```
# linked nodes based code for implementation of LIST basics
class doublyLinkedList:
    class Node:
        def __init__(self,data=None):
            self.data=data
            self.next=None
            self.prev=None

    class DLLIterator:
        def __init__(self, node, list):
            self.current = node
            self.list = list
        def __eq__( self, rhs ):
            return self.current == rhs.current
        def __ne__( self, rhs ):
            return self.current != rhs.current
        def getObject( self ):
            return self.current.data
        def __iter__(self):  # this is not mandatory, why
            return self
        def __next__( self ):
            if self.current.next != self.list.tail:
                cur_node = self.current
                self.current = self.current.next
                return self
            else:
                raise StopIteration

    class DLLRIterator:
        def __init__(self, node, list):
            self.current = node
            self.list = list
        def __eq__( self, rhs ):
            return self.current == rhs.current
        def __ne__( self, rhs ):
            return self.current != rhs.current
        def getObject( self ):
            return self.current.data
        def __iter__(self):  # this is mandatory, why
            return self
        def __next__( self ):
            if self.current.prev != self.list.head:
                cur_node = self.current
                self.current = self.current.prev
                return self
            else:
                raise StopIteration
```

```
# array based version of linked nodes based code in doubly-LL5.py
class ArrayList:
#    class Node: not required for array based list, delete this line




    class ALIterator:
        def __init__(self, index=0, list=None):
            self.current = index-1
            self.list = list
        def __eq__( self, rhs ):
            return self.current == rhs.current
        def __ne__( self, rhs ):
            return self.current != rhs.current
        def getObject( self ):
            return self.list.Array[self.current]
        def __iter__(self):  # this is not mandatory, why
            return self
        def __next__( self ):
            if self.current != self.list.ALSize-1:
                self.current = self.current + 1
                return self
            else:
                raise StopIteration


    class ALRIterator:
        def __init__(self, index, list=None):
            self.current = index+1
            self.list = list
        def __eq__( self, rhs ):
            return self.current == rhs.current
        def __ne__( self, rhs ):
            return self.current != rhs.current
        def getObject( self ):
            return self.list.Array[self.current]
        def __iter__(self):  # this is mandatory, why
            return self
        def __next__( self ):
            if self.current != 0:
                self.current = self.current - 1
                return self
            else:
                raise StopIteration
```

```
    def __init__(self):
        self.head=self.Node()
        self.tail=self.Node()
        self.head.next=self.tail
        self.tail.prev=self.head

    def __iter__(self):
        return self.DLLIterator(self.head, self)

    def __reversed__(self):
        return self.DLLRIterator(self.tail, self)

    def begining(self):
        return self.DLLIterator(self.head, self)

    def end(self):
        return self.DLLIterator(self.tail.prev, self)

    def rbegining(self):
        return self.DLLRIterator(self.tail, self)

    def rend(self):
        return self.DLLRIterator(self.head.next, self)
```

```
    def append(self,o):
        t=self.Node(o)
        t.next=self.tail
        t.prev=self.tail.prev
        self.tail.prev.next=t
        self.tail.prev=t

    def remove(self, index):
        cur = None
        if type(index) is int:
            cur=self.head.next
            i=0
```

```
    def __init__(self, ARSize = 100):
        self.ARSize=ARSize
        self.ALSize=0
        self.Array=[None]*self.ARSize


    def __iter__(self):
        return self.ALIterator(0, self)

    def __reversed__(self):
        return self.ALRIterator(self.ALSize-1, self)

    def begining(self):
        return self.ALIterator(0, self)

    def end(self):
        return self.ALIterator(self.ALSize, self)

    def rbegining(self):
        return self.ALRIterator(self.ALSize-1, self)

    def rend(self):
        return self.ALRIterator(-1, self)

    def isFull(self):
        return self.ALSize == self.ARSize

    def isEmpty(self):
        return self.ALSize == 0

    def size(self):
        return self.ALSize


    def append(self,o):
        if not self.isFull():
            self.Array[self.ALSize] = o
            self.ALSize = self.ALSize + 1


    def remove(self, index):
        if type(index) is self.ALIterator:
            index = index.current
        if index < 0 or index >= self.ALSize:
            raise Exception("Invalid Index")
```

```
            while i!=index and cur!=self.tail:
                i=i+1
