linkedListBasic.py 코드

```
from listNode import ListNode

class linkedListBasic:
    def __init__(self):
        '''
        더미 헤드 __head 를 만듦
        항목 갯수를 0 으로 초기화
        '''
        self.__head = ListNode('dummy', None)
        self.__numItems = 0
    def insert(self, i:int, newItem):
        '''
        i 가 list 의 item 갯수보다 적으면,
        i 번째에 newItem 값을 넣어라
        그렇지 않으면 error 출력
        '''
        if i >= 0 and i <= self.__numItems:
            prev = self.__getNode(i - 1)
            newNode = ListNode(newItem, prev.next)
            prev.next = newNode
            self.__numItems += 1
        else:
            print("index", i, "out of bound in insert()")

    def append(self, newItem):
        '''
        newItem 을 list 의 맨 끝에 삽입해라.
        이전 node 의 next 속성에 newNode 를 부여.
        __numItems 를 1 증가
        '''
        prev = self.__getNode(self.__numItems - 1)
        newNode = ListNode(newItem, prev.next)
        prev.next = newNode
        self.__numItems += 1
    def pop(self, i:int):
        '''
        i 가 0 이상 list 길이 이하이면
        prev 의 next 속성에 다음 노드(next)를 부여.
        __numItems 를 1 감소
        '''
        if (i >= 0 and i <= self.__numItems - 1):
            prev = self.__getNode(i - 1)
            curr = prev.next
            prev.next = curr.next
            retItem = curr.item
            self.__numItems -= 1
            return retItem
        else:
            return None

    def remove(self, x):
        '''
        x 를 찾음. x 가 list 에 없으면 아무것도 안함
        x 가 list 에 있으면
        이전 node 의 next 속성에 다음 노드를 부여
        __numitems 를 1 감소
        '''
        (prev, curr) = self.__findNode(x)
        if curr != None:
            prev.next = curr.next
```

```
            self.__numItems -= 1
            return x
        else:
            return None

    def get(self, i:int):
        '''
        i 번째의 item 을 return 하는 함수.
        list 가 비었다면 None 을 return.
        __getNode(i) 함수로 i 번째 노드를 가져오고,
        i 번째 노드의 item 을 return'''
        if self.isEmpty():
            return None
        if (i >= 0 and i <= self.__numItems - 1):
            return self.__getNode(i).item
        else:
            return None

    def index(self, x) -> int:
        '''
        x 값의 index 를 return 하는 함수.
        curr 변수를 첫 번째 노드로 설정
        노드의 갯수 동안 curr 의 값이 x 인지 확인
        x 인 curr 을 찾으면 그 index 를 return
        아니면 -2(없는 인덱스)를 return
        '''
        curr = self.__head.next
        for index in range(self.__numItems):
            if curr.item == x:
                return index
            else:
                curr = curr.next
        return -2
    def isEmpty(self) -> bool:
        return self.__numItems == 0
    def size(self) -> int:
        return self.__numItems
    def clear(self):
        self.__head = ListNode('dummy', None)
        self.__numItems = 0
    def count(self, x) -> int:
        cnt = 0
        curr = self.__head.next
        while curr != None:
            if curr.item == x:
                cnt += 1
                curr = curr.next
        return cnt
    def extend(self, a):
        for index in range(a.size()):
            self.append(a.get(index))
    def copy(self):
        a = linkedListBasic()
        for index in range(self.__numItems):
            a.append(self.get(index))
        return a
    def reverse(self):
        a = linkedListBasic()
        for index in range(self.__numItems):
            a.insert(0, self.get(index))
        self.clear()
        for index in range(a.size()):
            self.append(a.get(index))
```

```
    def sort(self) -> None:
        a = []
        for index in range(self.__numItems):
            a.append(self.get(index))
        a.sort()
        self.clear()
        for index in range(len(a)):
            self.append(a[index])
    def __findNode(self, x) -> (ListNode, ListNode):
        prev = self.__head
        curr = prev.next
        while curr != None:
            if curr.item == x:
                return (prev, curr)
    def __getNode(self, i:int) -> ListNode:
        curr = self.__head
        for index in range(i+1):
            curr = curr.next
        return curr
    def printList(self):
        curr = self.__head.next
        while curr != None:
            print(curr.item, end = ' ')
            curr = curr.next
        print()
    def __iter__(self):
        self.position = 0
        return self

    def __next__(self):
        if self.position >= self.__numItems:
            raise StopIteration
        result = self.get(self.position)
        self.position += 1
        return result
```

