Linked Lists :-

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. In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

Advantages of linked lists :-

Applications of linked list in computer science –

1. Implementation of stacks and queues
2. Implementation of graphs: Adjacency list representation of graphs is most popular which is uses linked list to store adjacent vertices.
3. Dynamic memory allocation: We use linked list of free blocks.
4. Maintaining directory of names
5. Performing arithmetic operations on long integers
6. Manipulation of polynomials by storing constants in the node of linked list
7. representing sparse matrices

Applications of linked list in real world-

1. *Image viewer* – Previous and next images are linked, hence can be accessed by next and previous button.
2. *Previous and next page in web browser* – We can access previous and next url searched in web browser by pressing back and next button since, they are linked as linked list.
3. *Music Player* – Songs in music player are linked to previous and next song. you can play songs either from starting or ending of the list.

Functions :

isEmpty(*self*)

Input requirements : No inputs.

It checks whether the linked list is empty or not. Returns **True** if Empty , else returns **False**.

count(*self*)

input requirements : No inputs.

It counts number of occurrences of a data element in the linked list and returns the count in dictionary format.

Return data format : { data_element : no. of occurrences }

Returns empty dictionary if linked list is empty.

headNode(*self*)

input requirements : No inputs.

It checks whether the linked list is empty or not. If it is not empty then returns the head node, else returns none.

Return data format : Node

endNode(*self*)

input requirements : No inputs.

It checks whether the linked list is empty or not. If it is not empty then returns the last node by traversing the linked list , else returns none.

Return data format : Node

nodeAtPosition(*self* , position=None)

Input requirements : Index value when linked list considered as list.

Acceptable input type: integers

It traverses the linked list till the given position and the returns the node. Default value of position is None.

Return data type : node

Exception cases :

- if no position is given.
- if position is out of bounds.

sum(*self*, start =None , end=None , step=None)

Input requirements : Start position , end position , step size and these are ptional.

Acceptable types : integers

It return sum of nodes' data from node at start position to node at end position(except end position's node data). Step is the increment value.

Default values for **start** is starting node, for **end** is last node, and **step** by 1.

User can call the function just by passing only 1 argument or 2 arguments as :

- sum(start = value) : Here it assigns last node for end and 1 for step size.
- sum(start = value , end = value) : Here it assigns 1 for step size.
- sum(start = value , end = value , step=value)

Return data type : integer or float.

Exception cases :

- If linked list is empty.
- if linked list contains any data type other than integers and float values.

mean(*self*)

Input requirements : No inputs

It returns the mean value of the linked list data .

Return data type : integer or float.

Exception cases :

- if linked list is empty
- if linked list contains any data types other than integers and float values.

median(*self*)

Input requirements : No inputs.

It returns the median value of the linked list data.

Return data type : integer or float

Exception cases:

- if linked list is empty
- if linked list contains any data types other than integers and float values

mode(*self*)

Input requirements : No inputs.

It returns the mode of the linked list data. If linked list contains more than one mode it return the mode in list format.

Return data format : integers or float values / list if multiple mode exist.

Exception cases:

- if linked list is empty
- if linked list contains any data types other than integers and float values.

slice(*self,start=None,end=None,step=None*)

Input requirements : No inputs required are else we can give start,end,step

It returns the sliced linked list.

Start: Starting node's index

End: Ending node's index

Step: Step increment or decrement

Return data format :Sliced linked list according to parameters

Exception cases:

- if linked list is empty

isUniqueType(*self*)

Input requirements : No inputs.

It returns the type of the linked list data if all the data in the linked list are same type. Otherwise False

Return data format : type of data/False

Exception cases:

- if linked list is empty

doesHaveType(*self*)

Input requirements :Required data type /List of data types/Tuple of Data types/Set of Data types

It returns True if linked list have at least one element of given data type/types

Return data format : True/False/List of True or False

Exception cases:

- if linked list is empty
- If incorrect data type is given

addToStart(*self,data=None,node=None*)

Input requirements : Either data/List of data to be entered on linked list or connective node

It add the given data/List of data to starting of the linked list. Or it adds the given nodes at starting of given linked list. It returns True if the given data added successfully.

