가천대학교
Gachon University

02

Algorithm

자료구조 기초 : 리스트

2025-03-14

조윤실

목 차

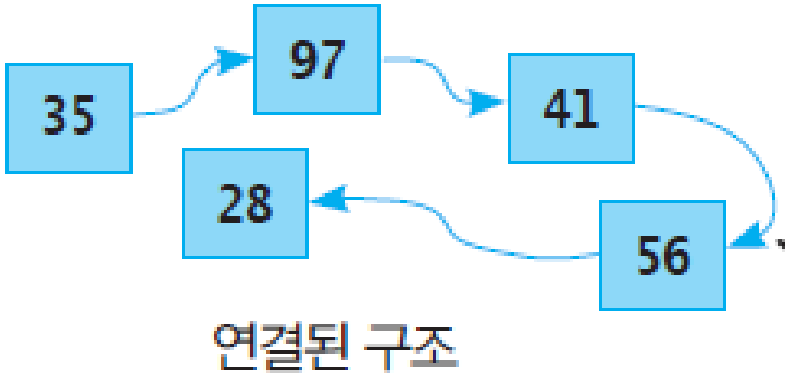
※ 가천대학교 컴퓨터공학과 '알고리즘' 과정

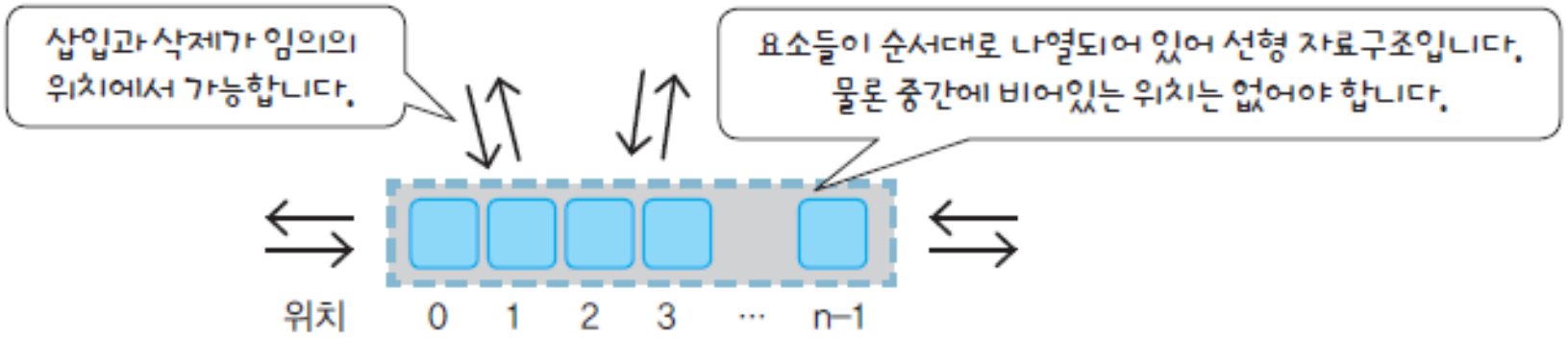
리스트

리스트 개요

리스트(List)

- 리스트란(List or Linear list)
  - 가장 자유로운 선형 자료구조(연결 자료구조)
  - 각 자료는 순서 또는 위치(position)를 가짐
  - 다양한 항목들을 저장, 조회할 수 있음





[그림 출처]: 자료구조와 알고리즘 with 파이썬 by 생능북스

리스트

- 리스트 자료구조의 특징
  - 장점은 동적 메모리 할당이 가능하여 크기가 가변적임
  - 단점은 배열에 비해 노드 접근 시간이 느리고, 추가 메모리 공간(링크 필드)이 필요함
  - 리스트는 다양한 알고리즘과 응용 프로그램에서 활용됨
    - 스택(Stack), 큐(Queue), 연결 리스트, 해싱(Linked List Hashing) 등의 구현에 사용됨
  - [참고]파이썬 리스트 구조

    https://github.com/zpoint/CPython-Internals/blob/master/BasicObject/list/list.md

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

대부분의 프로그래밍 언어에서 리스트 지원

- 리스트 자료구조를 지원하는 프로그램 언어
  - 대부분의 프로그래밍 언어에서 리스트를 지원함

1. **Python:** list

python Copy code

```python
fruits = ['apple', 'banana', 'cherry']
```

2. **Java:** ArrayList (java.util.ArrayList)

java Copy code

```java
ArrayList<String> fruits = new ArrayList<>();
fruits.add("apple");
fruits.add("banana");
fruits.add("cherry");
```

3. **C++:** vector (std::vector)

cpp Copy code

```cpp
vector<string> fruits = {"apple", "banana", "cherry"};
```

4. **JavaScript:** Array

javascript Copy code

```javascript
let fruits = ["apple", "banana", "cherry"];
```

5. **C#:** List (System.Collections.Generic.List)

csharp Copy code

```csharp
List<string> fruits = new List<string>() { "apple", "banana", "cherry" };
```

6. **Ruby:** Array

ruby Copy code

```ruby
fruits = ["apple", "banana", "cherry"]
```

……

파이썬의 리스트 자료구조

- 파이썬 리스트는 어떤 자료형도 Element로 담을 수 있다.
  - 파이썬의 리스트는 연속된 메모리를 사용하는 배열구조
  - 단, 용량이 제한되지 않도록 동적 배열로 구현됨

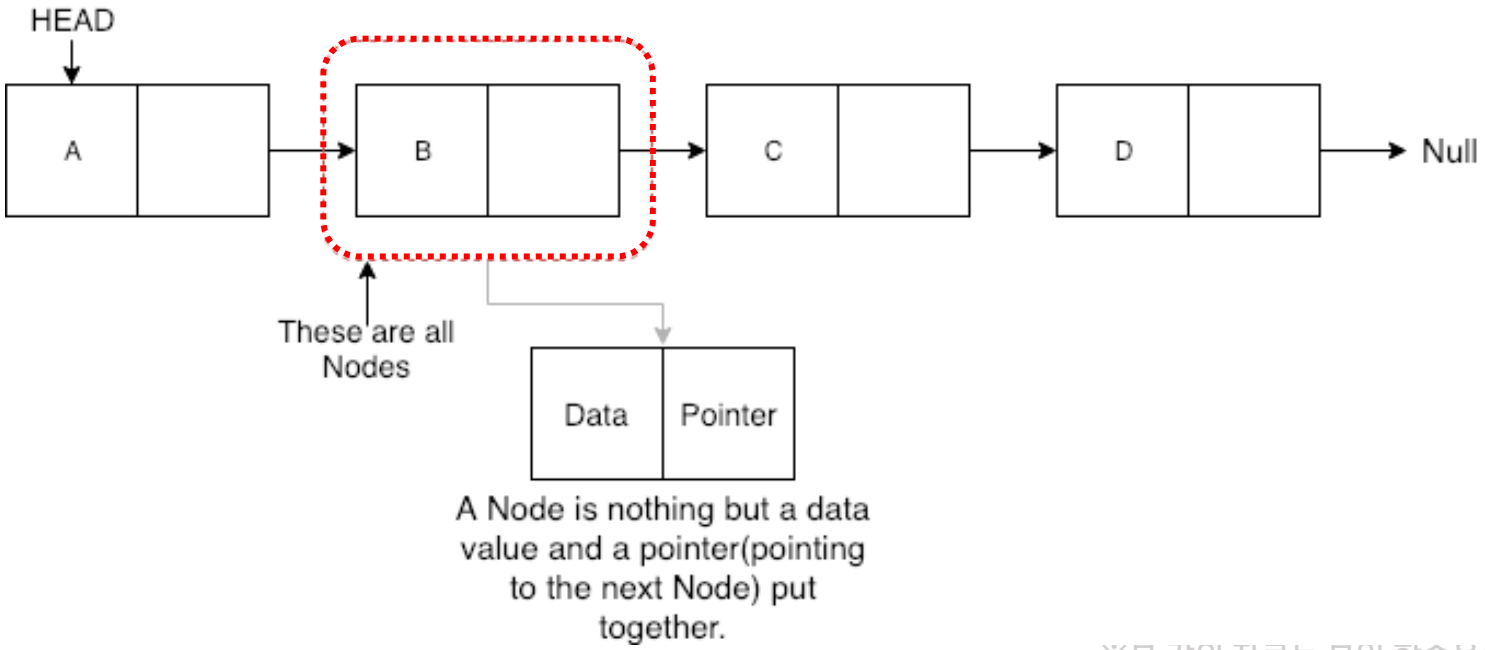
```
def print_number(num):
    print("function number : ", num)

class Text:
    def __init__(self, num):
        self.num = num
    def print_number_method(self):
        print("method number : ", self.num)


a_list = [1, print_number, Text]
```

```
a = [1, 2.2, (3,4), "5", np.array([6,7]), {"key8": 9}, 10]
```

```
a_list = [1, <function print_number at 0x7e7e03089ee0>, <class '__main__.Text'>]
```

