Types of Linked List:

Now we have an idea about Linked List and how a linked list works. One important thing to remember is that everything regarding the linked list is created and maintained manually by us. We have designed the node class and also created the linked list. After that, we implemented the idea of indexing along with different operations such as insert, removal, rotation, and so on. If you think about it all these operations are a bit complicated because of our node class design which only has the option of moving from one node to another in a forward manner.

This brings the question can we design our linked list in such a way that we can traverse from one node to another in a forward and backward manner? In addition, can we make our linked list circular and is there any way where we will not need to handle the head explicitly?

To answer all these questions, the types of the linked list have been introduced. Linked Lists type can be determined by considering three different categories. Each category is independent of the other two options. We need to understand first what these options mean.

Category 1	Category 2	Category 3
Non-Dummy Headed	Singly	Linear
Dummy Headed	Doubly	Circular

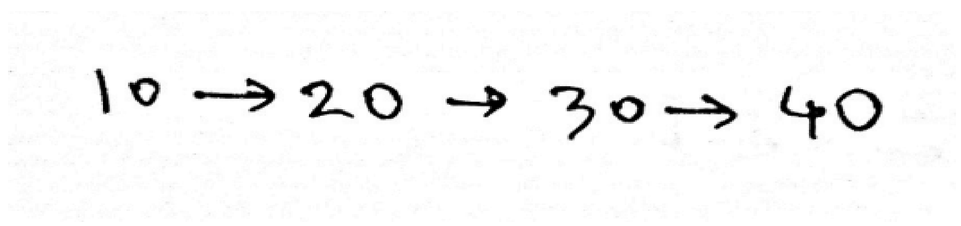
Non-Dummy Headed: It means that the linked list's first node (the head) contains an item along with the information of the next node. For example, the lists we have used till now are Non-Dummy headed.

Example: 10 → 20 → 30 → 40. In this Linked list, the first node stores the value head.

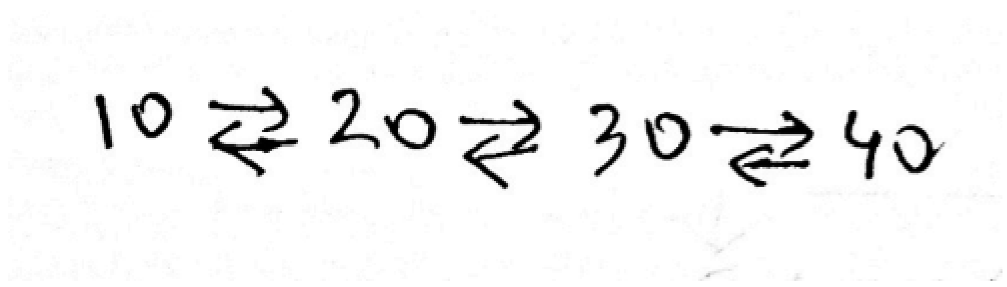
Dummy-Headed: This refers to the linked list where a head node is a node-type object but it does not store any element on its own. Rather it stores the information of the next node where the starting value is stored. The benefit of this is that we do not need to handle the first item of the data carefully now. To illustrate, we can remove or add items at beginning of the list without handling the head delicately. The reason is the first item is stored after the dummy head.

Example: DH → 10 → 20 → 30 → 40. In this linked list, DH refers dummy head which is a node without any element. The first item is stored after the dummy head.

Singly: It means every node has only the information of its next node. The reason is the design of the Node class has only one variable (next variable). Up until now all the list we worked on is Singly Linked List.

