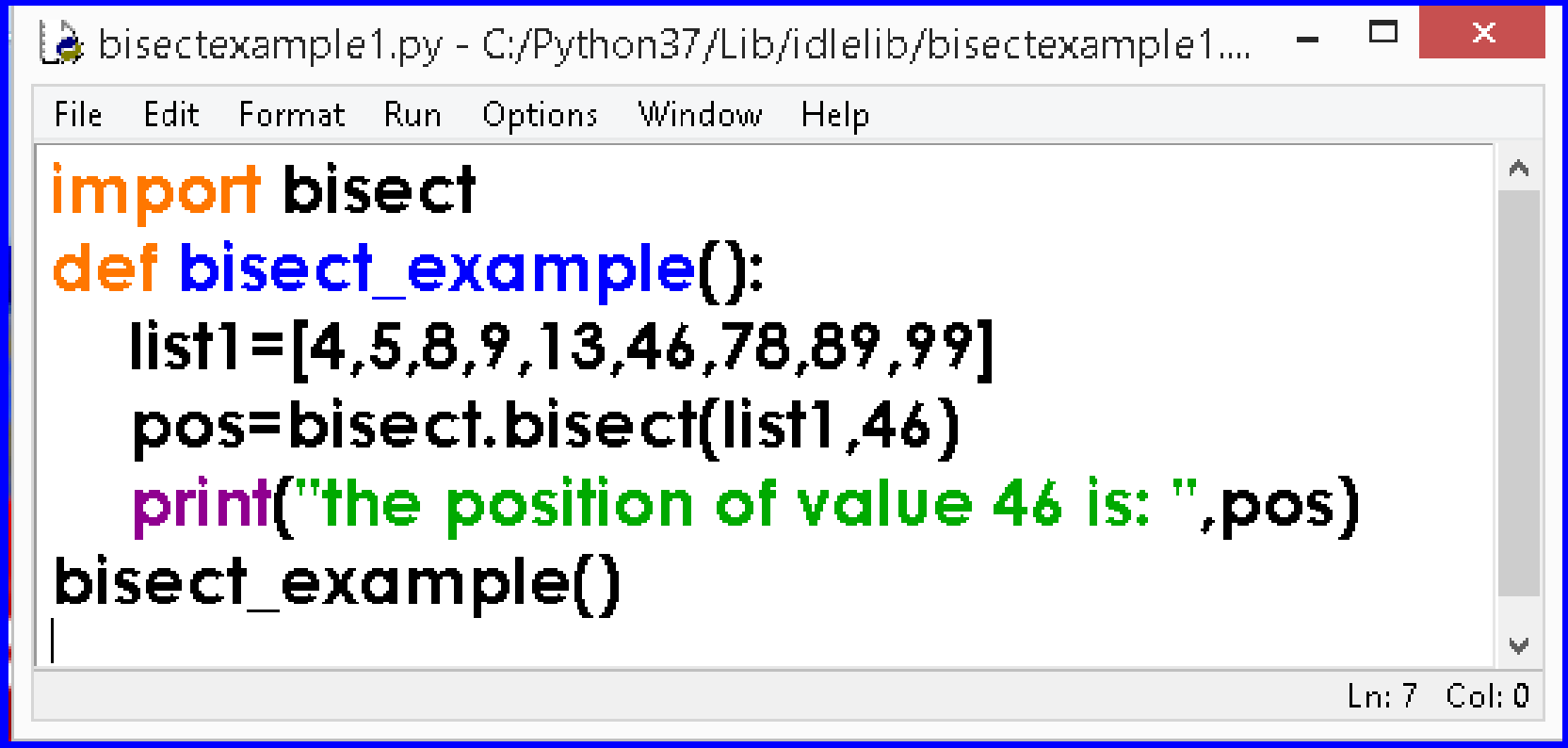
bisect MODULE

To know the position of element in the given list execute the following method.

