In [1]:

```
'''
Linked List (LList) ADT

LList()           -> create an empty list
__len__           -> returns its size
is_empty()        -> returns whether the list is empty or not
add_first(obj)    -> insert an item at the beginning of the list
add_last(obj)     -> insert an item at the end of the list
add(pos, obj)     -> insert an item at the given position
o = get(pos)      -> get the item at the given position
removeAt(pos)     -> remove the item at the given position
remove(object)    -> remove an item with the given object
'''

class LList:
    ## Start: Nested Node class, not visible from outside ----------------------
    class _Node:
        __slots__ = '_element', '_next'    # Optional: assign memory space for the member variables
, faster!

        def __init__(self):
            self._element = None
            self._next = None

        def __init__(self, element, nxt):
            self._element = element
            self._next = nxt
    ## End: Nested Node class ----------------------

    __slots__ = '_head', '_tail', '_size'

    def __init__(self):
        self._head = None
        self._tail = None
        self._size = 0

    def __len__(self):
        return self._size

    def __str__(self):
        obj = self._head
        ret_str = f'{len(self)}: ['
        while obj != None:
            ret_str += str(obj._element)
            obj = obj._next
            if obj != None:
                ret_str += ','

        ret_str += ']'
        return ret_str
    def __repr__(self):
        return self.__str__()

    def is_empty(self):
        return len(self) == 0

    def add_first(self, obj):
        new_node = self._Node(obj, self._head)
        self._head = new_node
        self._size += 1

        if self._size == 1:
            self._tail = self._head

    # Add element at the tail of the list
    def add_last(self, obj):
        new_node = self._Node(obj, None)

        if len(self) == 0:
            self._head = new_node
            self._tail = new_node
        else:
```

```
            self._tail._next = new_node
            self._tail = new_node

        self._size += 1

    # Add an element at the given position
    def add(self, pos, obj):
        # Three cases: insert at the head, or insert at the tail, or insert at the middle of the l
ist.
        if pos == 0:
            self.add_first(obj)
        elif pos == len(self):
            self.add_last(obj)
        else:
            prev = None
            current = self._head
            for i in range(0, pos):
                prev = current
                current = current._next

                if current == None:
                    raise IndexError

            new_node = self._Node(obj, current)
            prev._next = new_node
            self._size += 1

    def get(self, pos):
        current = self._head
        for i in range(0, pos):
            current = current._next
            if current == None:
                raise IndexError
        return current._element

    def removeAt(self, pos):
        # Three cases: remove at the head, or remove at the tail, or insert at the middle of the l
ist.
        if pos < 0 or pos >= len(self):
            raise IndexError

        prev = self._head
        current = self._head

        for i in range(0, pos):
            prev = current
            current = current._next

        if current == self._head:
            self._head = current._next
        if current == self._tail:
            self._tail = prev

        prev._next = current._next
        self._size -= 1

    def remove(self, obj):
        prev = None
        current = self._head

        while current != None:
            if current._element == obj:
                if current == self._head:
                    self._head = current._next
                if current == self._tail:
                    self._tail = prev
                if prev != None:
                    prev._next = current._next

                self._size -= 1
                return

            prev = current
            current = current._next

        raise LookupError
```

In [2]:

```
# Test: edge cases

a = LList()
a.add(0, 'Susan')
print(a)
a.add(0, 'Jenny')
print(a)
a.add(2, 'Mike')
print(a)
a.add(3, 'Freddy')
print(a)
a.add_first('Violet')
print(a)
a.add_last('Jeff')
print(a)
a.add(3, 'Josh')
print(a)

a.remove('Susan')
print(a)
a.remove('Violet')
print(a)
a.remove('Jeff')
print(a)

print(a.get(1))
a.removeAt(1)
print(a)

print(a.get(2))
a.removeAt(2)
print(a)

print(a.get(1))
a.removeAt(1)
print(a)

print(a.get(0))
a.removeAt(0)
print(a)
```

```
1: [Susan]
2: [Jenny,Susan]
3: [Jenny,Susan,Mike]
4: [Jenny,Susan,Mike,Freddy]
5: [Violet,Jenny,Susan,Mike,Freddy]
6: [Violet,Jenny,Susan,Mike,Freddy,Jeff]
7: [Violet,Jenny,Susan,Josh,Mike,Freddy,Jeff]
6: [Violet,Jenny,Josh,Mike,Freddy,Jeff]
5: [Jenny,Josh,Mike,Freddy,Jeff]
4: [Jenny,Josh,Mike,Freddy]
Josh
3: [Jenny,Mike,Freddy]
Freddy
2: [Jenny,Mike]
Mike
1: [Jenny]
Jenny
0: []
```

In [3]:

```
b = LList()
b.add_first('a')
print(b)

c = LList()
c.add_last('a')
print(c)
```

```
1: [a]
1: [a]
```

In [ ]: