Q. What are Linked Lists?

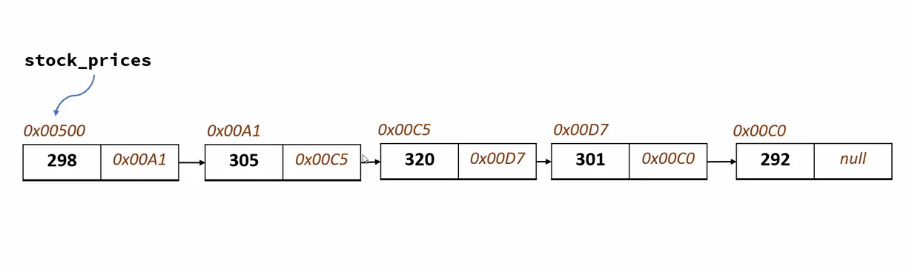
A -A **linked list** is a sequence of data elements, which are connected together via **links**. Each data element contains a connection to another data element in form of a **pointer**. **Python** does not have **linked lists** in its standard library. ... In this type of data structure there is only one **link** between any two data elements. (but can be doubly linked to be iterable backwards)

Q. Why do we use Linked Lists?

A- Linked lists can be a better option over Lists for certain reasons-

1) We do not need to preallocate space, unlike lists.

For example, when declaring an array, we need to declare its size. Using linked lists, we do not have to worry about preallocating space, the space is worked on as we add the data. Linked list works by keeping **data** at different parts of the memory and keeping track of the next data using a reference called **next**.

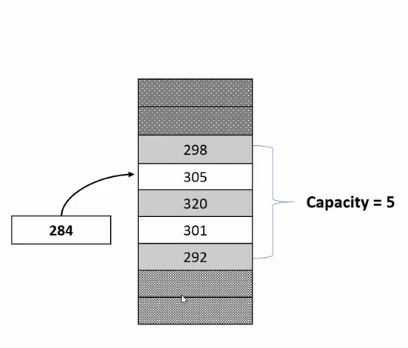


Note- The numbers starting with 0x…. Are the memory addresses. They refer to the next object’s address, where the data is located.

**A linked list object has two parts, data and next. Data contains the number etc, and next refers to the next object.**

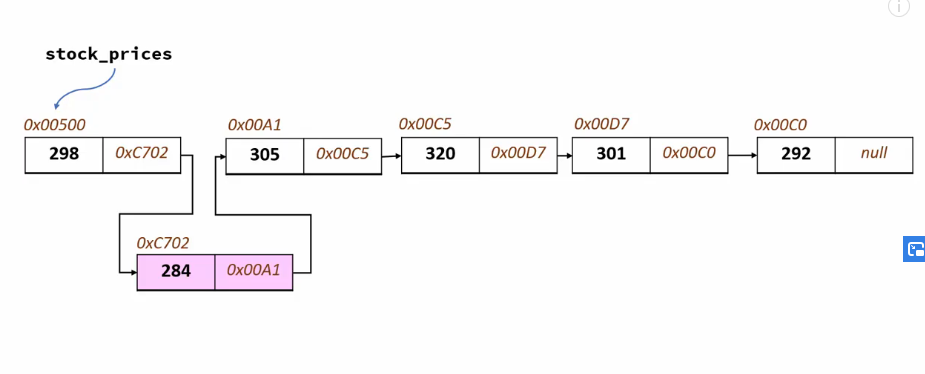
2) Insertion is easier

If we want to insert a number at a certain place, it will take much less time and memory than a list, as a list puts a number at the desired position and moves all the elements to its right an index further.

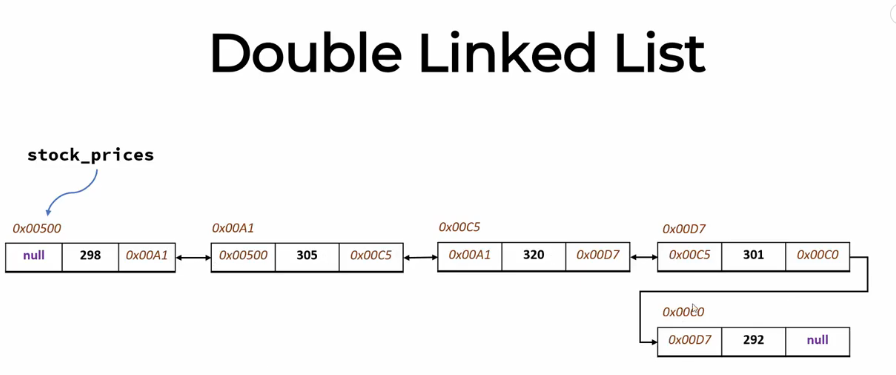


Note - If we wanted to insert 284 to our index number one, we would have to move indexes 2-5 one index.