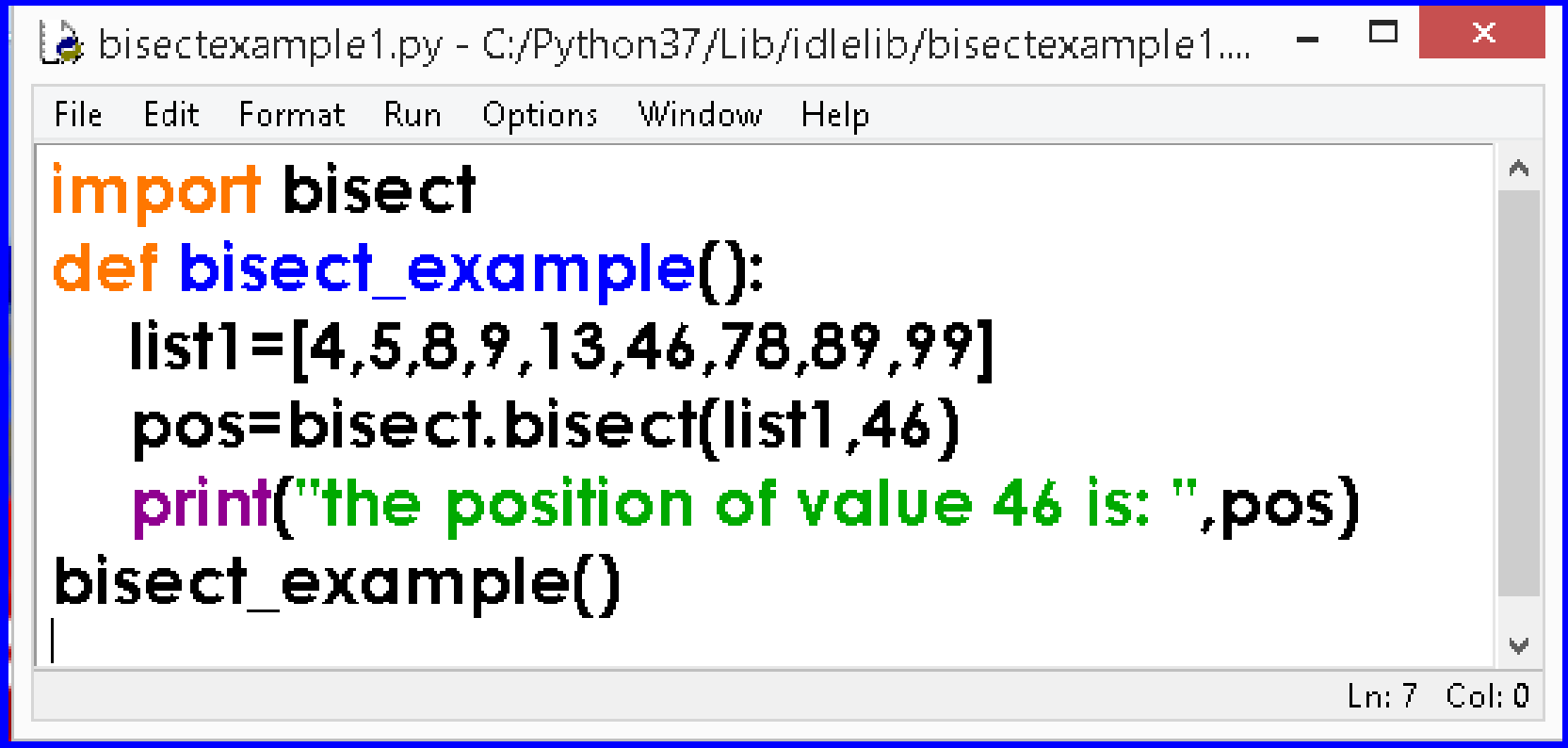
bisect.bisect(list,element)

