Week 1

Introduction to Data Structures, operations, classification, Characteristics. Primitive types – primitive data structures, python examples. Non primitive types - Non primitive data structures, python examples. Linear and nonlinear data structures – with python examples. Introduction, Abstractions, Abstract Data Types, An Example of Abstract Data Type (Student, Date, Employee), Defining the ADT, Using the ADT, Implementing the ADT

Introduction to Data Structures:

- The data structure deals with the study of how the data is organized in the memory, how the data can be retrieved & how the data are manipulated.
- The way of representing data internally in the memory is called data structure.
- A **data structure** is a particular way of organizing data in a computer so that it can be used effectively.

Operations:

There are different types of operations that can be performed for the manipulation of data in every data structure.

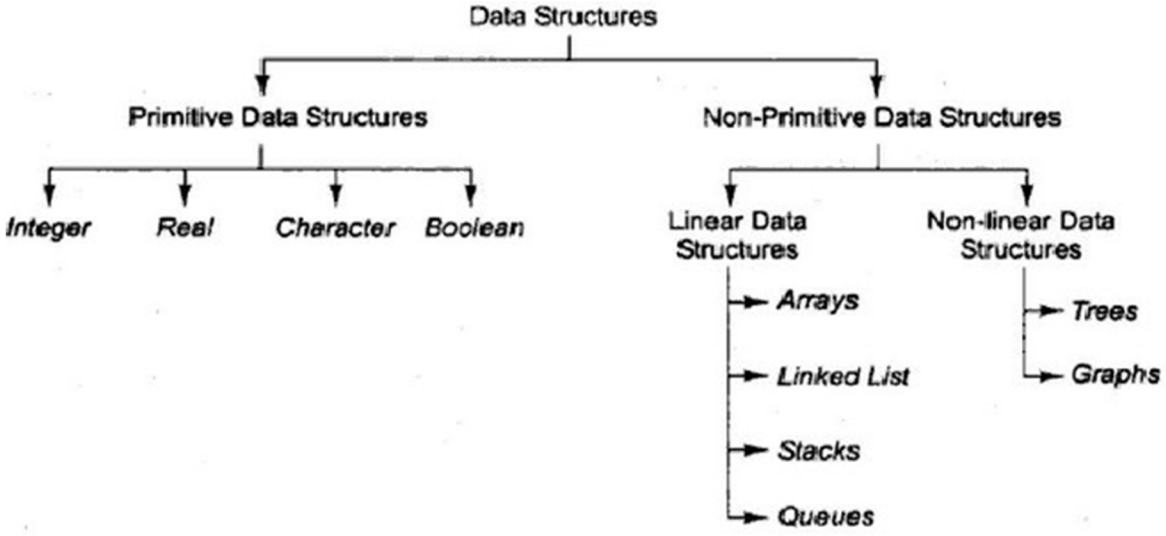
- **Traversing**: Traversing a Data Structure means to visit the element stored in it. This can be done with any type of DS.
- **Searching:** Searching means to find a particular element in the given data-structure. It is considered as successful when the required element is found
- **Insertion:** Insertion means to add an element in the given data structure. The operation of insertion is successful when the required element is added to the required data-structure.
- **Deletion:** Deletion means to delete an element in the given data structure. The operation of deletion is successful when the required element is deleted from the data structure.

Characteristics:

- **Correctness** − Data structure implementation should implement its interface correctly.
- **Time Complexity** − Running time or the execution time of operations of data structure must be as small as possible.
- **Space Complexity** − Memory usage of a data structure operation should be as little as possible.

Classification:

- The classification of data structures is given below.



Primitive Data Structures:-

- The primitive data structures are the kind of data structures that are derived from the primitive data types.
- These are directly manipulated by the machine instructions
- Ex: - integers, floating point numbers, String, Boolean.

✓ **Integers –** This data type is used to represent numerical data, that is, positive or negative whole numbers without a decimal point. Ex: **-1, 3, or 6**.

✓ **Float –** Float signifies **'floating-point real number'.** It is used to represent rational numbers, usually containing a decimal point like **2.0 or 5.77**.

✓ **String –** This data type denotes a collection of alphabets, words or alphanumeric characters. It is created by including a series of characters within a pair of double or single quotes. **Ex: 'blue', 'red', etc.,**

✓ **Boolean –** This data type is useful in comparison and conditional expressions and can take up the values **TRUE** or **FALSE**.

Non Primitive data structures:-

✓ The non primitive data structures are the kind of data structures that are derived from the primitive data structures.