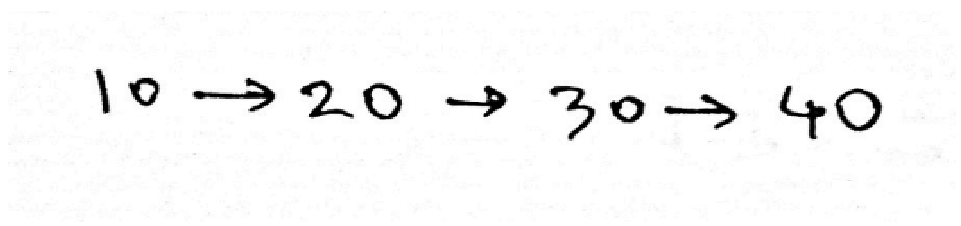


Doubly: It means a node knows its next node and its previous node's information. To achieve this we need to modify the Node class design where we store the previous node's information. Example of Doubly Node class in python

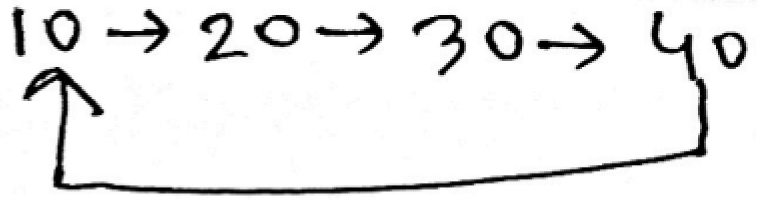


```
class DoublyNode:
        def __init__(self, elem, next, prev):
                self.elem = elem
                self.next = next # To store the next node's reference.
                self.prev = prev # To store the previous node's reference.
```

Linear: Linear linked list refers to the linked list where the linked list is linear in structure. To illustrate, the last node of the linked list will refer to None.



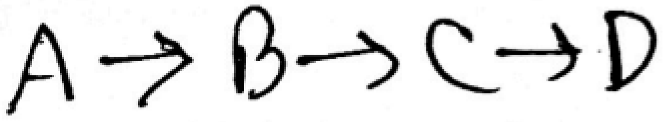
Circular: Circular linked list refers to a linked list where the linked list is circular in structure. To illustrate, the last node of the linked list will refer to the starting node.



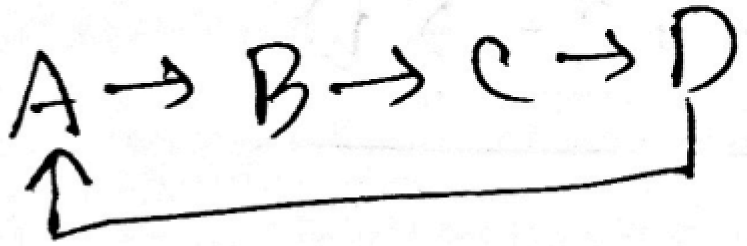
Now that we understand the options of each category, we can easily identify the type of a linked list. The main idea is that a linked list will contain one option from each category mentioned above. The option of one category does not have any impact on other categories.

The linked list we are taught until now is Non-Dummy Headed Singly Linear Linked List. You can think of this as a default linked list. It means if not mentioned, think of the head as Non-Dummy Head, Connection as Singly, and the structure as Linear. Now combining the categories the linked list can be 8 types. DH means dummy head in below diagram

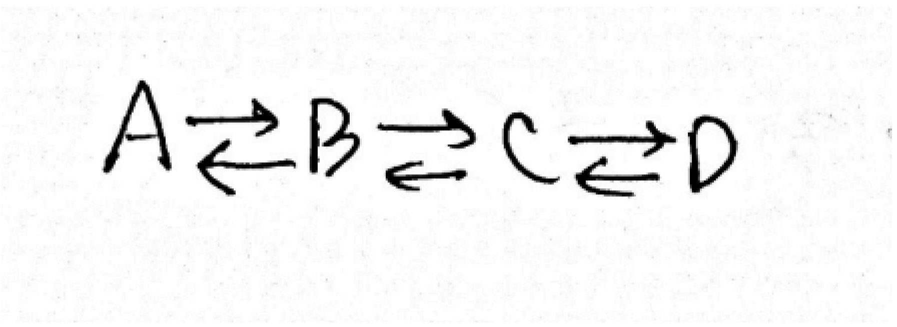
1. **Non-Dummy Headed Singly Linear Linked List:**