For example: bisect.bisect(list1,46)
6

bisect MODULE

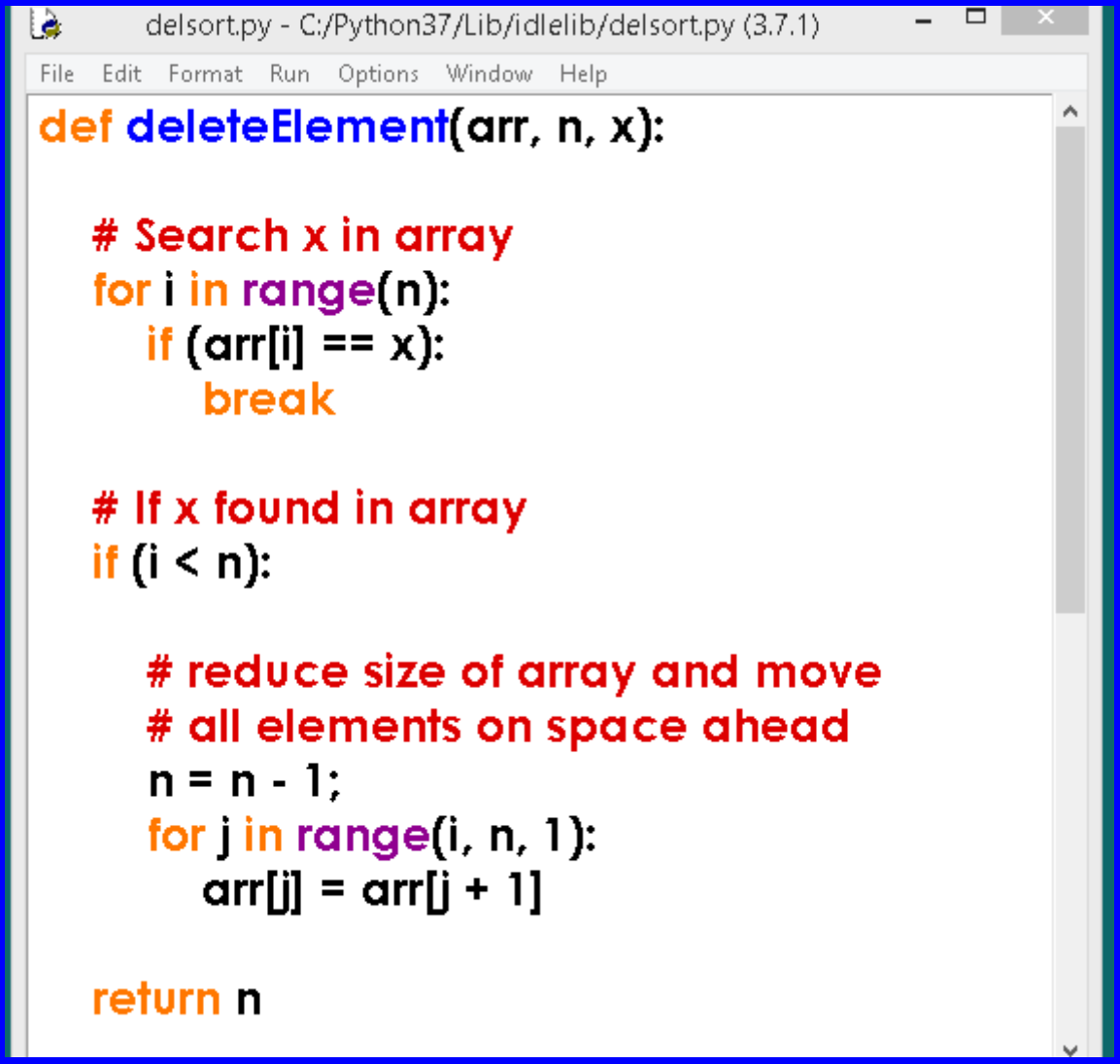


bisect MODULE



Delete an element from an array

Delete an element from an array



LIST COMPREHENSIONS (LC)

LIST COMPREHENSIONS (LC)

What is List Comprehensions (LC)?

List comprehension is an elegant way to define and create list in python.

We can create lists just like mathematical statements and in one line only. The syntax of list comprehension is easier to grasp.

LIST COMPREHENSIONS

The list comprehension starts with a '[' and ']', to help you remember that the result is going to be a list..