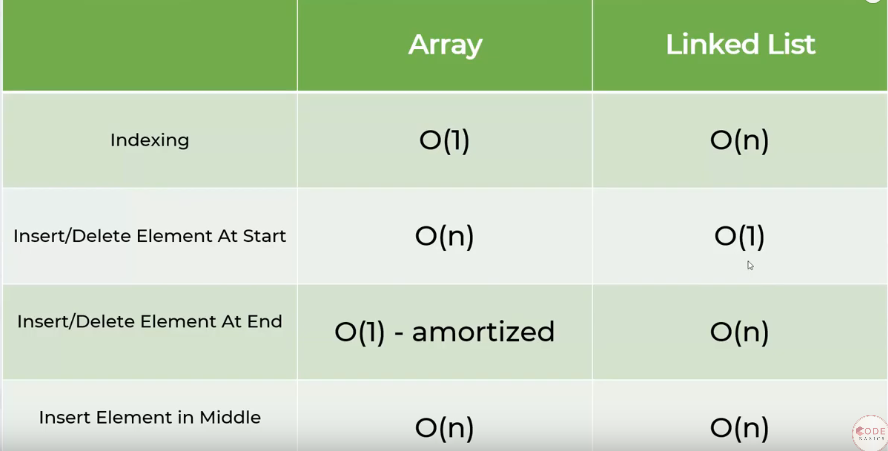
Linked lists are different, they just change the reference to the new item and connect the previously connected item. This is visualized in the below diagram.



Note - Linked lists can also be doubly linked, which will enable us to iterate backwards.



**Comparing Complexities of Lists and Linked Lists-**

****

How to create a linked list?

* We basically create a Node, and then keep attaching nodes to it and then rename the one to be referring to None.
* The head node never has data, it just helps us to know the starting point of the list and is passed on during functions.

Algorithm for creating a Linked List-

* Create a **Node** class.
* Create a class **Linked List** which is a subclass of **Node.**
* Create a function to iterate over the values to be added and link them to the nodes.
* Name the last node as None**.**

**###** Linked lists can have functions as needed. Very common ones are - adding a data, removing a data, finding the size of the linked list, checking if a data is in this list and so on.

**Python Implementation of Linked List -**

class Node:

def \_\_init\_\_(self,data):

self.data = data

self.next = None

class Linkedlist:

def display(self,head):

current = head

while current:

print(current.data,end=' ')

current = current.next

def insert(self,head,data):

**#This method lets us add the values we get from input one by one**

if (head == None):

head = Node(data)

elif (head.next == None):

head.next = Node(data)

else:

self.insert(head.next,data)

return head

mylist= Linkedlist()

**#We are creating a Linkedlist object and assigning it to mylist variable.**

T=int(input())

head=None

for i in range(T):

data=(input())

head=mylist.insert(head,data)

mylist.display(head)

##Linked list method to insert at the beginning of the list -

Algorithm - We first assign the new node’s next pointer to the current head of the linked list.

**def InsertAtBeginning(self,newdata):**

**NewNode = Node(newdata)**

**# Update the new nodes next val to existing node**

**NewNode.nextval = self.headval**

**self.headval = NewNode**

##Linked list method to insert at the end of the list -

**def InsertAtEnd(self, data):**

**NewNode = Node(data)**

**if self.head is None:**

**self.head = NewNode**

**return**

**last = self.head**

**while(laste.next):**

**last = last.next**

**laste.next=NewNode**

##Linked List method to get the size of the list -

**def size(self):**

**current = head**

**count = 0**

**while current:**

**count+=1**

**current = current.next**

**return count**

##Full code implementing a linked List with insert and size methods -

**class Node:**

**def \_\_init\_\_(self,data):**

**self.data = data**

**self.next = None**

**class Linkedlist:**

**def display(self,head):**

**current = head**

**while current:**

**print(current.data,end=' ')**

**current = current.next**

**def insert(self,head,data):**

**if (head == None):**

**head = Node(data)**

**elif (head.next == None):**

**head.next = Node(data)**

**else:**

**self.insert(head.next,data)**

**return head**

**def size(self):**

**current = head**

**count = 0**

**while current:**

**count += 1**

**current = current.next**

**return count**

**mylist= Linkedlist()**

**t = int(input())**

**head = None**

**for i in range(t):**

**data = (input())**

**head = mylist.insert(head,data)**

**mylist.display(head)**

**mylist.size()**

**Important links -** [**https://www.tutorialspoint.com/python\_data\_structure/python\_linked\_lists.htm**](https://www.tutorialspoint.com/python_data_structure/python_linked_lists.htm)

[**https://www.tutorialspoint.com/python\_data\_structure/python\_linked\_lists.htm**](https://www.tutorialspoint.com/python_data_structure/python_linked_lists.htm)