✓ These cannot be directly manipulated by the machine instructions

✓ Ex:- Arrays, Stack, Queues, Linked List, Graphs, Trees, etc

✓ Non Primitive data structures are further classified as

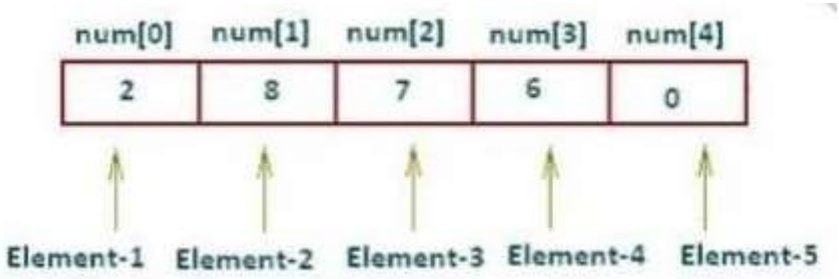
- Linear Data Structures
- Non Linear Data Structures

Linear Data Structures:

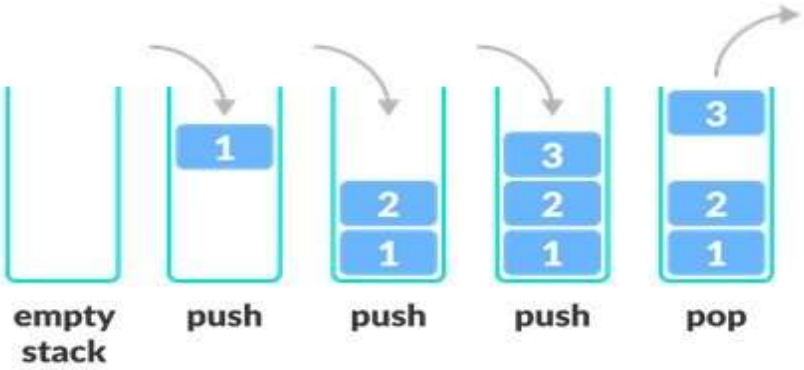
- In linear data structure, data elements are organized **sequentially** and therefore it is easy to implement in the computer memory.

These are the data structures which store the data elements in a sequential manner.

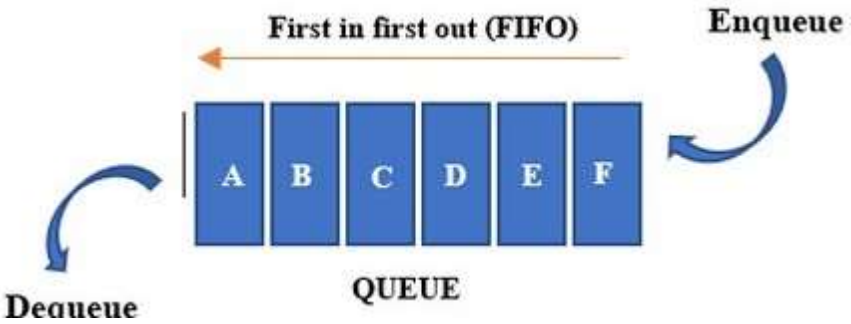
✓ **Array** − Array is a collection of elements with similar data type. It holds all elements in a sequential order.



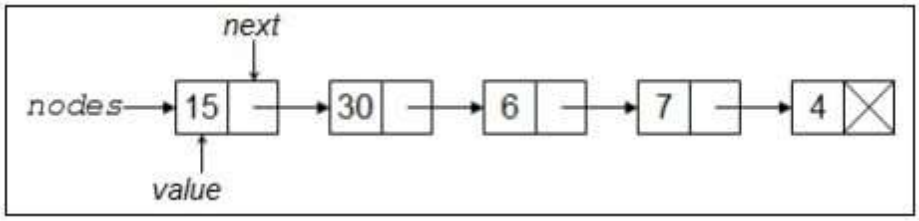
✓ **Stack** − Stack is a linear data structure which follows a particular order **LIFO**(Last In First Out) in which the operations are performed.