Syntax:

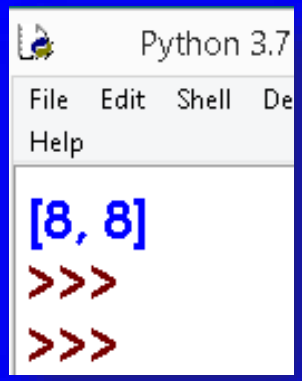
List1=[expression for item in list if conditional]

For example:

```
list2=[i for i in range(0,5)]
print (list2)
```

LIST COMPREHENSIONS - Examples

OUTPUT



Understanding the Code:

If n is less than 5 generate 2 values i.e n x 2 as per the for loop else generate 2 values i.e n x 4, the out put will be:- 8 8

LIST COMPREHENSIONS - Examples

Create a list called points

i) Using for loop

ii) Using List comprehension

Ans: i)

```
Points=[]
for i in range (0,5):
    Points.append(i)
```

Ans ii)

```
Points=[i for i in range (0,5)]
```

LIST COMPREHENSIONS - Examples

Create a list called vals which stores square of numbers

Ans:

```
vals=[i*2 for i in range (0,5)]
```

LIST COMPREHENSIONS - Examples

Given an input list scores, produce a list namely scores2, using list comprehension having numbers from scores that are multiples of 4.

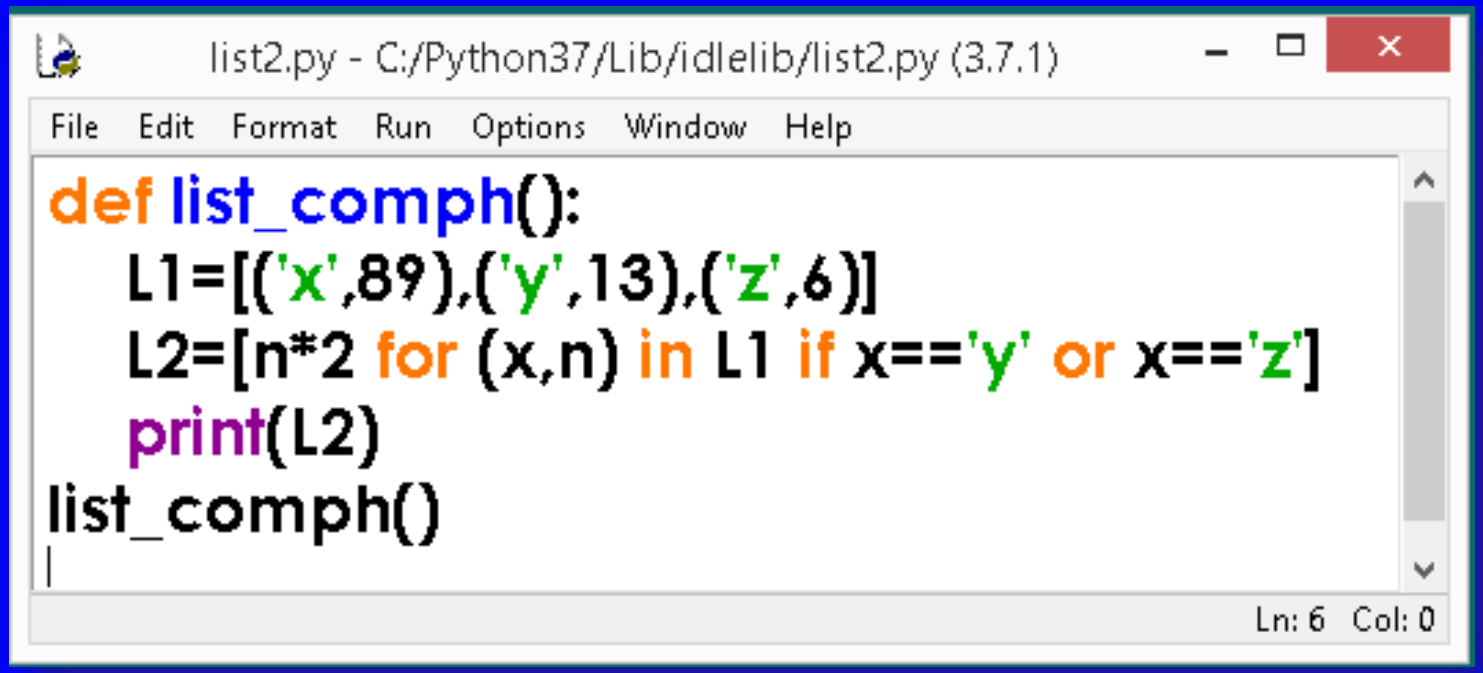
Scores=[23,4,12,16,18,9]