리스트의 구조

- 리스트의 구조
  - 순서가 있는 데이터의 모음을 저장하는 자료구조
  - 각 데이터 항목을 노드(Node)라고 부르며 노드들이 연결(link)되어 리스트를 구성함



리스트의 응용

▪ 리스트의 응용 예

- **웹 브라우저의 방문기록 관리기능**

  웹브라우저에서 사용자가 방문한 웹 페이지의 URL을 리스트 형태로 저장하고 관리

- **문서 편집기의 실행 취소/재실행 기능취소**

  문서 편집기문서 편집기에서는 사용자의 편집 작업을 리스트에 저장함. 이를 통해 실행 취소(Undo) 기능과 재실행(Redo) 기능을 구현할 수 있음

- **데이터 관리 및 조작**

  리스트는 데이터 집합을 유연하게 관리하고 조작할 수 있게 해 줌. 데이터베이스 쿼리 결과, 사용자 입력 데이터, 파일에서 읽은 데이터 등을 저장하고, 필요에 따라 추가, 삭제, 정렬, 검색 등의 조작 수행

- **알고리즘 구현**

리스트의 연산

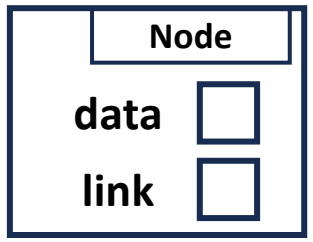
- 리스트의 주요 연산
  - insert(pos, e): pos 위치에 새로운 데이터(요소) 삽입
  - delete(pos): pos 위치에 있는 요소 꺼내서 반환
  - getEntry(pos): pos 위치에 있는 요소를 삭제하지 않고 반환
  - isEmpty(): 리스트가 비어 있는지 여부 반환, True/False 반환
  - isFull(): 리스트가 가득 차 있는지 확인, True/False 반환
  - size(): 리스트에 들어 있는 전체 요소의 수 반환

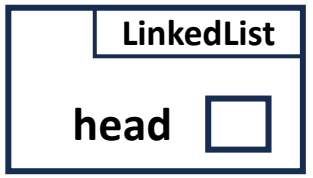
  - 활용이 자유로워 추가적인 다양한 추가 연산이 가능하다.
    - append(e), pop( ), find(e), replace(pos, e), display( ) 등

리스트의 오류 상황

- 인덱스 오류(IndexError)
  - 공백 상태의 리스트에서 삭제 연산을 시도하려고 할 경우(underflow)
  - 존재하지 않는 데이터를 삭제하려고 할 경우(Data Not found)
- 메모리 오류:
  - 너무 큰 리스트를 생성하려고 할 경우(MemoryError)
- 속성/메서드 오류:
  - 존재하지 않는 속성/메서드를 사용할 경우(AttributeError)
- 자료형 오류:
  - 리스트에 허용되지 않는 자료형을 추가할 경우(TypeError)

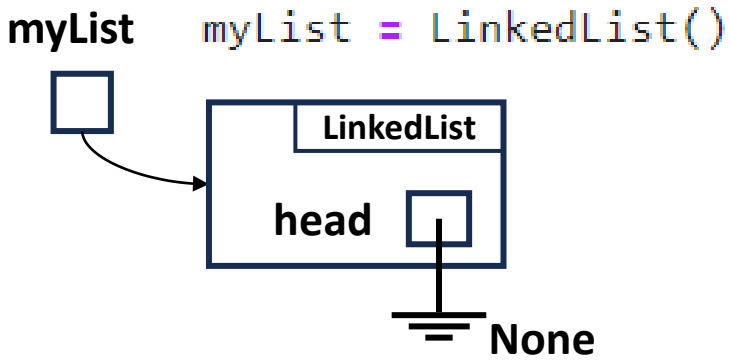
연결 리스트의 동작



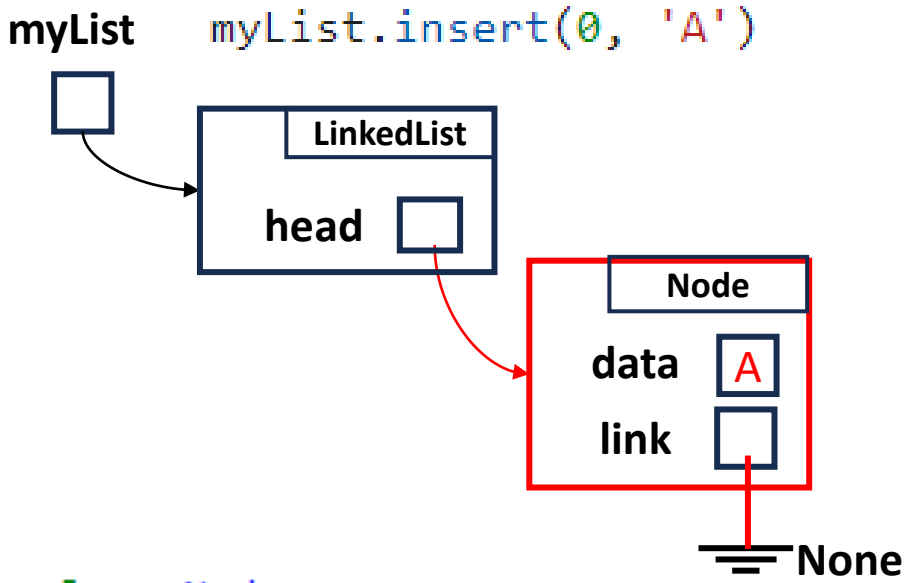



```python
class Node:
    def __init__(self, data):
        self.data = data
        self.link = None

class LinkedList:
    def __init__(self):
        self.head = None
```

```python
myList = LinkedList()
myList.insert(0, 'A')
myList.insert(1, 'B')
myList.insert(1, 'C')
myList.delete(0)
```



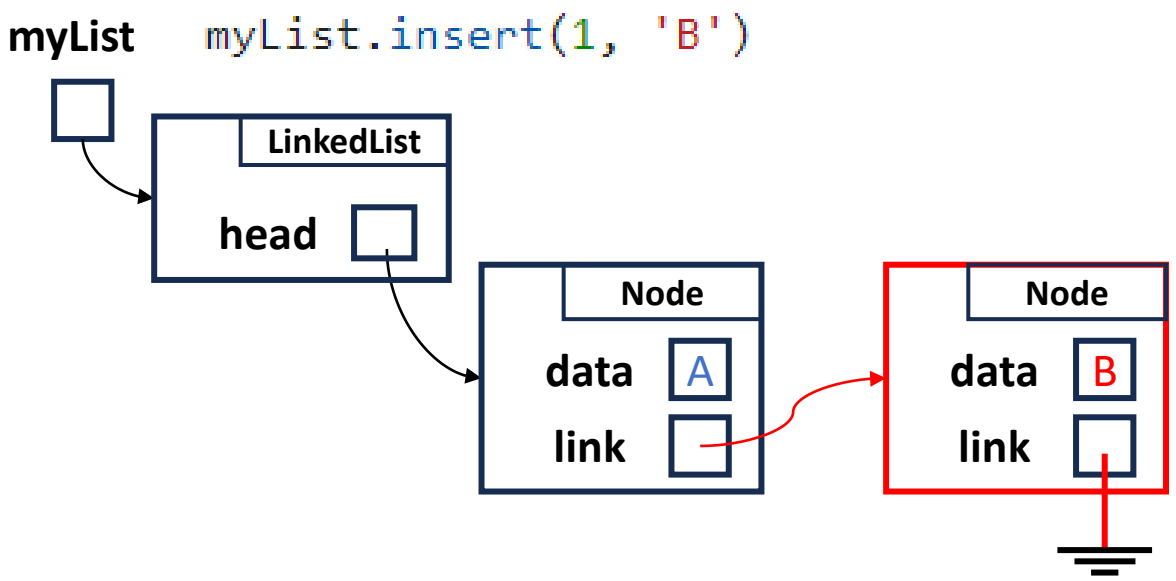
연결 리스트의 동작: insert




```
class Node:
    def __init__(self, data):
        self.data = data
        self.link = None

class LinkedList:
    def __init__(self):
        self.head = None
```

```
def insert(self, pos, e): # 삽입 연산
    new_node = Node(e)
    if pos == 0:
        new_node.link = self.head
        self.head = new_node
        return
```

