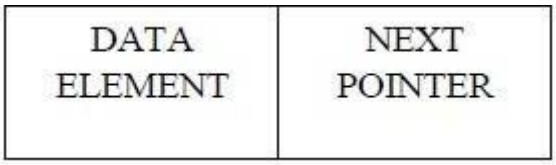
2.5 LINKED LIST IMPLEMENTATION OF LIST ADT

- Linked list consists of series of nodes.
- Each node contains the element and a pointer to its successor node.
- The pointer of the last node is NULL.



Reason for Linked List

- While declaring arrays, we have to specify the size of the array, which will restrict the number of elements that the array can store.

Advantages of using linked list

- A linked list does not store its elements in consecutive memory locations and the user can add any number of elements to it.
- However, unlike an array, a linked list does not allow random access of data. Elements in a linked list can be accessed only in a sequential manner.

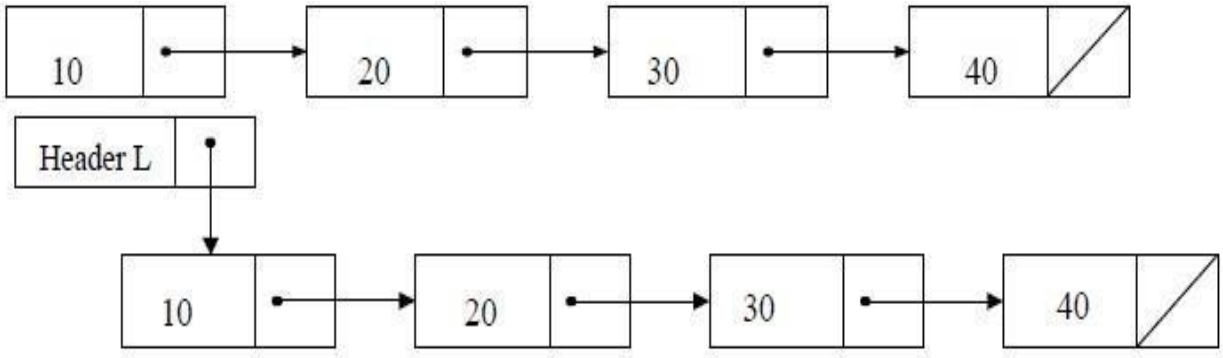
Types of linked list

1. Singly Linked List 2. Doubly Linked List 3. Circular Linked List.

2.6 SINGLY LINKED LIST

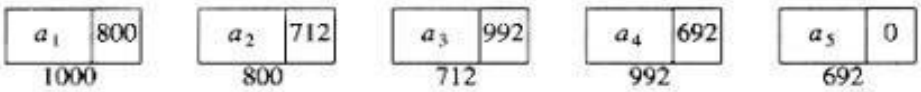
A singly linked list is a linked list in which each node contains only one link field pointing to the next node in the list.

Singly linked list



Header node points to the first node in the list

Linked list with actual pointer values



NODE DECLARATION

```
class Node
{
public:
   int data;
   Node *next;
};
```

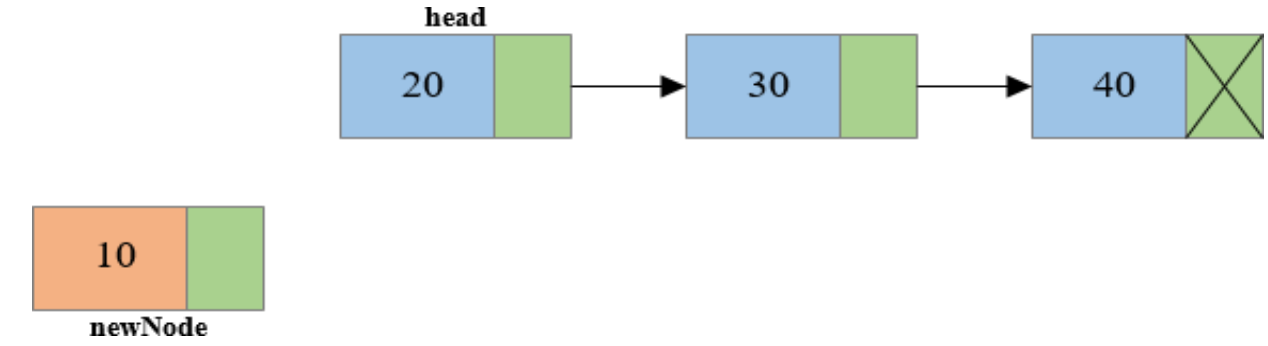
INSERTION:

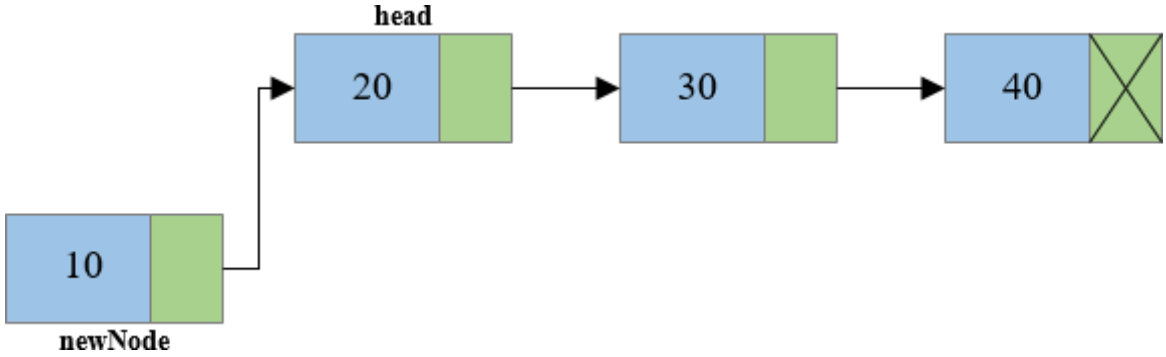
Overflow is a condition that occurs when no free memory cell is present in the system. When this condition occurs, the program must give an appropriate message.

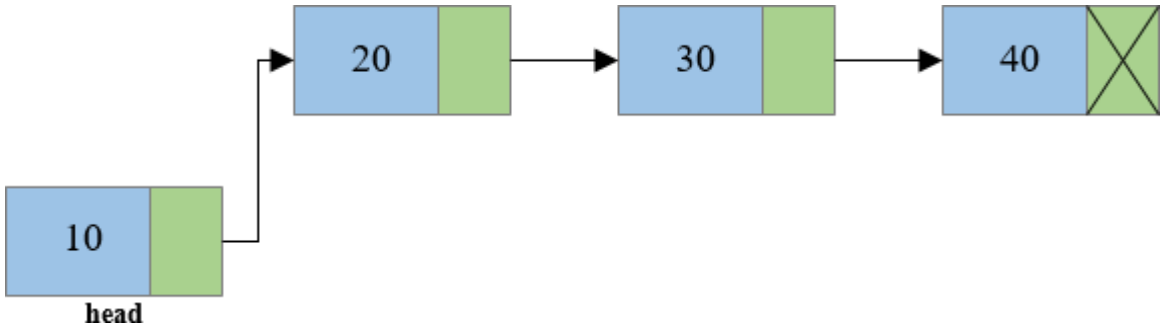
Insertion at the Beginning of the linked list:

To insert a node at the beginning of a singly linked list in Python, you need to follow these steps:

- Create a new node with the given data.
- Set the "next" pointer of the new node to point to the current head of the list.
- Update the head of the list to point to the new node.







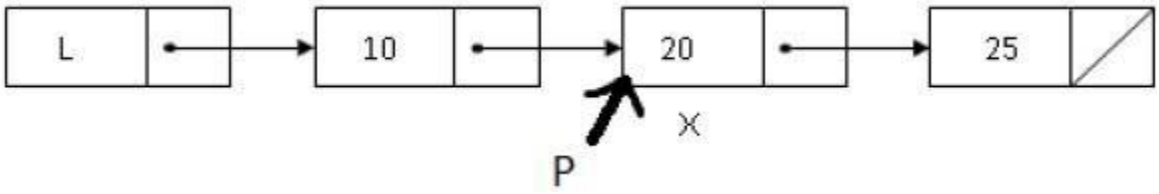

```
void insert(int item)
   {
      Node *temp, *newNode;
      newNode = new Node;
      newNode->data = item;
      newNode->next = NULL;

      if (head == NULL)
         head = newNode;
      else
      {
         temp = head;
         while (temp->next != NULL)
            temp = temp->next;
         temp->next = newNode;
      }
        cout << item << " inserted\n";
   }
```

SEARCHING:

- Start from the first node and compare the data part with

element to be searched

- If it is present, then the corresponding position is returned. target=20




```
void search(int key)
   {
      Node *temp = head;
      int pos = 1;

      while (temp != NULL)
      {
         if (temp->data == key)
         {
            cout << "Element " << key << " found at position " << pos << endl;
            return;
         }
         temp = temp->next;
         pos++;
      }

      cout << "Element " << key << " not found in the list" << endl;
   }
```