Ans:

```
Scores2=[num for num in scores if
    num%4==0]
```

LIST COMPREHENSIONS - Examples

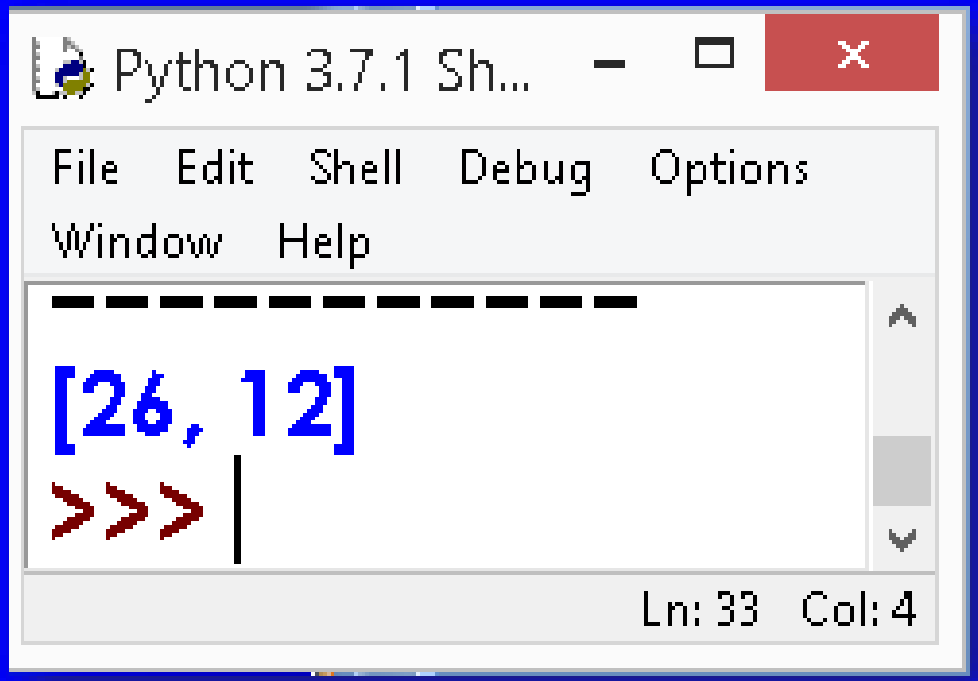
Consider the following code and write the output.



LIST COMPREHENSIONS - Examples

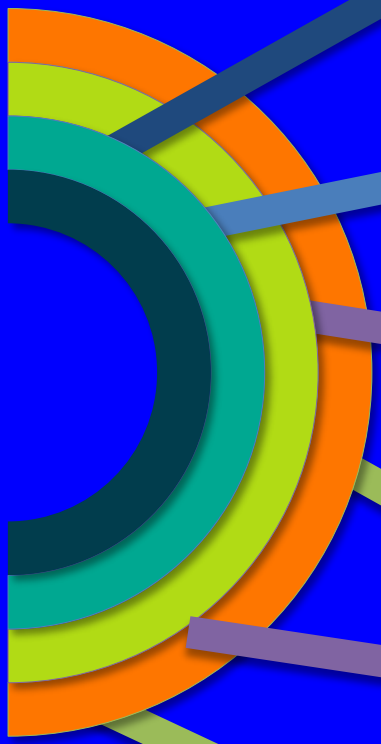
Consider the following code and write the output.

OUTPUT



LIST COMPREHENSIONS - Advantages

LIST COMPREHENSIONS - Advantages



01 CODE REDUCTION

02 FASTER CODE EXECUTION

03 READABLE

04 LESS ERROR PRONE

05 PYTHONIC

06 INTENT IS CLEAR

LIST COMPREHENSIONS - Advantages

1. CODE REDUCTION

A code of multiple statements gets reduced to single line of code.