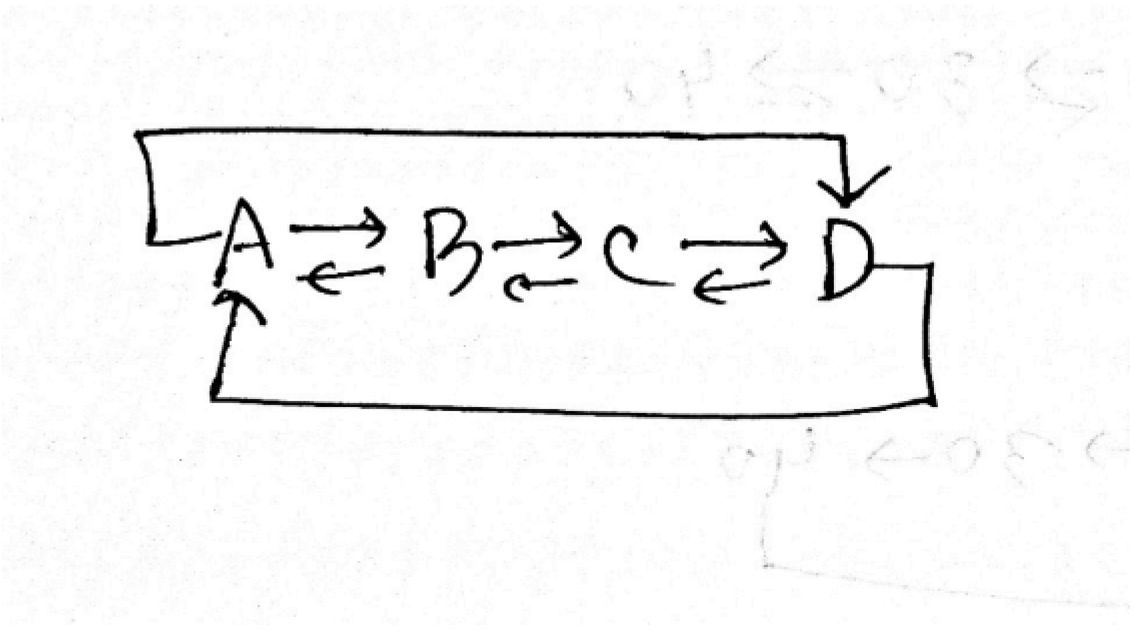
2. **Non-Dummy Headed Singly Circular Linked List:**



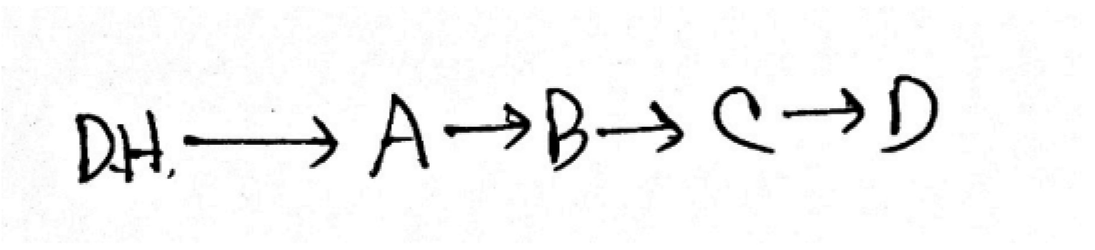
3. **Non-Dummy Headed Doubly Linear Linked List:**