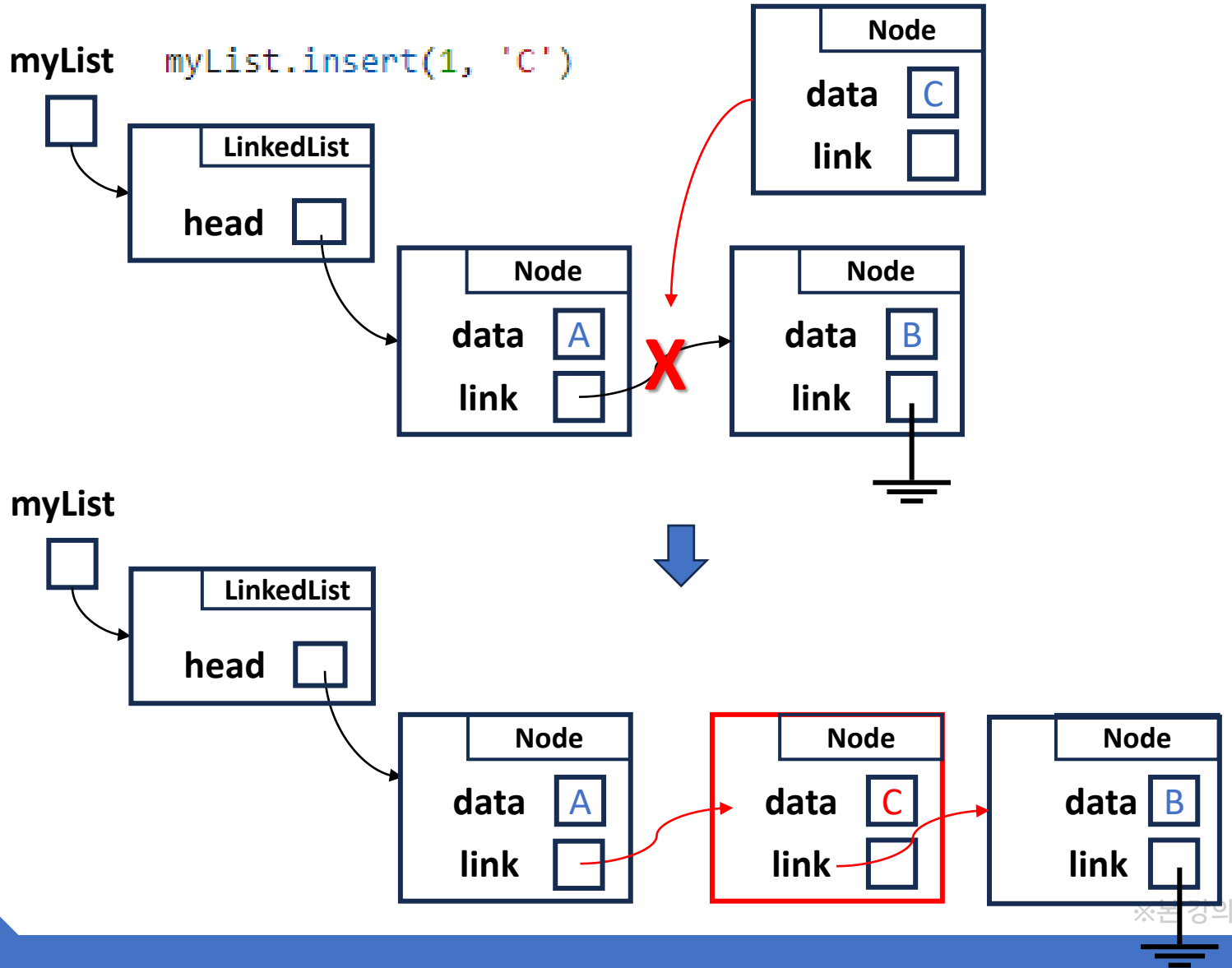
연결 리스트의 동작: insert




```
def insert(self, pos, e): # 삽입 연산
    new_node = Node(e)
    if pos == 0:
        new_node.link = self.head
        self.head = new_node
        return
    current = self.head
    count = 1
    while current and count < pos:
        current = current.link
        count += 1
    if current is None:
        raise IndexError("Index out
    new_node.link = current.link
    current.link = new_node
```

처음부터 노드 탐색 -위치찾기

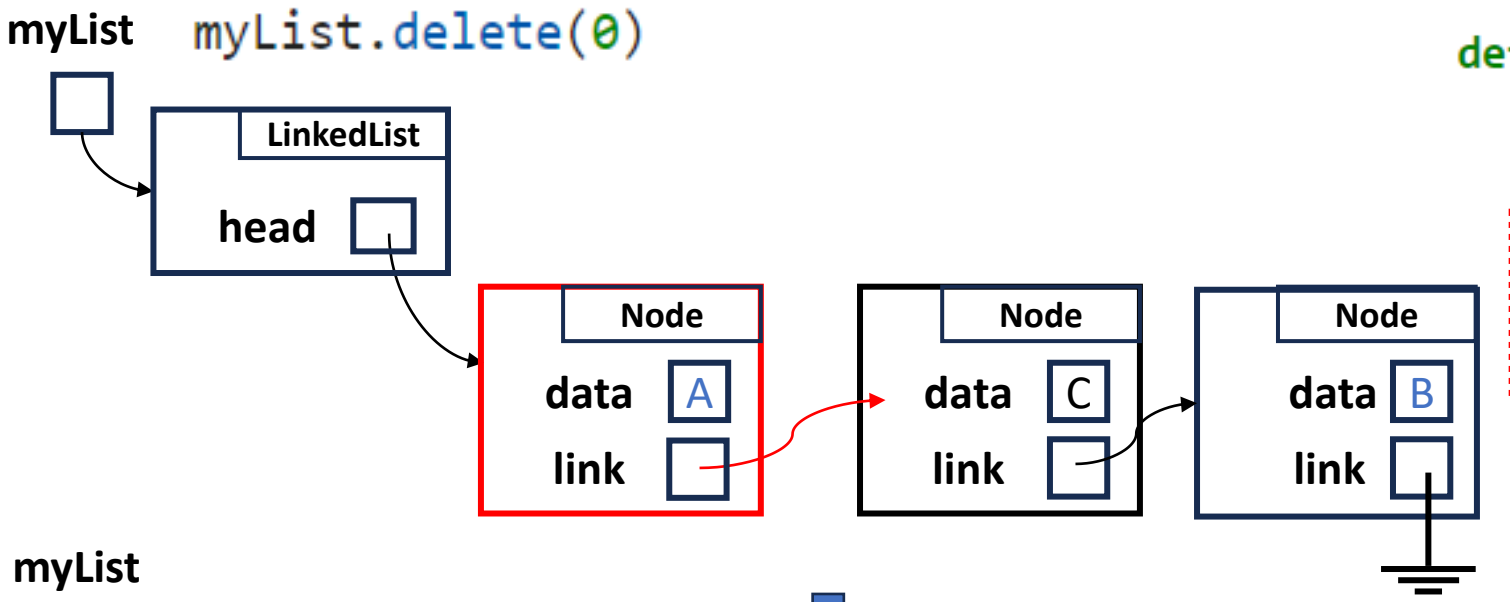
연결 리스트의 동작: insert

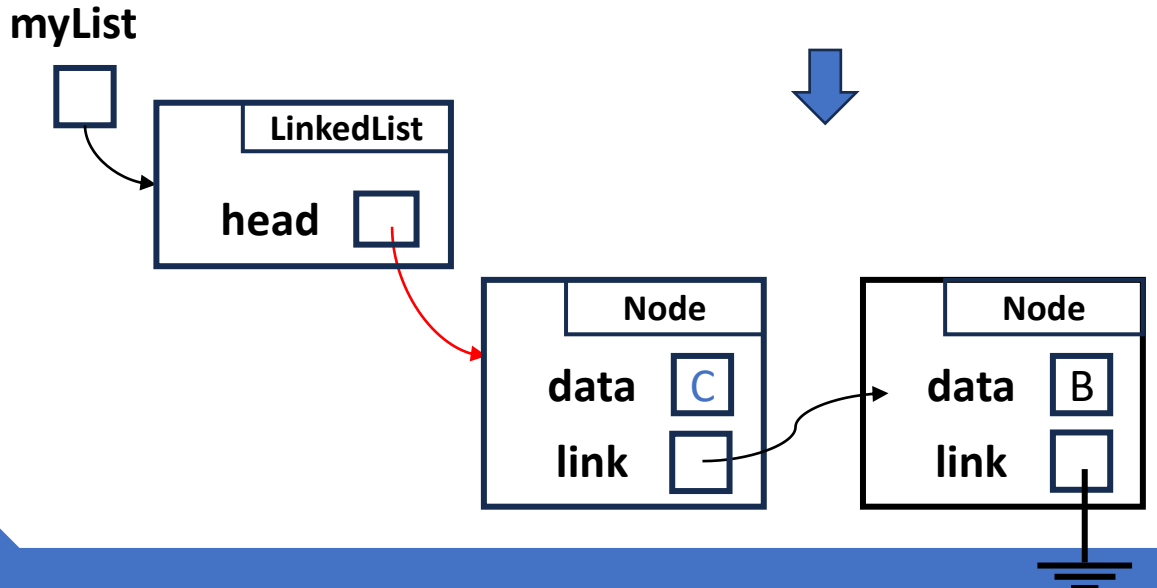



```
def insert(self, pos, e): # 삽입 연산
    new_node = Node(e)
    if pos == 0:
        new_node.link = self.head
        self.head = new_node
        return
    current = self.head
    count = 1
    while current and count < pos:
        current = current.link
        count += 1
    if current is None:
        raise IndexError("Index out
    new_node.link = current.link
    current.link = new_node
```