2. FASTER CODE EXECUTION

LC are executed faster than loops and other equivalent statements.

LIST COMPREHENSIONS - Advantages

2. FASTER CODE EXECUTION

There are two reasons for faster execution:

i) Python will allocate memory first before adding elements to it.

ii) Calls append() function get avoided, reducing function overhead time.

LIST COMPREHENSIONS - Advantages

3. READABLE

It is more readable (*when you get used to it*).

4. LESS ERROR PRONE

Once you are familiar with LC, there is less chance to error prone.

LIST COMPREHENSIONS - Advantages

3. READABLE

It is more readable (*when you get used to it*).

4. LESS ERROR PRONE

Once you are familiar with LC, there is less chance to error prone.

LIST COMPREHENSIONS - Advantages

5. PYTHONIC

LC are more PYTHONIC as they truly represent the python code, hence generate interest among developers to write concise and accurate code.

LIST COMPREHENSIONS - Advantages

6. INTENT IS CLEAR

Intent is clear: to initialise a list. A loop could be doing anything, you have to read it to check that it doesn't have any other side effects. A list comprehension just sets up the list, you know it isn't up to anything else.

NESTED LISTS or TWO DIMENSIONAL LISTS

NESTED LISTS or TWO DIMENSIONAL LISTS

A list is a container which holds comma-separated values (items or elements) between square brackets where items or elements need not all have the same type.

A nested list is a list that appears as an element in another list. In this list.

NESTED LISTS or TWO DIMENSIONAL LISTS

```
nums = [[1, 2], [3, 4], [5, 6]]
print(nums[0])
print(nums[1])
print(nums[2])
print(nums[0][0])
print(nums[0][1])
print(nums[1][0])
print(nums[2][1])
```

Creating 2D List

Creating 2D List

```
L1=[]
r=int(input("How many rows?"))
c=int(input("How many Columns?"))
for i in range(r):
    row = []
    for j in range( c):
        ele=int(input("element", i , " , " , j ))
        row.append(ele)
    L1.append(row)
print("List is " ,L1)
```

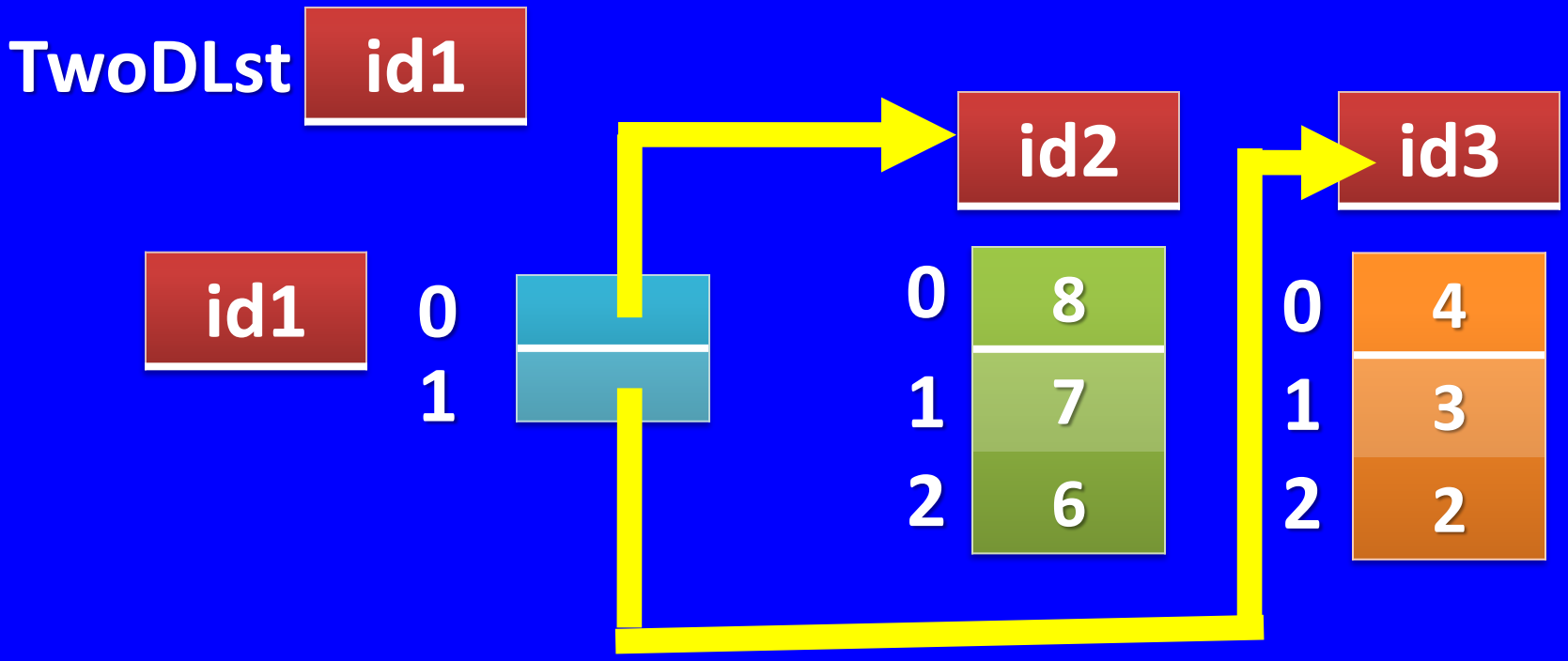
TRAVERSING 2D LIST

```
print("List is " ,L1)
for i in range(r):
    print("\t[",end=" ")
    for j in range( c):
        print(L1[i][j], end=" ")
        print("]")
    print("\t ] " )
```