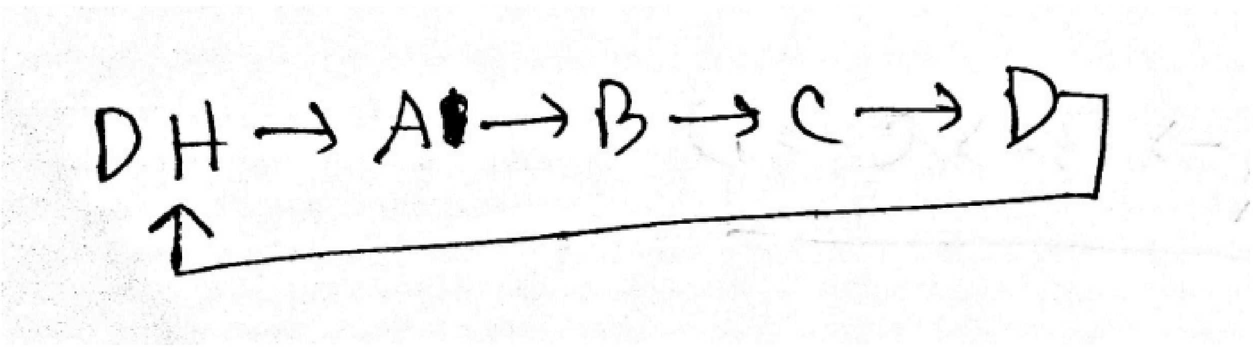
4. **Non-Dummy Headed Doubly Circular Linked List:**



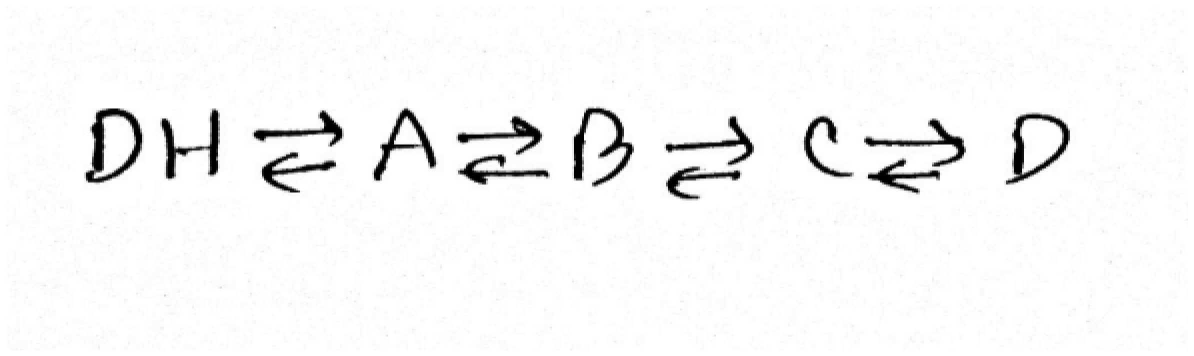
5. **Dummy Headed Singly Linear Linked List:**



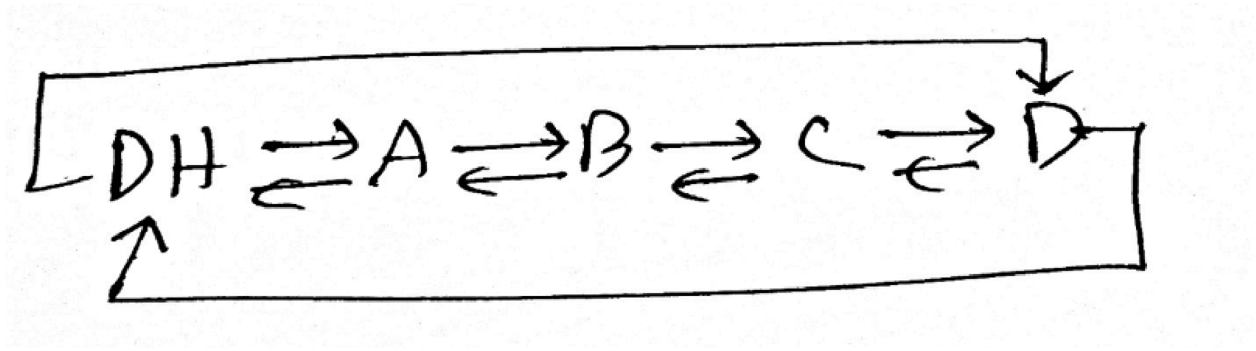
6. **Dummy Headed Singly Circular Linked List:**