위치 찾기

연결 리스트의 동작: delete



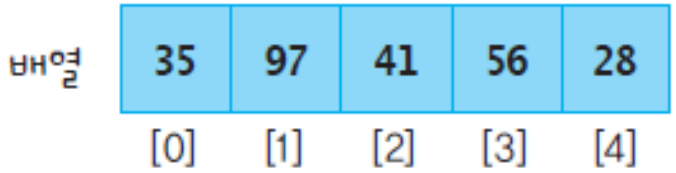



```
def delete(self, pos):          # 삭제 연산
    if self.head is None:
        raise IndexError("List is empty"
    if pos == 0:
        self.head = self.head.link
        return
    current = self.head
    count = 1
    while current.link and count < pos:
        current = current.link
        count += 1
    if current.link is None:
        raise IndexError("Index out of b
    current.link = current.next.link
```

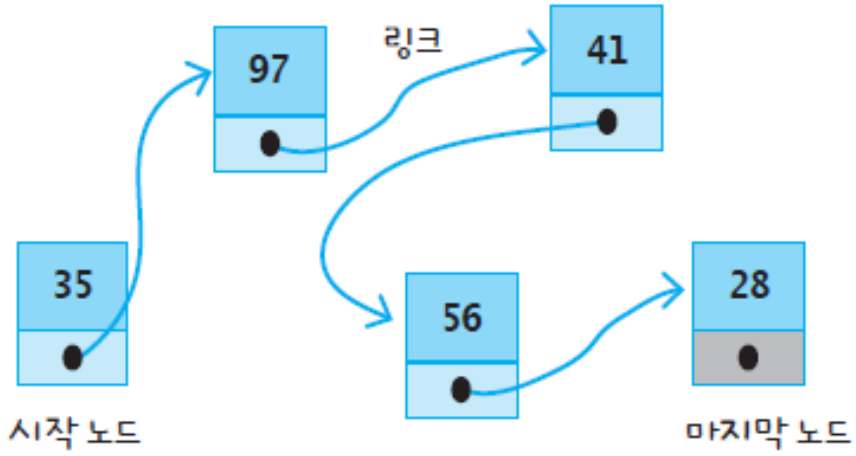
배열 구조 리스트와 연결 구조 리스트

배열 구조 리스트 vs 연결된 구조 리스트

- 배열 구조 리스트
  - 모든 요소의 크기가 같다
  - 연속된 메모리 공간에 있다



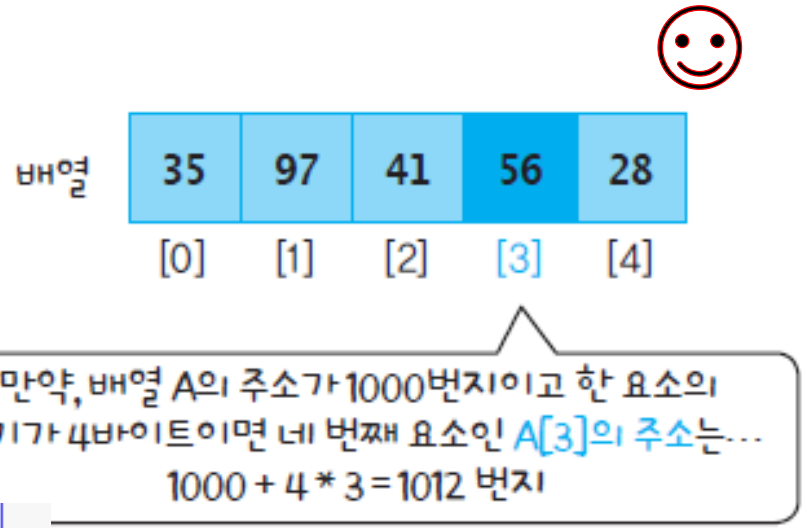
- 연결된 구조 리스트
  - 노드(node) : data + link



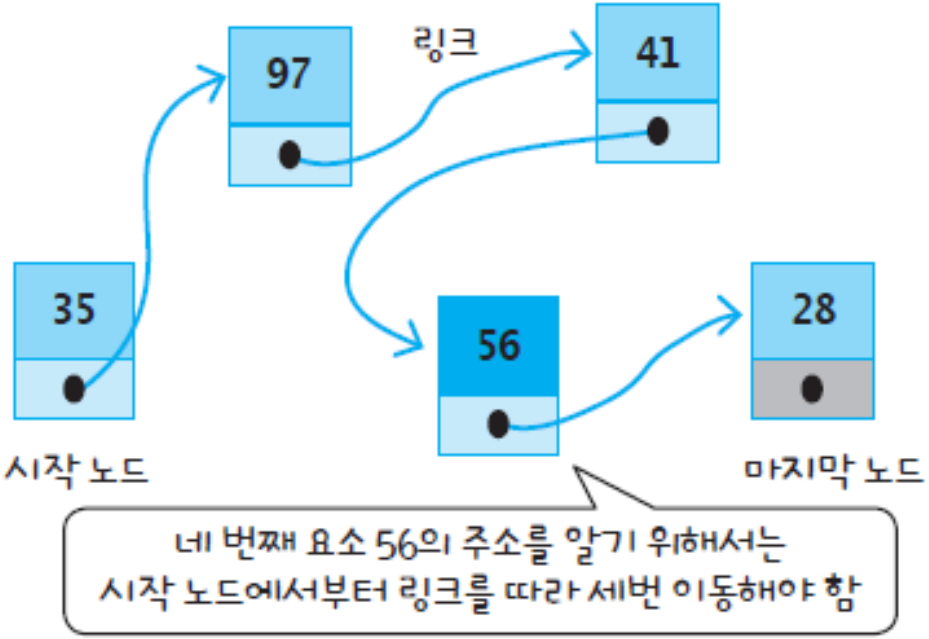
[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

배열 구조 리스트 vs 연결된 구조 리스트

- 리스트 요소들에 대한 접근



(a) 배열 구조



(b) 연결된 구조

```
myList = [10,20,30,40]
print( id(myList) )
print( id(myList[0]) )
print( id(myList[1]) )
```

```
139079692050496
10751144
10751464
```


[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

배열 구조 리스트 vs 연결된 구조 리스트

- 리스트의 용량

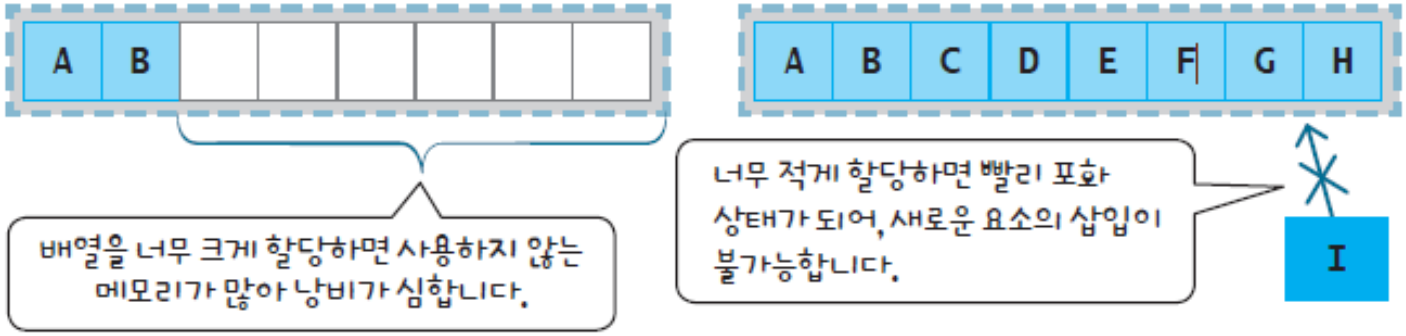


그림 3.5 | 배열은 용량이 고정됩니다.