- ✓ **Queue** − Queue is also a linear data structure, in which the element is inserted from one end called **REAR**, and the deletion of existing element takes place from the other end called as **FRONT** and the order of queue is **FIFO** (First In First Out)



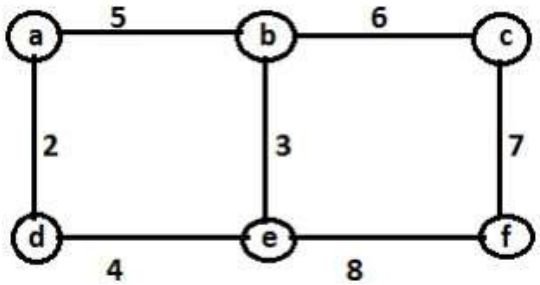
- ✓ **Linked List** − Linked List is a linear data structure and each data element contains a link to another element along with the data present in it.



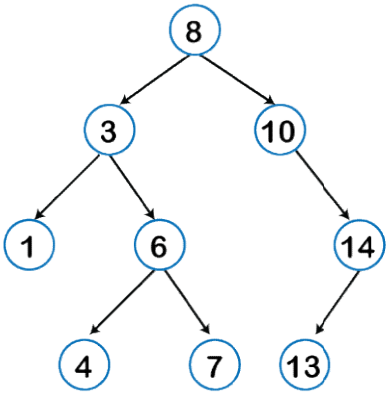
Non Linear Data Structures:

- ➢ In non-linear data structure, data elements are connected to several other data elements so that given data element has reach to one or more data elements.
- ➢ In non-linear data structure there is no any sequences.
- ➢ Every data item is attached to several other data items so that relationship is established between the items.
- ➢ Ex: Graph, Tree
- ➢ **Graph:**
  - ✓ Graph is collection of vertices and edges.
  - ✓ Graph is used to represents networks; the network may include paths and nodes
  - ✓ It is represented as **G={V,E}**

Where V-> vertices and E -> Edges



- ➢ **Tree:**
  - ✓ Tree is a connected acyclic graph.
  - ✓ Tree is a non-linear structure.
  - ✓ Tree is a collection of nodes linked together to simulate hierarchy



Difference between Linear and Non Linear Data Structures:

Linear Data Structures	Non Linear Structures
Data is arranged in linear form.	Data is arranged in non linear form.
Every item is related to its previous and next item.	Every item is attached with many other items
Implementation is easy	Implementation is difficult
Data items can be traversed in a single run.	Data items cannot be traversed in a single run.
Ex:- array, linked list, stack, queue	Ex:- tree, graph

Abstract Data type (ADT):

- Abstract Data type (ADT) is a special kind of data type (or class) whose behavior is defined by a set of values and a set of operations.
- The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented.
- It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.
- It is called "**abstract**" because it gives an implementation-independent view.
- The process of providing only the essentials and hiding the details is known as **abstraction**.

Stack Abstract Data Type:

- A stack is structured as an ordered collection of items where items are added to and removed from the end called the "top". Stacks are ordered LIFO.

- **Stack():** creates a new stack that is empty. It needs no parameters and returns an empty stack.
- **push(item):** adds a new item to the top of the stack. It needs the item and returns nothing.
- **pop():** removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.
- **peek():** returns the top item from the stack but does not remove it. It needs no parameters. The stack is not modified.
- **isEmpty():** tests to see whether the stack is empty. It needs no parameters and returns a boolean value.

- **size():** returns the number of items on the stack. It needs no parameters and returns an integer.

STACK ADT

```
class stack:
    def__init_(self):
        self.items = []
    def isEmpty(self):
        return self.items == []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def peek(self):
        return self.items[len(self.items) - 1]
    def size(self):
        return len(self.items)
    def display(self):
        return (self.items)
s=stack()
print(s.isEmpty())
print("push operations")
s.push(11)
s.push(12)
s.push(13)
print("size:",s.size())
print(s.display())
print("peek",s.peek())
print("pop operations")
print(s.pop())
print(s.pop())
print(s.display())
print("size:",s.size())
```

Date Abstract Data Type:

```
class date:
    def__init_(self,a,b,c):
        self.d=a
        self.m=b
        self.y=c
    def  day(self):
       print("Day = ", self.d)
    def month(self):
       print("Month = ", self.m)
    def year(self):
       print("year = ", self.y)
    def monthName(self):
       months = ["Unknown","January","Febuary","March","April","May","June","July",
        "August","September","October","November","December"]
       print("Month Name:",months[self.m])
    def isLeapYear(self):
       if (self.y % 400 == 0) and (self.y % 100 == 0):
          print("It is a Leap year")
       elif (self.y % 4 == 0) and (self.y % 100 != 0):
         print("It is a Leap year")
       else:
         print("It is not a Leap year")
d1 = date(3,8,2000)
d1.day()
d1.month()
d1.year()
d1.monthName()
d1.isLeapYear()
```