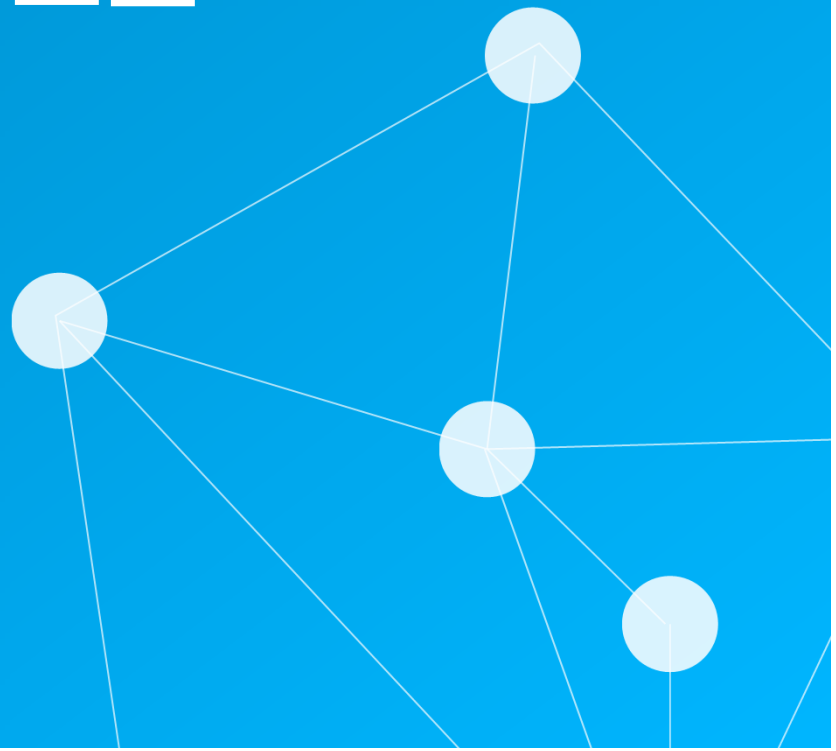

파이썬
자료구조

리스트



리스트란?

- 리스트(list), 선형리스트(linear list)
  - 순서를 가진 항목들의 모임
  - $L = [item_0,\ item_1,\ item_2,\ \ldots,\ item_{n-1}]$

    0번째 원소 1번째원소 n-1번째 원소

  - 집합: 항목간의 순서의 개념이 없음

리스트 ADT

정의 3.1 List ADT

데이터: 같은 유형의 요소들의 순서 있는 모임

연산

- List(): 비어 있는 새로운 리스트를 만든다.
- insert(pos, e): pos 위치에 새로운 요소 e를 삽입한다.
- delete(pos): pos 위치에 있는 요소를 꺼내고(삭제) 반환한다.
- isEmpty(): 리스트가 비어있는지를 검사한다.
- getEntry(pos): pos 위치에 있는 요소를 반환한다.
- size(): 리스트안의 요소의 개수를 반환한다.
- clear(): 리스트를 초기화한다.
- find(item): 리스트에서 item이 있는지 찾아 인덱스를 반환한다.
- replace(pos, item): pos에 있는 항목을 item으로 바꾼다.
- sort(): 리스트의 항목들을 어떤 기준으로 정렬한다.
- merge(lst): 다른 리스트 lst를 리스트에 추가한다.
- display(): 리스트를 화면에 출력한다.
- append(e): 리스트의 맨 뒤에 새로운 항목을 추가한다.

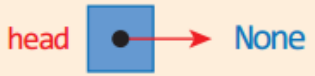
4. 연결리스트로 구현한 리스트

- 연결리스트 구조

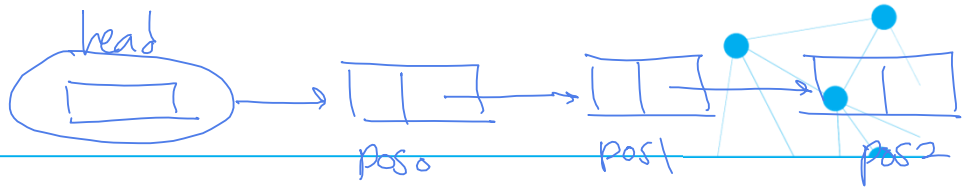
- 노드 클래스: 연결된 스택에서와 동일
- 연결 리스트 클래스

```
class LinkedList:                                  # 연결된 리스트 클래스
    def __init__( self ):
        self.head = None

    def isEmpty( self ): return self.head == None  # 공백상태 검사
    def clear( self ) : self.head = None           # 리스트 초기화
    def size( self ) : ...                         # self.top->self.head로 수정. 코드 동일
    def printList(self) : ...
```