너무 많이 할당하면 메모리 낭비
너무 적게 할당하면 포화 상태

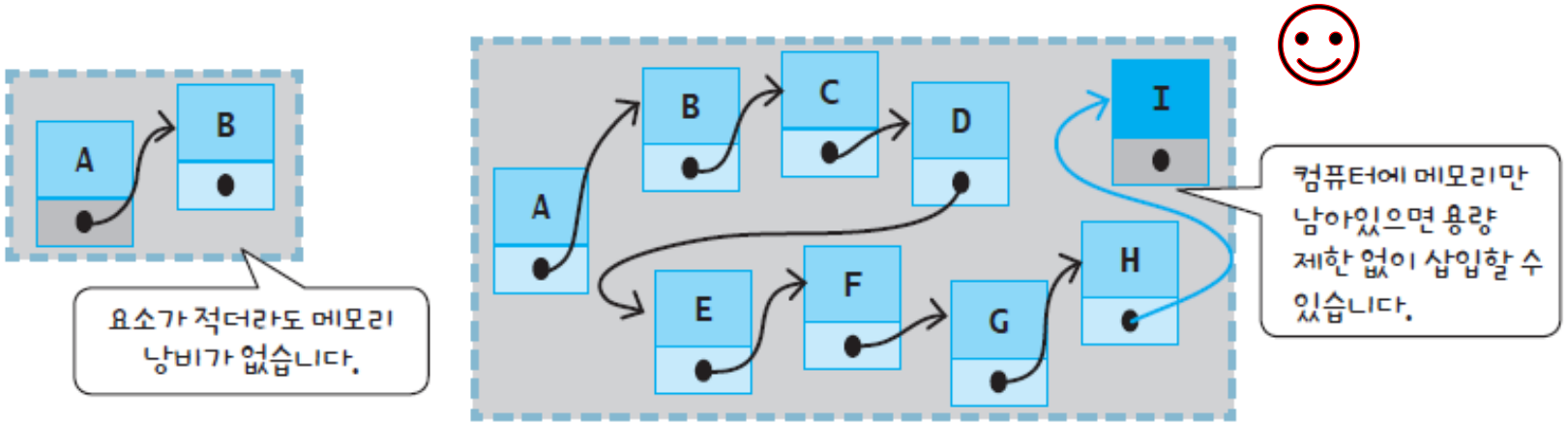


그림 3.6 | 연결된 구조는 용량이 고정되지 않습니다.

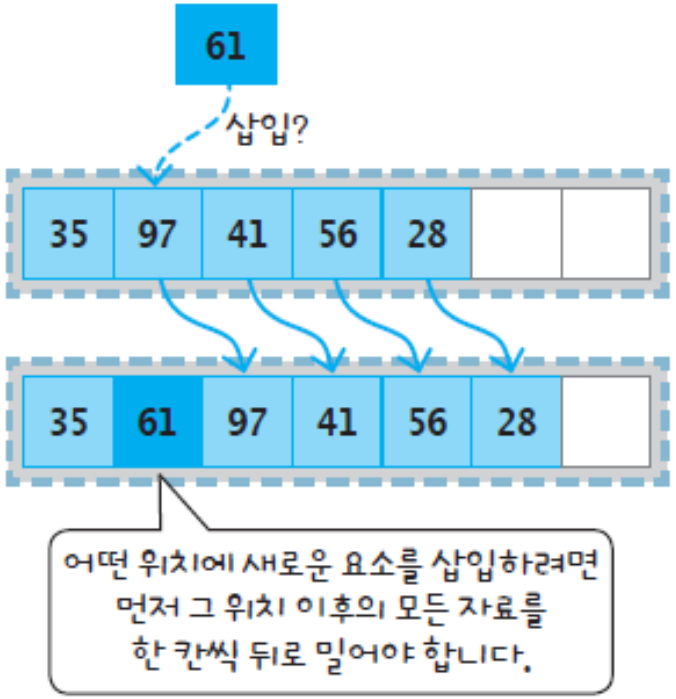
메모리만 여유 있으면 무제한

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

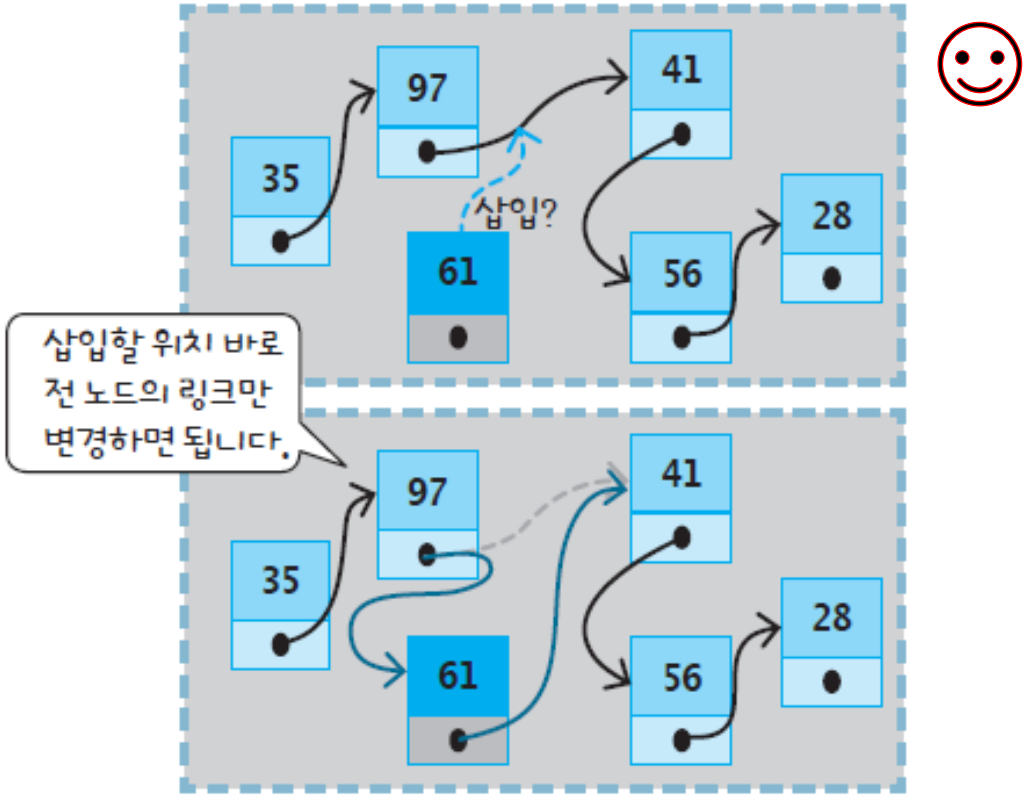
배열 구조 리스트 vs 연결된 구조 리스트

- 리스트의 삽입 연산

요소를
뒤로 이동



(a) 배열구조의 삽입 연산



(b) 연결된 구조의 삽입 연산

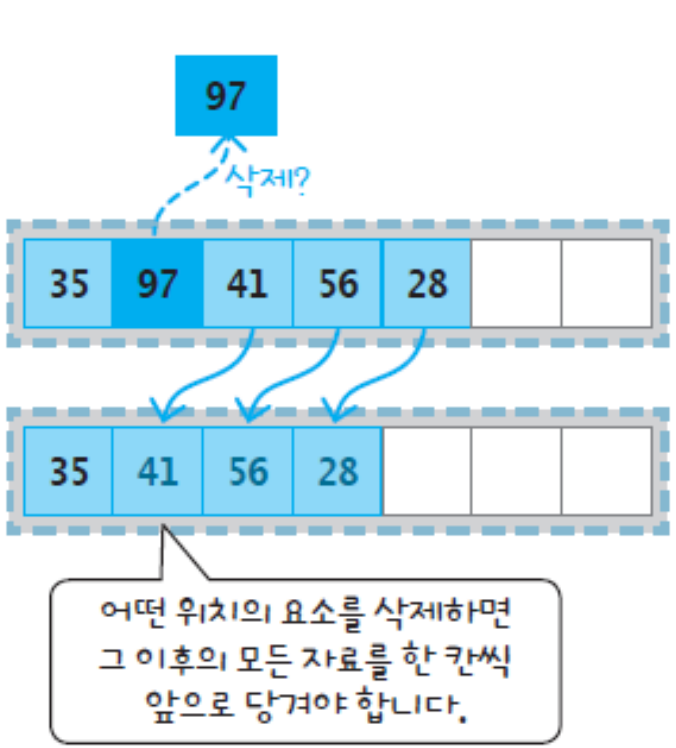
링크 수정

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

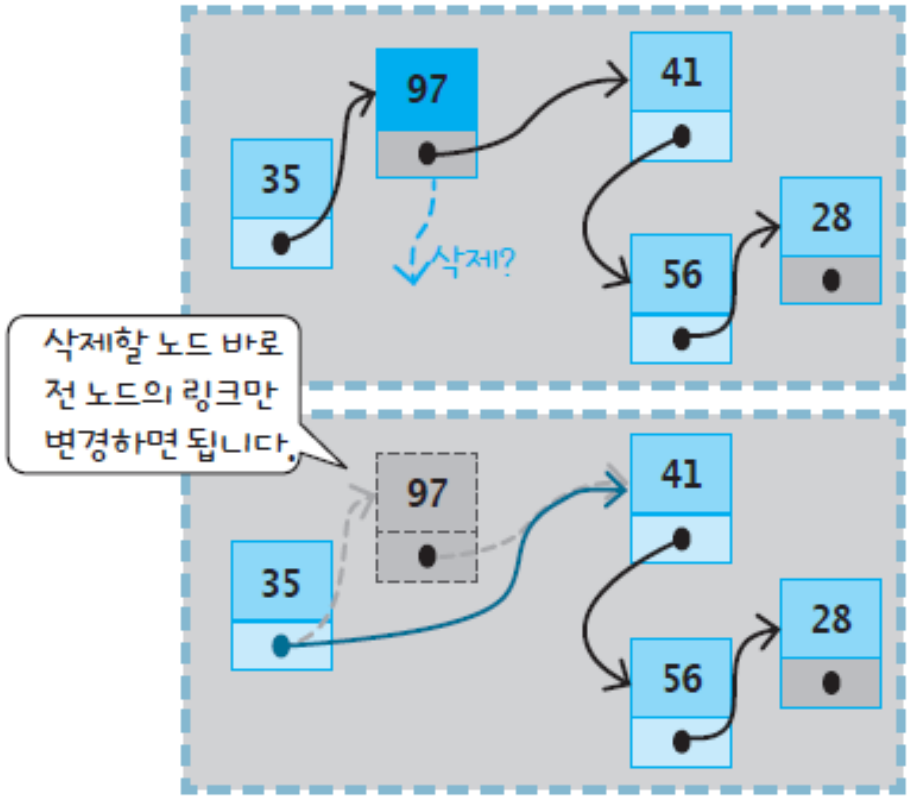
배열 구조 리스트 vs 연결된 구조 리스트

- 리스트의 삭제 연산

요소를
앞으로 이동



(a) 배열구조의 삭제 연산



(b) 연결된 구조의 삭제 연산

링크 수정

※본 강의 자료는 본인 학습용으로만 사용 가능하며 무단 복제/배포를 금지합니다.

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

파이썬 리스트는 배열 구조 리스트다

- 파이썬 리스트의 용량
  - 파이썬은 배열 구조 리스트이지만 용량이 제한되지 않도록 동적 배열로 구현됨
  - 파이썬에서 용량 확장은 내부적으로 처리되므로 사용자는 신경을 쓰지 않아도 됨
  - 파이썬 리스트의 append() 연산의 처리 시간은 항상 동일하지 않음

파이썬 리스트는 배열 구조 리스트다

- 파이썬 리스트의 다양한 연산들

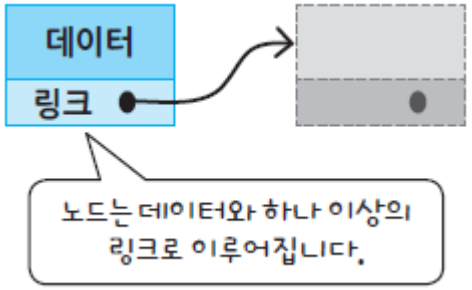
멤버함수(메서드)	설명
append(e)	새로운 요소 e를 추가합니다.
extend(lst)	리스트 lst를 리스트 s에 삽입합니다.
count(e)	리스트에서 요소 e의 개수를 세어 반환합니다.
index(e,[시작],[종료])	요소 e가 나타나는 가장 작은 위치(인덱스)를 반환합니다. 탐색을 위한 시작 위치와 종료 위치를 지정할 수도 있습니다.
insert(pos, e)	pos 위치에 새로운 요소 e를 삽입합니다.
pop(pos)	pos 위치의 요소를 꺼내고 반환합니다.
pop()	맨 뒤의 요소를 꺼내고 반환합니다.
remove(e)	요소 e를 리스트에서 제거합니다.
reverse()	리스트 요소들의 순서를 뒤집습니다.
sort([key], [reverse])	요소들을 key를 기준으로 오름차순으로 정렬합니다. reverse=True이면 내림차순으로 정렬합니다.

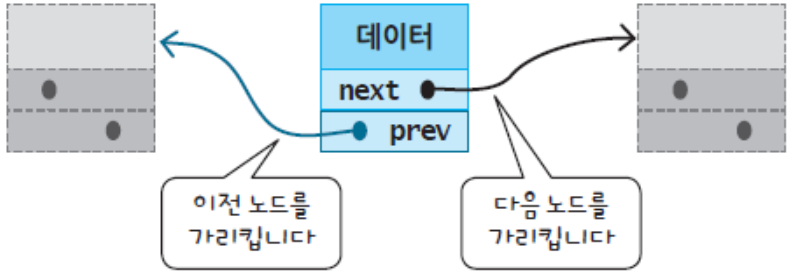
- help(list) 로 확인

연결된 구조의 리스트

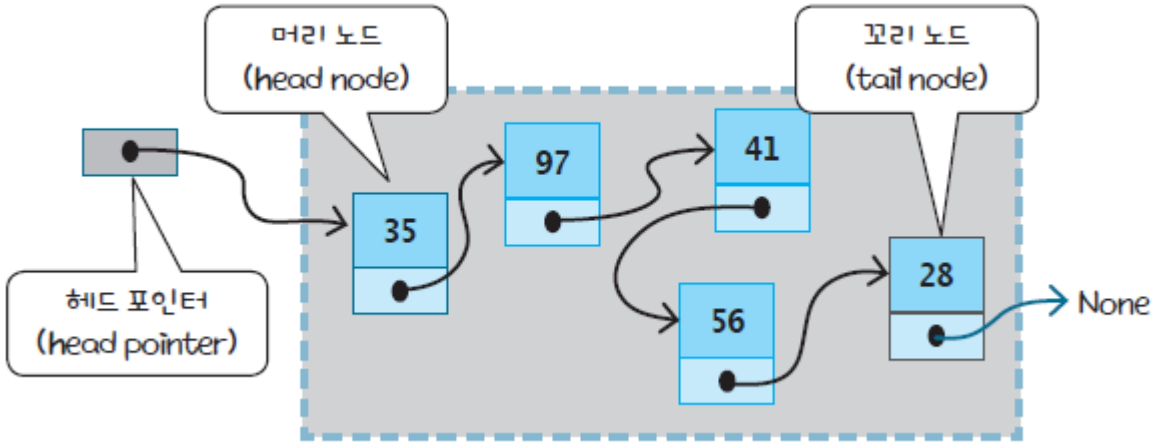
- 연결 리스트의 구조
  - 노드(node)

    : data(1개) + link(1개 이상)





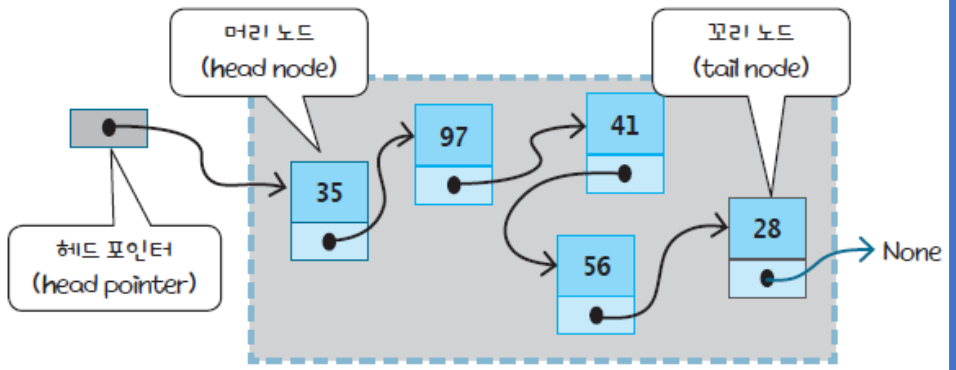
  - 헤드 포인터(head)를 잘 관리해야 함

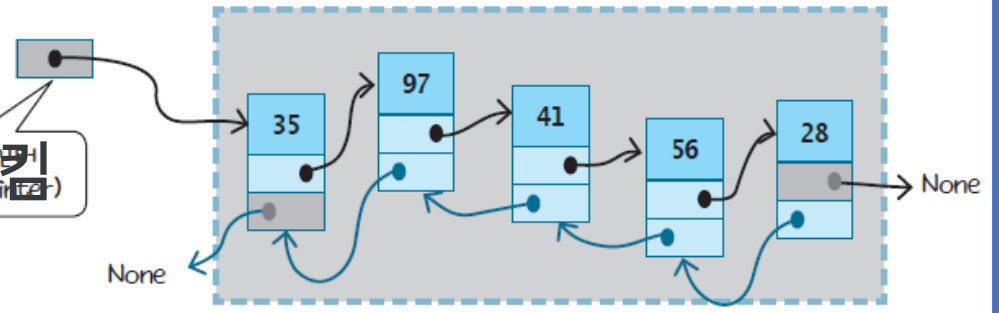


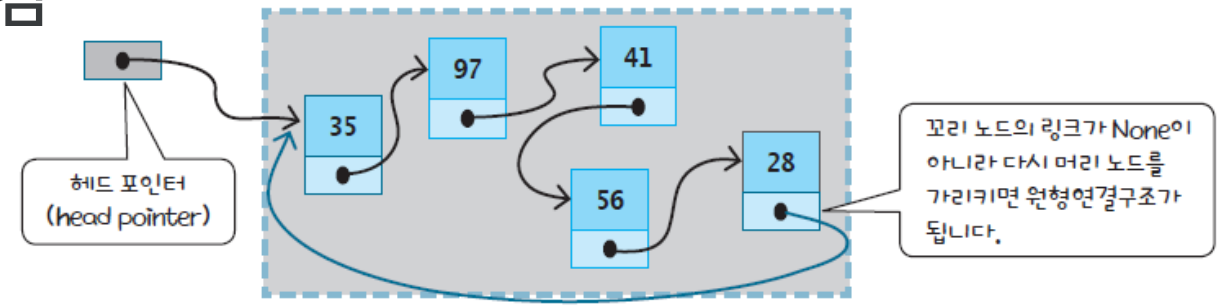
[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