REPRESENTATION OF 2D LIST IN MEMORY

REPRESENTATION OF 2D LIST IN MEMORY

TwoDLst= [[8 , 7 , 6] , [4 , 3 , 2]]



REPRESENTATION OF 2D LIST IN MEMORY

What is Regular 2D List?

If row size and column size are same then it is said to be **Regular 2D List**.

What is Ragged List?

If row size and column size are not same then it is said to be **Ragged List**.

SLICES IN 2D LISTS

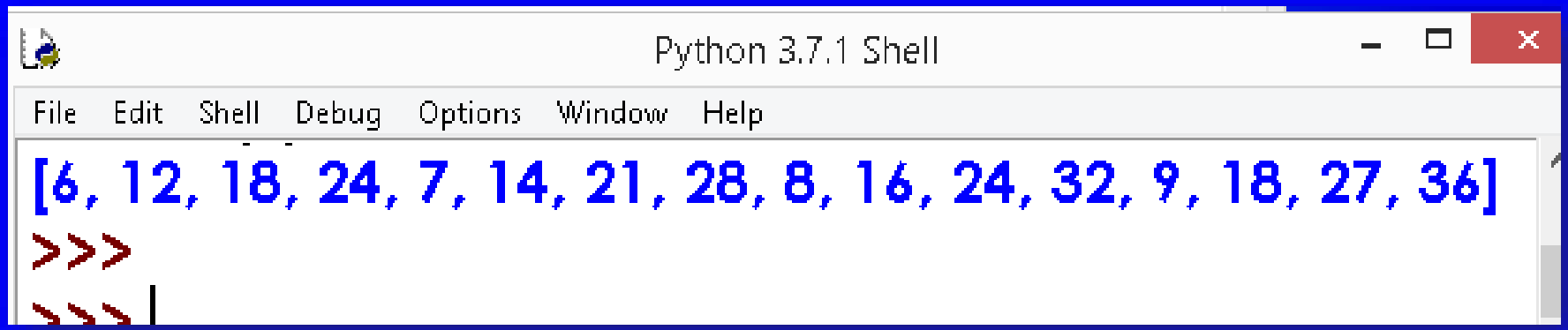
TwoDLst= [[8 , 7 , 6] , [4 , 3 , 2] , [7 ,3 , 1]]

```
>>>TwoDLst[:1]
[ [ 8 , 7 , 6 ] , [ 4 , 3 , 2 ]]
>>> TwoDLst[1:]
[[ 4 , 3 , 2 ] , [ 7 ,3 , 1 ] ]
```

FIND THE OUTPUT

FIND THE OUTPUT

OUTPUT



Thank You