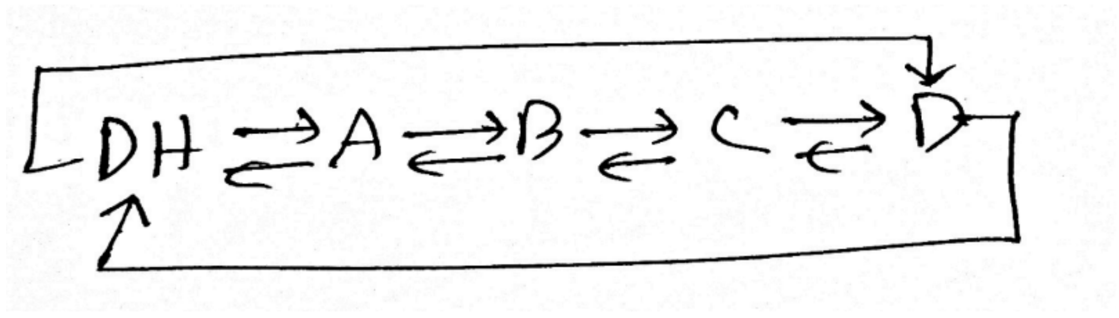
7. **Dummy Headed Doubly Linear Linked List:**



8. **Dummy Headed Doubly Circular Linked List:**



Dummy Headed Doubly Circular Linked List Operations:



To practice we will learn about the dummy-headed doubly circular linked list. As the title suggests, the list will have a dummy head, doubly connection, and circular structure. For this reason, the Node class will be

```
class DoublyNode:
        def __init__(self, elem, next, prev):
                self.elem = elem
                self.next = next # To store the next node's reference.
                self.prev = prev # To store the previous node's reference.
```

Creation:

It is similar to creating a linked list. The difference is that we need to ensure the connections in a proper way.

The below function takes an array and creates a dummy-headed doubly circular linked list.

```python
def createList(a):
  dh = DoublyNode(None, None, None)
  dh.next = dh
  dh.prev = dh
  tail = dh

  for i in range(len(a)):
    n = DoublyNode(a[i], dh, tail)
    tail.next = n
    tail = tail.next
    dh.prev = tail

  return dh
```

Iteration:
It is similar to the previous linked list. The difference is that it will start from the next item of the dummy head and run till the pointer comes back to the dummy head.

```python
def iteration(dh):
  temp = dh.next
  while temp != dh:
    print(temp.elem)
    temp = temp.next
```

NodeAt:
The nodeAt method is similar to the previous one. The difference is that the dummy head does not represent any index as it does not have any value.

```python
def nodeAt(dh, idx):
    temp = dh.next
    c = 0
    while temp != dh:
        if c == idx:
            return temp
        c += 1
        temp = temp.next
    return None # Invalid Index
```

Note that, if you want to count the number of total nodes, you should ignore the dummy head too.

Insertion:

To insert a new node in the list, you need the reference to the predecessor to link in the new node. Unlike for a singly-linked linear list, there is no "special" case here, since there is always a valid predecessor node available, thanks to the dummy head.

```python
def insertion(dh, elem, idx):
  # Assuming the idx is valid
  node_to_insert = DoublyNode(elem, None, None)
  indexed_node = nodeAt(dh, idx) # Retriving the node at that index
  prev_node = indexed_node.prev # There will always be a previous node
  # Change the connection
  # Observe that no special case is needed
  node_to_insert.next = indexed_node
  node_to_insert.prev = prev_node
  prev_node.next = node_to_insert
  indexed_node.prev = node_to_insert
```

Removal:

Removing an element from the list is done by removing the node that contains the element. Just like inserting a new node in a list, removing requires that you have the reference to the predecessor node. Since we're using a doubly-linked list, finding a predecessor of a node is trivial — it's n.prev. And thanks to the dummy head, there is no "special" case here as well.

```python
def removal(dh, idx):
  # Assuming the idx is valid
  node_to_remove = nodeAt(dh, idx)
  prev_node = node_to_remove.prev
  next_node = node_to_remove.next
  # Change the connection
  # No special case is needed
  prev_node.next = next_node
  next_node.prev = prev_node
  node_to_remove.next = None
  node_to_remove.prev = None
  return node_to_remove.elem # Returning the removed element
```