연결 리스트의 종류

- 단순 연결 리스트(Singly Linked List)
  - 꼬리 노드의 링크가 None
- 더블 연결 리스트(Doubly Linked List)
  - 이전 노드(previous), 다음 노드(next)를 가리킴
- 원형 연결 리스트(Circular Linked List)
  - 꼬리 노드의 링크가 머리 노드를 가리킴





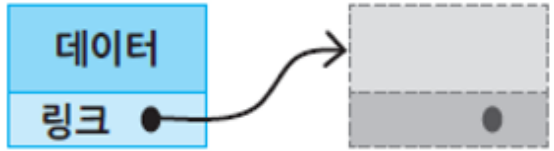


[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

단순 연결 리스트 구현

단순 연결 리스트 구현하기

- 노드 클래스 정의하기
  - 생성자 : data + link
  - append()
  - popNext()



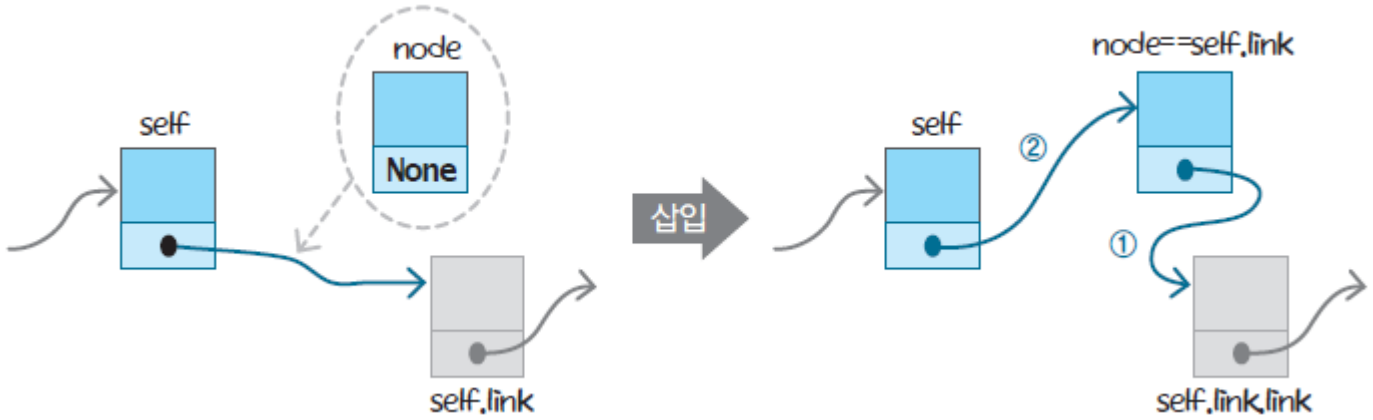
디폴트 인수

```
class Node:
    def __init__ (self, elem, link=None):
        self.data = elem  # 데이터 멤버 생성 및 초기화
        self.link = link  # 링크 생성 및 초기화
```

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

단순 연결 리스트 구현하기

- 노드 클래스 정의하기
  - 새로운 노드를 뒤에 추가 : append()



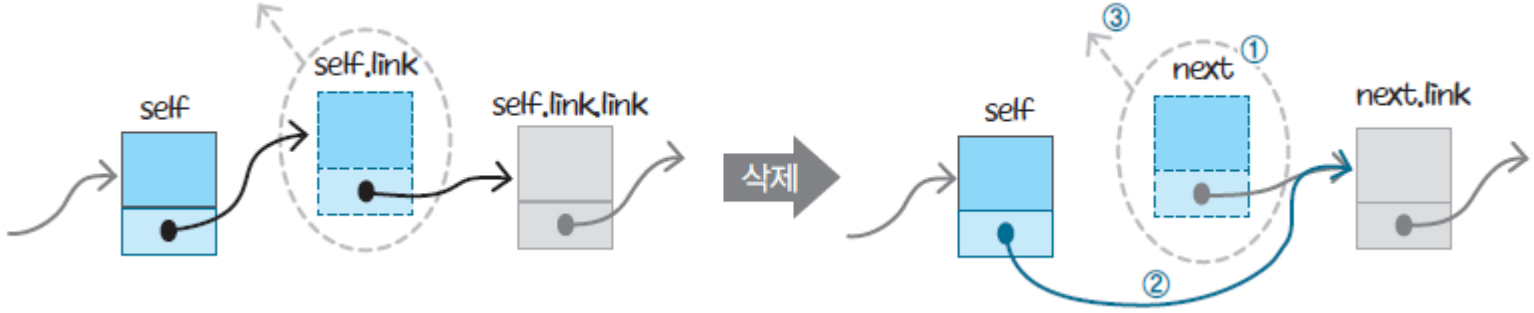

```
def append (self, node):      # self 다음에 node를 넣는 연산
    if node is not None :
        node.link = self.link
        self.link = node
```

← 삽입할 노드가 None이 아니면 ①과 ②단계를 통해 node를 다음 노드로 연결

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

단순 연결 리스트 구현하기

- 노드 클래스 정의하기
  - 다음 노드를 연결 구조에서 꺼내기 : popNext()




```
def popNext (self):                    # self의 다음 노드를 삭제하는 연산
    next = self.link                   # 현재 노드(self)의 다음 노드
    if next is not None :
        self.link = next.link
    return next                        # 다음 노드를 반환
```

① 단계

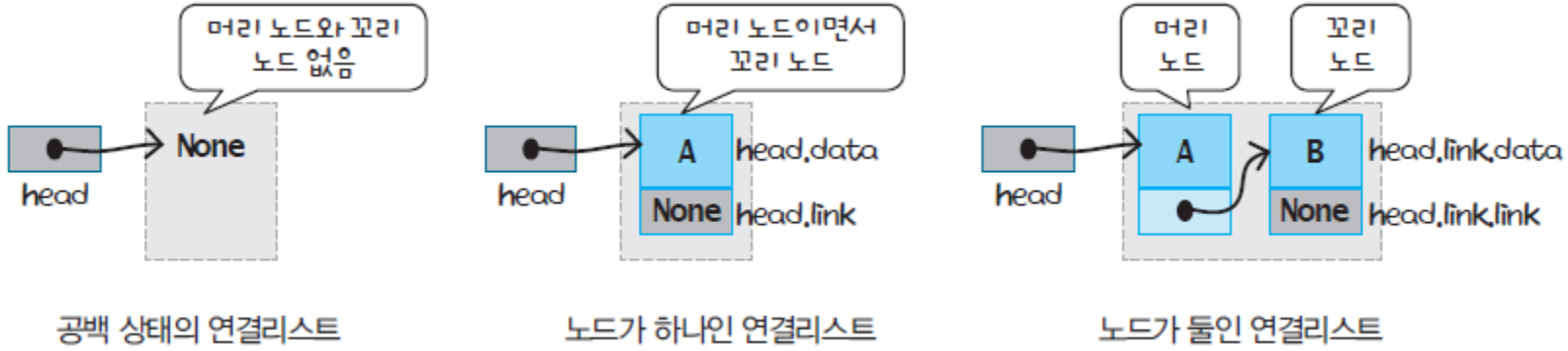
next가 None이 아니면 ②단계 처리

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

단순 연결 리스트 구현하기

- 단순 연결 리스트 클래스 정의하기
  - 생성자 : 헤더 포인터만 관리
  - 연산
    - isEmpty()
    - *isFull()*
    - getNode(pos)
    - insert(pos, e)
    - delete(pos)
    - size()

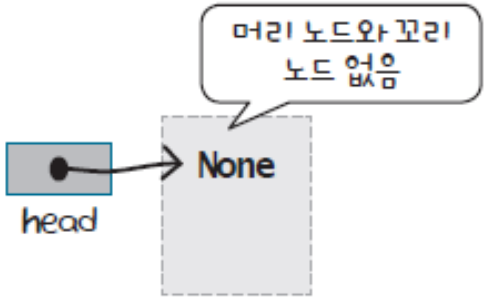
```
class LinkedList:
    def __init__( self ):
        self.head = None
```