                cur=cur.next
            if cur==self.tail:
                raise Exception
        elif type(index) is self.DLLIterator:
            cur = index.current
        cur.prev.next=cur.next
        cur.next.prev=cur.prev
        del cur

    def Display(self):
        cur=self.head.next
        while cur!=self.tail:
            print(cur.data, end=" ")
            cur=cur.next
        print()

def main():
    # Create a new Doubly Linked List
    dll = doublyLinkedList()
    # Insert the element to empty list
    dll.append(10)
    # Insert the element at the end
    dll.append(20)
    dll.append(30)
    dll.append(70)
    dll.append(50)
    dll.append(60)
    dll.append(80)
    dll.append(90)
    dll.append(40)
    # Display Data
    print("Display 10 items")
    dll.Display()
    # Delete elements from start
    dll.remove(0)
    # Delete elements from end
    dll.remove(0)
    # Display Data
    print("Display without (removed) first two items")
    dll.Display()
    print()

    print("OUTPUT of 'for d in dll:'")
    for d in dll:
        print(d.getObject())
```

```
        j = index
        while j < self.ALSize:
            self.Array[j] = self.Array[j+1]
            j = j + 1
        self.ALSize = self.ALSize - 1


    def set(self, index, o):
        pass
        # assign o at index in array

    def Display(self):
        j = 0
        while j < self.ALSize:
            print(self.Array[j])
            j = j + 1
        print()

def main():
    # Create a new Doubly Linked List
    AL = ArrayList()
    # Insert the element to empty list
    AL.append(10)
    # Insert the element at the end
    AL.append(20)
    AL.append(30)
    AL.append(70)
    AL.append(50)
    AL.append(60)
    AL.append(80)
    AL.append(90)
    AL.append(40)
    # Display Data
    print("Display 10 items")
    AL.Display()
    # Delete elements from start
    AL.remove(0)
    # Delete elements from end
    AL.remove(0)
    # Display Data
    print("Display without (removed) first two items")
    AL.Display()
    print()

    print("OUTPUT of 'for d in AL:'")
    for d in AL:
        print(d.getObject())
```

```
    print()

    print("OUTPUT of 'for d in reversed(dll):'")
    for d in reversed(dll):
        print(d.getObject())
    print()

    print("OUTPUT through manual next calls")
    itm1 = iter(dll)
    itm2 = dll.begining()

    print("first through itm1: " + str(next(itm1).getObject()))
    print("next  through itm1: " + str(next(itm1).getObject()))
    print("first through itm2: " + str(next(itm2).getObject()))
    print("next  through itm2: " + str(next(itm2).getObject()))
    print("next  through itm2: " + str(next(itm2).getObject()))
    print("next  through itm1: " + str(next(itm1).getObject()))
    print("Removing current data")
    dll.remove(itm1)
    print("Removed")
    print("next  through itm2: " + str(next(itm2).getObject()))
    print("next  through itm2: " + str(next(itm2).getObject()))
    print("next  through itm2: " + str(next(itm2).getObject()))
    # due to remove above, print("next  through itm2: " +
    print("next  through itm2: " + str(next(itm2).getObject()))
    # due to remove above, print("next  through itm2: " +
str(next(itm2).getObject()))
    print("next  through itm1: " + str(next(itm1).getObject()))

    print()

    print("OUTPUT through beginning and end")
    i = dll.begining()
    while i != dll.end():
        i = next(i)
        print(i.getObject())
    print()

    print("OUTPUT through rbeginning and rend")
    i = dll.rbegining()
    while i != dll.rend():
        t = next(i)
        print(t.getObject())
    print()

main()
```

```
    print()

    print("OUTPUT of 'for d in reversed(AL):'")
    for d in reversed(AL):
        print(d.getObject())
    print()

    print("OUTPUT through manual next calls")
    itm1 = iter(AL)
    itm2 = AL.begining()

    print("first through itm1: " + str(next(itm1).getObject()))
    print("next  through itm1: " + str(next(itm1).getObject()))
    print("first through itm2: " + str(next(itm2).getObject()))
    print("next  through itm2: " + str(next(itm2).getObject()))
    print("next  through itm2: " + str(next(itm2).getObject()))
    print("next  through itm1: " + str(next(itm1).getObject()))
    print("Removing current data")
    AL.remove(itm1)
    print("Removed")
    print("next  through itm2: " + str(next(itm2).getObject()))
    print("next  through itm2: " + str(next(itm2).getObject()))
    print("next  through itm2: " + str(next(itm2).getObject()))
    # due to remove above, print("next  through itm2: " +
    print("next  through itm2: " + str(next(itm2).getObject()))
    # due to remove above, print("next  through itm2: " +
str(next(itm2).getObject()))
    print("next  through itm1: " + str(next(itm1).getObject()))

    print()

    print("OUTPUT through beginning and end")
    i = AL.begining()
    while i != AL.end():
        i = next(i)
        print(i.getObject())
    print()

    print("OUTPUT through rbeginning and rend")
    i = AL.rbegining()
    while i != AL.rend():
        t = next(i)
        print(t.getObject())
    print()

main()
```