Return data format : True/False

Exception cases:

- if linked list is empty
- If given data or node is inappropriate format
- If both data and node is given at a time
- If both data and node are not given are given None

addToEnd(*self,data=None,node=None*)

Input requirements : Either data/List of data to be entered on linked list or connective node

It add the given data/List of data to ending of the linked list. Or it adds the given nodes at ending of given linked list. It returns True if the given data added successfully.

Return data format : True/False

Exception cases:

- if linked list is empty
- If given data or node is inappropriate format
- If both data and node is given at a time
- If both data and node are not given are given None

addToPosition (*self,data=None,index=None,node=None*)

Input requirements : Either data/List of data to be entered on linked list or connective node and index positions

It add the given data/List of data to particular given index of the linked list. Or it adds the given nodes at particular location of given linked list. It returns True if the given data added successfully. If we give list of

data and positions it will sort the positions as well as data and add it in the ascending order to avoid unnecessary conflicts

Return data format : True/False

Exception cases:

- if linked list is empty
- If given data or node is inappropriate format
- If both data and node is given at a time
- If both data and node are not given are given None
- If given position index if out of range

insertAfter(*self,ref_node=None,data=None,node=None*)

Input requirements : Either data/List of data to be entered on linked list or connective node and the reference node to add after it

It add the given data/List of data after the given reference node of the linked list. Or it adds the given nodes at given reference node linked list. It returns True if the given data added successfully.

Return data format : True/False

Exception cases:

- if linked list is empty
- If given data or node is inappropriate format
- If both data and node is given at a time
- If both data and node are not given are given None
- If ref node is not found in given linked list

dataNode(*self,data*)

input requirements : data

It checks whether the linked list has a node containing given data or not. If it contains the data then it returns that node , else returns none.

Return data format : Node

isNodeIn(*self,node=None*)

input requirements : node reference

It checks whether the linked list has the given node or not. If it contains the node then it returns True, else returns none.

Return data format : True/False

length(*self*)

input requirements : No inputs required

It return the length of the given linked list. Gives zero if there is no elements

Return data format : integer

index(*self,data=None,rtype=dict*)

input requirements : required data /List data

It checks whether the given data / List of data points are in linked list are not and returns the index position of each data point. Default it returns the values in dict format with data and index positions, or else we can request it to return in list or tuple format.

Return data format : dictionary/list/tuple

atIndex(*self,index,rtype=dict*)

input requirements : required index /List of indexes

It checks whether the given index /List of indexes are in linked list are not and returns the data of each index. Default it returns the values in dict format with data and index positions, or else we can request it to return in list or tuple format.

Return data format : dictionary/list/tuple

replce(*self,dat=None,New_data=None,All=True*)

input requirements : required data /List of data with replacing values/list of values

It checks whether the given data /List of data are in linked list are not and returns the True if each data can be replaced with new data. We can request whether we have to change first occurrence or all elements containing the data

Return data format : True

updateNode(*self,node=None,data=None*)

input requirements : node and data

It checks whether the given node is in linked list are not and update the value of node to data.

Return data format : True

removeItem(*self,item=None,All=None,node=True*)

input requirements : required data /List of data or nodes

It checks whether the given data /List of data are in linked list or not and removes the elements if it presents. We can request whether we have to remove first occurrence or all elements containing the data as well as applicable for nodes

Return data format : True

Max(*self*)

input requirements : No inputs.

It checks whether the linked list is empty or not. If it is not empty then returns the maximum data value present in linked list, else returns none.

Return data format : Maximum value

Min(*self*)

input requirements : No inputs.

It checks whether the linked list is empty or not. If it is not empty then returns the manimum data value present in linked list, else returns none.

Return data format : Manimum value

remove_Max(*self,All=True*)

input requirements : All True or False

It checks whether the linked list is empty or not. If it is not empty then removes the maximum data value present in linked list, else returns none. We can request whether we can remove first occurrence or else all elements with max value.

Return data format : None

remove_Min(*self,All=True*)

input requirements : All True or False

It checks whether the linked list is empty or not. If it is not empty then removes the minimum data value present in linked list, else returns none. We can request whether we can remove first occurrence or else all elements with min value.

Return data format : None

push(*self,data=True*)

input requirements : data/ List of data

Same as addToEnd function it adds the given list of data to end of linked list

Return data format : True

popEnd(*self,No_of_times=1,rType=list*)

input requirements : How many times you want to pop and in which format you want store the poped items

It checks whether the linked list is empty or not. If it is not empty then removes the ending data node present in linked list, and returns in format of list. We can request whether we can remove one element or more then one element using No_of_times parameter .

Return data format : list

copy(*self*)

input requirements : No inputs

It return the copy of given linked list

Return data format : new linked list head node

clear(*self*)

input requirements : No inputs

It empties the entire linked list

Return data format : True

removeAtStart(*self,No_of_nodes=1*)

Input requirements : No of nodes to be removed on linked list.

It remove the given number of nodes at starting of the linked list. It returns True if the given data is removed successfully.

Return data format : True/False

Exception cases:

- if linked list is empty
- if no_of_nodes>len(linked list)

removeAtEnd(*self,No_of_nodes=1*)

Input requirements : No of nodes to be removed on linked list.

It remove the given number of nodes at ending of the linked list. It returns True if the given data is removed successfully.

Return data format : True/False

Exception cases:

- if linked list is empty
- if no_of_nodes>len(linked list)

removeAtPosition(*self,index=None*)

Input requirements : indexes of nodes to be removed on linked list.

It removes the given indexed nodes of the linked list. It removes the linked list in descending order so conflict will accrue .It returns True if the given data is removed successfully.

Return data format : True/False

Exception cases:

- if linked list is empty
- if index> len(linked list)

merge(*self,start*)

Input requirements: start node of another linked list.

It merges the given linked list with current linked list and returns the head node

Return data format : head node

toString(*self,separator=" "*)

Input requirements: separator to separate the given linked list data

It returns the string with linked list data with separator separated. Default it is " "(space).

Return data format : string

toList(*self)*

Input requirements: no inputs

It returns the list with linked list data.

Return data format : list

toSet(*self)*

Input requirements: no inputs

It returns the set with linked list data.

Return data format : set

reverse(*self)*

Input requirements: no inputs

It returns the new linked list with reversed node data's of current linked list.

Return data format : head node

removeDuplicates(*self)*

Input requirements: no inputs

It returns the linked list without duplicate values. It removes the duplicate values.

Return data format : node

equal(*self,llist=None)*

Input requirements: Linked list

It returns True if both linked list are equal otherwise False

Return data format : True/False

search(*self,data=None,rtype=dict*)

input requirements : required data /List data

It checks whether the given data / List of data points are in linked list are not and returns the True/False for each data point. Default it returns the values in dict format with data and True/False according to whether it present on linked list or not, or else we can request it to return in list or tuple format.

Return data format : dictionary/list/tuple

sorted(*self,reverse=False*)

input requirements : reverse=False for ascending order and reverse=True for descending order.

It sorts the self linked list and returns current linked list's head node. If we give True for reverse then it will sort in descending format

Return data format : head node

New_sorted(*self,reverse=False*)

input requirements : reverse=False for ascending order and reverse=True for descending order.

It copy the self linked list and sort it and returns new linked list's head node. If we give True for reverse then it will sort in descending format

Return data format : head node

isSorted(*self,reverse=False*)

input requirements : reverse=False for ascending order and reverse=True for descending order.

It return True if the given linked list is in sorted order otherwise false

Return data format : True/False

sortedAssign(*self,data=None,reverse=False*)

input requirements : data/List of data and reverse=False for ascending order and reverse=True for descending order.

It sorts the linked list and adds the given data/ list of data in a sorted order. So that ultimately we can return sorted linked list with given add on data

Return data format : head node

count(*self*)

input requirements : No inputs

It return the dictionary with set of values in linked with their occurrence count

Return data format : dictionary

dataCount(*self,data=None,rType=dict*)

input requirements : data/List of data to count the occurrences with return type

It return the dictionary with given data values in linked with their occurrence count. Or else we can request other return type like list/tuple with occurrences count

Return data format : dictionary