linkedListBasic의 메서드 부분은 교재를 참고하여 작성하였습니다.

Iterator 은 직접 구현하였습니다.

linkedListBasics.py Iterator 해설.

```
    def __iter__(self):
        self.position = 0
        return self

    def __next__(self):
        if self.position >= self.__numItems:
            raise StopIteration
        result = self.get(self.position)
        self.position += 1
        return result
```

Iterator 부분은 __iter__(self) 메서드와 __next__(self) 메서드로 구성되어 있습니다.

__iter__(self) 메서드는 Iteration을 시작할 때 생성자로 사용됩니다. 여기에서 속성 self.position을 정의하여 Iterate할 때의 위치를 정의하였습니다.

__next__(self) 메서드에서 self.position이 리스트 항목 수보다 같거나 크면 StopIteration 오류를 raise하여 Iteration을 종료시킵니다.

그렇지 않은 경우, self.position번째 item을 return하고, self.position을 1 증가시킵니다.

출력 결과

```
PS C:\Users\kimgu\Desktop\자료구조\연결리스트> & C:/Users/kimgu/anaconda3/python.exe c:/Users/kimgu/Desktop/자료구조/연결리스트/main.py
Amy
Kevin
Mary
David
Amy David Kevin Mary Rose
```

circularLinkedListBasic.py 코드

```
from listNode import ListNode
class circularLinkedListBasic:
    def __init__(self):
        '''
        항목 갯수를 0 으로 초기화
        '''
        self.__tail = ListNode('dummy', self.__tail)
        self.__numItems = 0
    def insert(self, i:int, newItem) -> None:
        '''
        i 가 list 의 item 갯수보다 적으면,
        i 번째에 newItem 값을 넣어라
        그렇지 않으면 error 출력
        '''
        if (self.__numItems == i):
            self.append(i)
        elif i >= 0 and i < self.__numItems:
            prev = self.__prevNode(i)
            newNode = ListNode(newItem, prev.next)
            prev.next = newNode
            self.__numItems += 1
        else:
            print("index", i, "out of bound in insert()")

    def append(self, newItem):
        '''
        newItem 을 list 의 맨 끝에 삽입해라.
        이전 node 의 next 속성에 newNode 를 부여.
        새롭게
        __numItems 를 1 증가
        '''
        if self.__numItems == 0:
            self.__tail.next = ListNode(newItem, self.__tail.next)
            self.__numItems += 1
        else:
            first = self.__tail.next.next
            self.__tail.next.next = ListNode(newItem, first)
            self.__numItems += 1
    def pop(self, i:int=None):
        '''
        i 가 0 이상 list 길이 이하이면
        prev 의 next 속성에 다음 노드(next)를 부여.
        __numItems 를 1 감소
        '''
        if i == None or i == self.__numItems - 1:
            retItem = self.__tail.next.item
            first = self.__tail.next.next
            prev = self.__getNode(self.__numItems - 2)
            prev.next = first
            self.__numItems -= 1
            return retItem
        if (i >= 0 and i < self.__numItems - 1):
            prev = self.__getNode(i - 1)
            curr = prev.next
            prev.next = curr.next
            retItem = curr.item
            self.__numItems -= 1
            return retItem
```

```
        else:
            return None

    def remove(self, x):
        '''
        x를 찾음. x가 list에 없으면 아무것도 안함
        x가 list에 있으면
        이전 node의 next 속성에 다음 노드를 부여
        __numitems를 1 감소
        '''
        first = self.__tail.next.next
        (prev, curr) = self.__findNode(x)
        if curr != None:
            if curr == self.__tail.next:
                return self.pop()
            prev.next = curr.next
            self.__numItems -= 1
            return x
        else:
            return None

    def get(self, i:int):
        '''
        i번째의 item을 return하는 함수.
        list가 비었다면 None을 return.