DELETION:

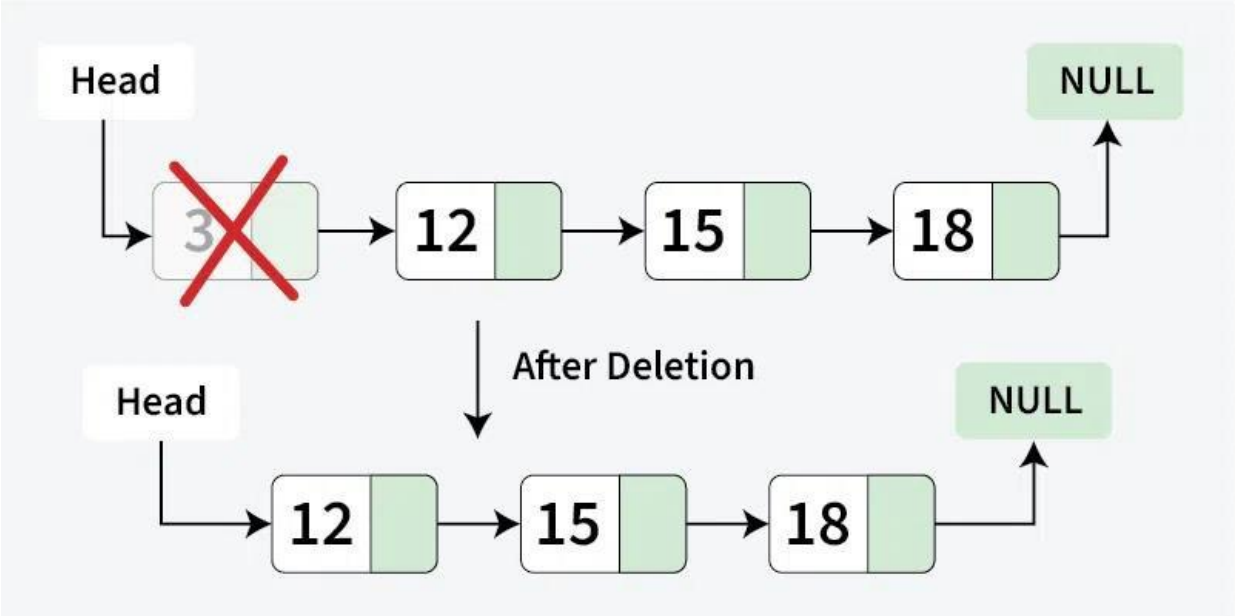
To delete a node from the beginning of a singly linked list in Python, you need

to follow these steps:

- If the list is empty (head is None), there is nothing to delete.
- Update the head of the list to point to the next node (if any).

```
void del(int item)
   {
      Node *temp = head;
      Node *prev = NULL;

      if (head == NULL)
      {
         cout << "List is empty\n";
         return;
      }

      if (head->data == item)
      {
         head = head->next;
         delete temp;
         cout << item << " deleted\n";
         return;
      }

      while (temp != NULL && temp->data != item)
      {
         prev = temp;
         temp = temp->next;
      }

      if (temp == NULL)
```

```
    {
        cout << item << " not found\n";
    }
    else
    {
        prev->next = temp->next;
        delete temp;
        cout << item << " deleted\n";
    }
}
```



TRAVERSE OR DISPLAY:

Traversal in a linked list means visiting each node and performing operations like printing or processing data.

Step-by-step approach:

- Initialize a pointer (current) to the head of the list.
- Loop through the list using a while loop until current becomes NULL.
- Process each node (e.g., print its data).

▶ Move to the next node by updating

temp = temp ->next.

```
void display()
  {
     Node * temp = head;
     if (head == NULL)
     {
        cout << "List is empty\n";
        return;
     }
     cout << "Singly Linked List: ";
     while (temp!= NULL)
     {
        cout << temp->data << " ";
        temp = temp ->next;
     }
     cout << endl;
  }
```

ADVANTAGES OF SINGLY LINKED LIST

1) Dynamic memory allocation avoids wastage of memory.

2) Insertion and deletion of values is easier as compared to array

DISADVANTAGES OF SINGLY LINLED LIST

1) Nodes can only be accessed sequentially.

2) Binary search algorithm cannot be implemented.

3) only forward traversal is possible.