What is Data Structure?

The data structure name indicates itself that organizing the data in memory. There are many ways of organizing the data in the memory as- Array, Linked list etc.

**Types of Data Structures**

There are two types of data structures:

1. Primitive data structure
2. Non-primitive data structure

**Primitive Data structure**

The primitive data structures are primitive data types. e.g- int, char, float, double etc.

**Non-Primitive Data structure**

The non-primitive data structure is divided into two types:

1. Linear data structure
2. Non-linear data structure

**Linear Data Structure**

The arrangement of data in a sequential manner is known as a linear data structure.

In these data structures, one element is connected to only one element in a linear form.

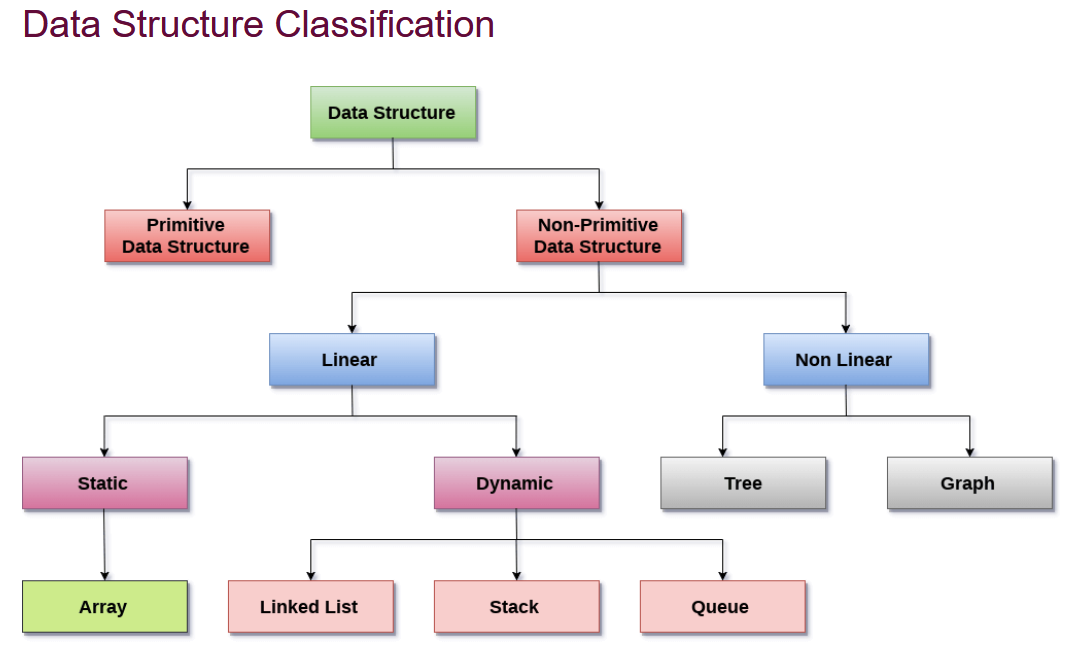
The data structures used for this purpose are Arrays, Linked list, Stacks, and Queues.

**Non-linear data structure**

When one element is connected to the 'n' number of elements known as a non-linear data structure.

In this case, the elements are arranged in a random manner.

The best example is trees and graphs.



**Static data structure:**

It is a type of data structure where the size is allocated at the compile/creation time. Therefore, the maximum size is fixed. e.g.-- tuple

**Dynamic data structure:**

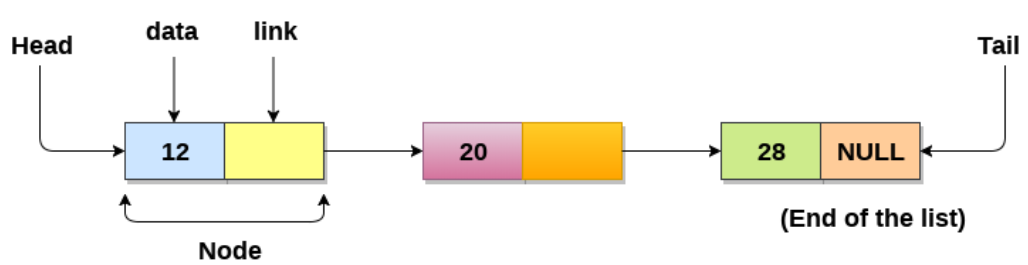
It is a type of data structure where the size is allocated at the run time. Therefore, the maximum size is flexible. e.g - list

**Linked List**

Linked List can be defined as collection of objects called nodes that are randomly stored in the memory.

A node contains two fields i.e. data and the pointer/link which contains the address of the next node in the memory.

The last node of the list must contain pointer to the null.



We can further break linked list in three parts—

1. Singly Linked list
2. Doubly linked list
3. Circular linked list

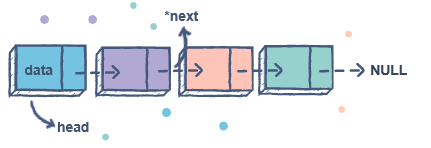
Linked list is same as array but array have below limitation which can be removed in linked list-

**Limitation of array**

1. The size of array must be known in advance before using it in the program.
2. All the elements in the array need to be contiguously stored in the memory. Inserting any element in the array needs shifting of all its predecessors.
3. Increasing size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.

**Singly Linked list**

A singly linked list is a type of linked list that is unidirectional, that is, it can be traversed in only one direction from head to the last node (tail).



**Creating singly linked list**

class Node:

   def \_\_init\_\_(self, dataval=None):

      self.dataval = dataval

      self.nextval = None

class SLinkedList:

    def \_\_init\_\_(self):

      self.headval = None

list1 = SLinkedList()

list1.headval = Node("Mon")

e2 = Node("Tue")

e3 = Node("Wed")

# Link first Node to second node

list1.headval.nextval = e2

# Link second Node to third node

e2.nextval = e3

print(list1.getvalue())

**Note:**

'headval' is instance variable which point to first value/node of the list. Don’t change of assign any other value of this variable b/c first node value may get changed.

**Accessing all values from singly linked list:**

class Node:

   def \_\_init\_\_(self, dataval=None):

      self.dataval = dataval

      self.nextval = None

class SLinkedList:

    def \_\_init\_\_(self):

      self.headval = None

    def getvalue(self):

        print('head value is: ',self.headval)

        while self.headval:

            print(self.headval.dataval)

            self.headval=self.headval.nextval # assign next node

list1 = SLinkedList()

list1.headval = Node("Mon")

e2 = Node("Tue")

e3 = Node("Wed")

# Link first Node to second node

list1.headval.nextval = e2

# Link second Node to third node

e2.nextval = e3

list1.getvalue()

**Inserting values at end of linked list**

Here we have defined a method insertval. When nextval points to None that means we are at end of list and will assign it’s address pointer to new value.

class Node:

   def \_\_init\_\_(self, dataval=None):

      self.dataval = dataval

      self.nextval = None

class SLinkedList:

    def \_\_init\_\_(self):

      self.headval = None

    def insertval(self, data):

        newNode = Node(data)

        if(self.headval):

            current = self.headval

            while(current.nextval):

                current = current.nextval

            current.nextval = newNode # we are at last element, add new data in list

        else:

            self.headval = newNode

    def getvalue(self):

        # print('head value is: ',self.headval.dataval)

        while self.headval:

            print(self.headval.dataval)

            self.headval=self.headval.nextval

list1 = SLinkedList()

list1.headval = Node("Mon")

e2 = Node("Tue")

e3 = Node("Wed")

# Link first Node to second node

list1.headval.nextval = e2

# Link second Node to third node

e2.nextval = e3

# list1.getvalue()

list1.insertval('Thu')

list1.getvalue()

Getting the size of linked list

class SLinkedList:

    def \_\_init\_\_(self):

      self.headval = None

    def getsize(self):

        size=0

        if (self.headval) != None:

            while self.headval:

                print(self.headval.dataval)

                self.headval=self.headval.nextval

                size+=1

            print('size is: ',size)

        else:

            print('size is: ',size)

**Note:**

Here we have created getsize method which count number of data till the last node.

Last data means nextval is None.

**Questions**

Do insert operation in the linked list and then get the size of linked list

class Node:

   def \_\_init\_\_(self, dataval=None):

      self.dataval = dataval

      self.nextval = None

class SLinkedList:

    def \_\_init\_\_(self):

      self.headval = None

    def getsize(self):

        size=0

        if (self.headval) != None:

            while self.headval:

                print(self.headval.dataval)

                self.headval=self.headval.nextval

                size+=1

            print('size is: ',size)

        else:

            print('size is: ',size)

    #this methos is for inserting new value at end of linkedlist

    def insertval(self, data):

        newNode = Node(data)

        if(self.headval) != None:

            current = self.headval

            while(current.nextval):

                current = current.nextval

            current.nextval = newNode

        else:

            self.headval = newNode

list1 = SLinkedList()

list1.headval = Node("Mon")

e2 = Node("Tue")

e3 = Node("Wed")

# Link first Node to second node

list1.headval.nextval = e2

# Link second Node to third node

e2.nextval = e3

list1.insertval('Thu')

print()

list1.getsize()

**Searching data into linked list**

class Node:

   def \_\_init\_\_(self, dataval=None):

      self.dataval = dataval

      self.nextval = None

class SLinkedList:

    def \_\_init\_\_(self):

      self.headval = None

    def search(self,data):

        firstnode=self.headval

        while firstnode:

            if firstnode.dataval==data:

                print(data,' found in node')

                return 1

            firstnode=firstnode.nextval

        else:

            print(data,' not found in node')

Note:

We have below logic in search method:

1. Get the head(first value) value of node if comparing\_data is present the return success.
2. If comparing\_data not present the go to next node(till next node is not None, means last node) compare with next node data.
3. If comparing \_data found then return success else repeat step 2.

**Delete data from linked list:**

This is same as search but here wi will have to remember the previous node also.

When data is found then then we will assign/move pointer/address of previous\_node next of found node.

class Node:

   def \_\_init\_\_(self, dataval=None):

      self.dataval = dataval

      self.nextval = None

class SLinkedList:

    def \_\_init\_\_(self):

      self.headval = None

    def deletedata(self,data):

        current\_node=self.headval

        previous\_node=None

        while current\_node:

            if current\_node.dataval!=data:

                previous\_node=current\_node #assing curret\_node to previous\_node variable

                current\_node=current\_node.nextval #change current\_node to next node

            else: #data found, come out of loop

                break

        if previous\_node is None: #required data is at first Node, just change the headval

            self.headval=current\_node.nextval

            return 1

        if current\_node is None: #required data is not present in Node

            return 1

            print('data is not present in Node')

        else:                    #requried data is presetn in Node **----- Line xyx**

            previous\_node.nextval=current\_node.nextval

**line xyz**

here we are assigning pointer of previous\_node next of current\_node.