        __getNode(i) 함수로 i번째 노드를 가져오고,
        i번째 노드의 item을 return'''
        if self.isEmpty():
            return None
        if (i >= 0 and i < self.__numItems):
            return self.__getNode(i).item
        else:
            return None

    def index(self, x) -> int:
        '''
        x 값의 index를 return하는 함수.
        curr 변수를 첫 번째 노드로 설정
        노드의 갯수 동안 curr의 값이 x인지 확인
        x인 curr을 찾으면 그 index를 return
        아니면 -2(없는 인덱스)를 return
        '''
        curr = self.tail.next.next
        for index in range(self.__numItems-1):
            if curr.item == x:
                return index
            else:
                curr = curr.next
        return -2
    def isEmpty(self) -> bool:
        return self.__numItems == 0
    def size(self) -> int:
        return self.__numItems
    def clear(self):
        self.__tail = ListNode('dummy', self.__tail)
        self.__numItems = 0
    def count(self, x) -> int:
        cnt = 0
        first = self.__tail.next.next
        curr = first
        while curr != first:
            if curr.item == x:
                cnt += 1
```

```
            curr = curr.next
        return cnt
    def extend(self, a):
        first = self.__tail.next.next
        self.__tail.next.next = a.copy.getfirst()
        self.__numItems += a.size()
        curr = first.next
        for index in range(self.__numItems-2):
            curr = curr.next
        curr.next = first
        self.__tail.next = curr
    def copy(self):
        a = circularLinkedListBasic()
        for index in range(self.__numItems):
            a.append(self.get(index))
        return a
    def reverse(self):
        a = circularLinkedListBasic()
        for index in range(self.__numItems):
            a.insert(0, self.get(index))
        self.clear()
        for index in range(a.size()):
            self.append(a.get(index))
    def sort(self) -> None:
        a = []
        for index in range(self.__numItems):
            a.append(self.get(index))
        a.sort()
        self.clear()
        for index in range(len(a)):
            self.append(a[index])
    def __findNode(self, x) -> (ListNode, ListNode):
        prev = self.__tail.next
        curr = prev.next
        while curr != self.__tail.next:
            if curr.item == x:
                return (prev, curr)
            else:
                prev = curr; cur = curr.next
        if curr.item == x:
            return (prev, curr)
        else:
            return (None, None)
    def __getNode(self, i:int) -> ListNode:
        curr = self.__tail.next.next
        for index in range(i):
            curr = curr.next
        return curr
    def printList(self):
        curr = self.__tail.next.next
        while curr != self.__tail.next:
            print(curr.item, end = ' ')
            curr = curr.next
        print(curr.item)
    def getfirst(self) -> ListNode:
        return self.__tail.next.next
    def __iter__(self):
        self.position = 0
        return self

    def __next__(self):
        if self.position >= self.__numItems:
            raise StopIteration
        result = self.get(self.position)
```

```
        self.position += 1
        return result
```

__init__함수에서는__tail 속성을 LinkedList("dummy", self.__tail)로 정의하였습니다. __tail.next.next 속성은 리스트의 첫 항목으로 정의하여 다른 메서드를 작성하였습니다. Iterator 부분은 LinkedListBasic 부분과 같이 작성하였습니다.

다만, 충분히 검토하지 못하여 버그가 발생하는 상황입니다.