연결 리스트 메소드



- pos번째 노드 반환: getNode(pos) ->반복 이용

```
def getNode(self, pos) :                    # pos번째 노드 반환
    if pos < 0 : return None
    node = self.head;                       # node는 head부터 시작
    while pos > 0 and node != None :        # pos번 반복
        node = node.link                    # node를 다음 노드로 이동
        pos -= 1                            # 남은 반복 횟수 줄임
    return node                             # 최종 노드 반환
```

- pos번째 노드 반환: getNode(pos) ->재귀 이용

```
def getNode(self, pos):
    return self.recurGetNode(self.head, pos)

def recurGetNode(self, ptr, pos):    # ptr이 참조하는 연결리스트에서 pos번째 노드 반환
    if pos < 0:
        return None
    elif pos == 0:
        return ptr                   # pos = 0이면 ptr이 참조하는 노드를 반환
    else:
        return self.recurGetNode(ptr.link, pos-1) # ptr.link가 참조하는 연결리스트에서 pos-1번째 노드 반환
```

연결 리스트 메소드

- getEntry(pos), replace(pos,elem), find(val)

```
def getEntry(self, pos) :                # pos번째 노드의 데이터 반환
    node = self.getNode(pos)             # pos번째 노드
    if node == None : return None        # 찾는 노드가 없는 경우
    else : return node.data              # 그 노드의 데이터 필드 반환
```

연결 리스트 메소드

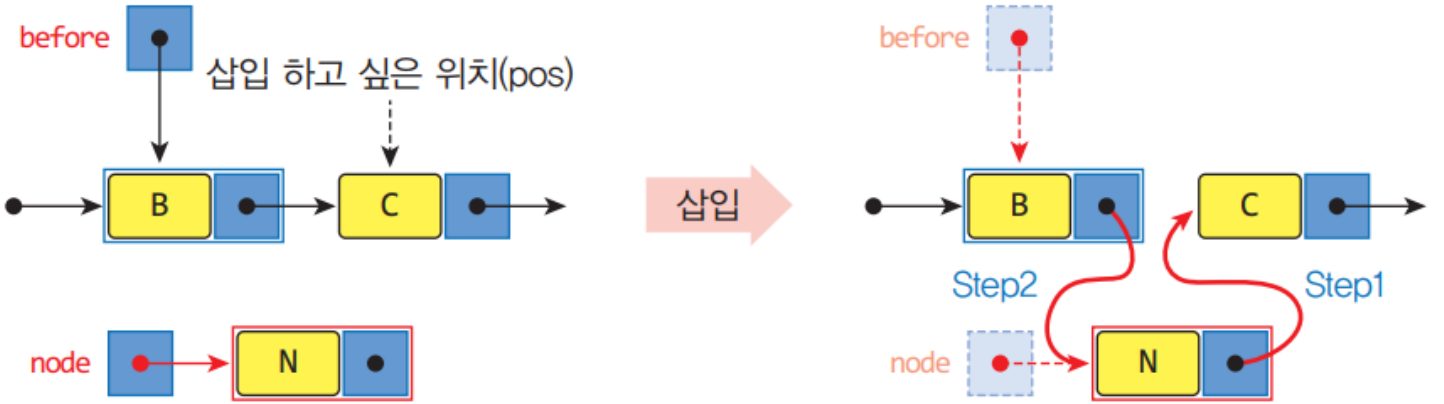
- 노드 개수 반환: size()

```
# iterative version
    def size(self):
        count = 0
        current  = self.head
        while current is not None:
            count += 1
            current = current.link
        return count
```

```
# recursive version
    def nodeCount(self, node):
        if node == None:
            return 0
        else:
            return self.nodeCount(node.link)+1

    def size(self):
        return self.nodeCount(self.head)
```

삽입 연산: insert(pos, elem)



① 노드 N이 노드 C를 가리키게 함: node.link = before.link

② 노드 B가 노드 N을 가리키게 함: before.link = node

```
def insert(self, pos, elem) :
    node = Node(elem)
    before =self.getNode(pos-1) #before 노드 찾음
    if before == None:              # 맨 앞에 삽입
        node.link = self.head
        self.head = node
    else:                           # 중간에 삽입
        node.link = before.link
        before.link = node
```

삭제 연산: delete(pos)

① before의 link가 삭제할 노드의 다음 노드를 가리키도록 함 : bofore.link = before.link.link

```
def delete(self, pos) :
    before = self.getNode(pos-1)                # before 노드를 찾음
    if before == None :                         # 시작노드를 삭제
        if self.head is not None :              # 공백이 아니면
            self.head = self.head.link          # head를 다음으로 이동
    elif before.link != None :                  # 중간에 있는 노드 삭제
        before.link = before.link.link          # Step1
```

출력 : printAll(self) (ADT의 display)

```
# iterative version
    def printList(self):
        node = self.head
        while node in not None:
            print(node.data)
            node = node.link
-----------------------------------------------------
# recursive version
    def printAll(self, node):
        if node is not None:
            print(node.data)
            self.printAll(node.link)

    def printList(self):
        self.printAll(self.head)
```