[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

단순 연결 리스트 구현하기

- 단순 연결 리스트 클래스 정의하기
  - 공백 상태 검사 : isEmpty()
  - 포화 상태 검사 : *isFull()*



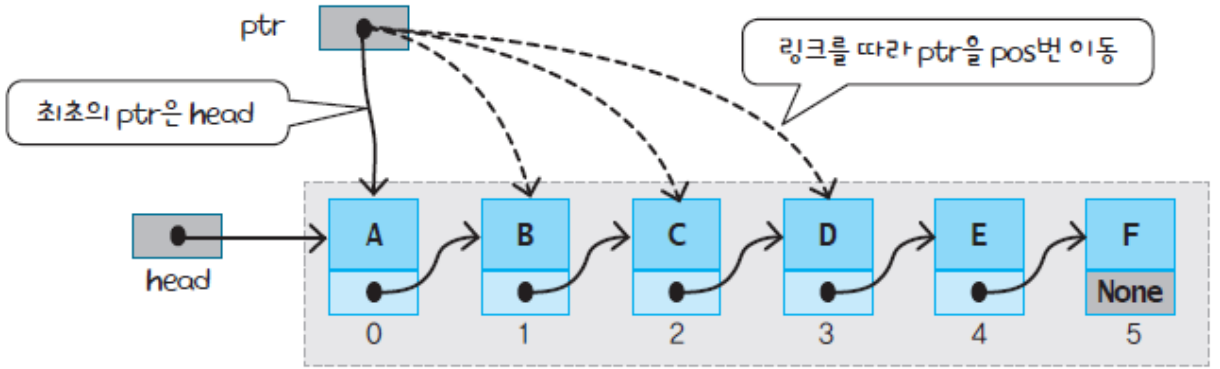
공백 상태의 연결리스트

```
def isEmpty( self ):            # 공백 상태 검사
    return self.head == None    # head가 None이면 공백

def isFull( self ):             # 포화 상태 검사
    return False                # 연결된 구조에서는 포화 상태 없음
```


[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

단순 연결 리스트 구현하기

- 단순 연결 리스트 클래스 정의하기
  - pos번째 노드 반환 : getNode(pos)




```
def getNode(self, pos) :
    if pos < 0 : return None          # 잘못된 위치 -> None 반환
    ptr = self.head                   # 시작 위치 -> head
    for i in range(pos):
        if ptr == None :
            return None
        ptr = ptr.link
    return ptr                        # 최종 노드를 반환
```

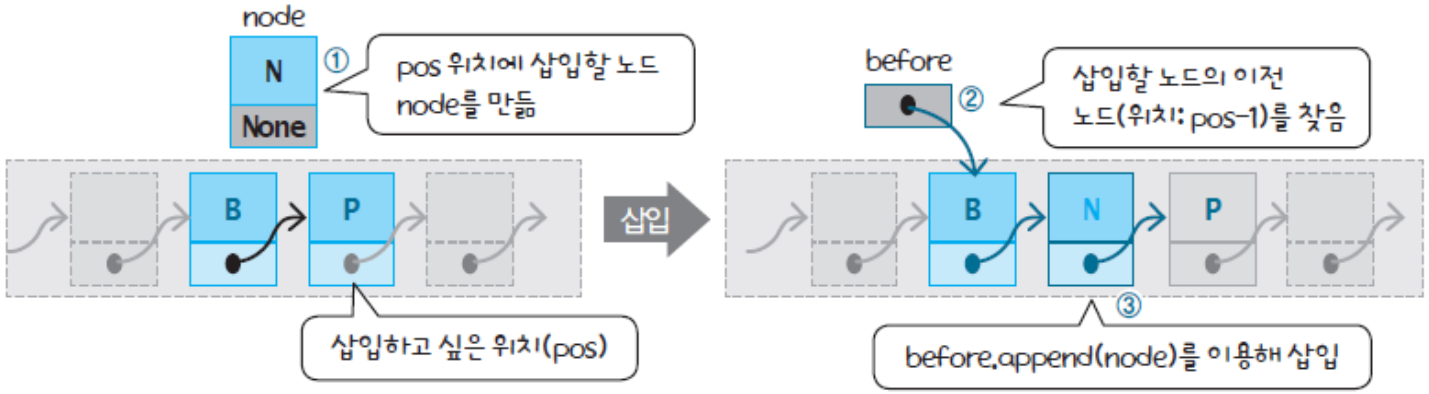
← 머리 노드에서부터 링크를 따라 pos번 이동하면 pos 위치의 노드에 도착.
위치는 0부터 시작한다고 가정함.

※본 강의 자료는 본인 학습용으로만 사용 가능하며 무단 복제/배포를 금지합니다.

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

단순 연결 리스트 구현하기

- 단순 연결 리스트 클래스 정의하기
  - pos위치에 새 요소 삽입 : insert(pos, e)



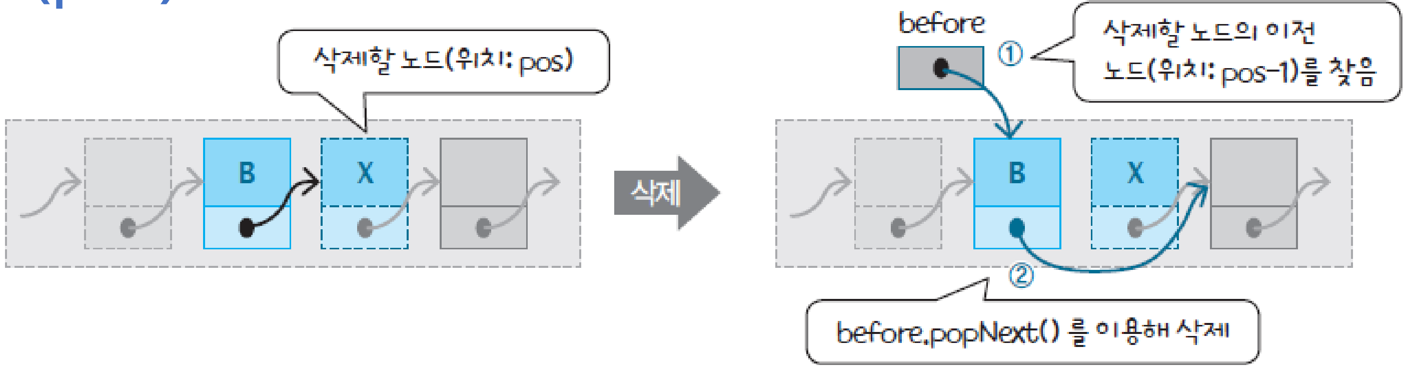

```
def insert(self, pos, e) :
    node = Node(e, None)                # 삽입할 새로운 노드를 만듦
    before = self.getNode(pos-1)        # 삽입할 위치 이전 노드 탐색
    if before == None :
        node.link = self.head
        self.head = node
    else : before.append(node)          # 아닌 경우: before뒤에 추가
```

← before가 None이면 맨 앞에 추가, 리스트의 머리노드(head)가 변경됨.

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

단순 연결 리스트 구현하기

- 단순 연결 리스트 클래스 정의하기
  - pos위치의 요소 삭제 : delete(pos)



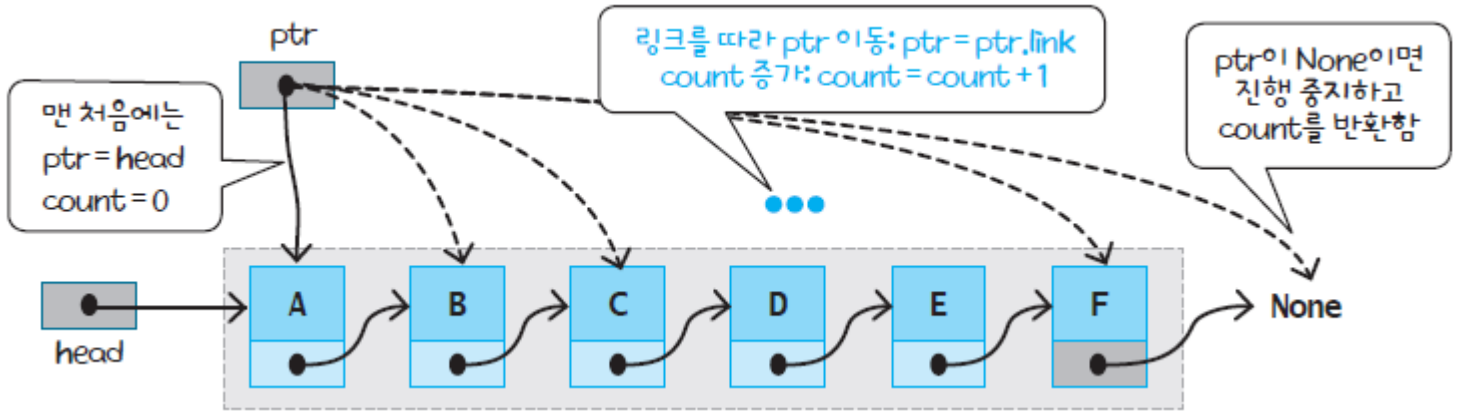

```
def delete(self, pos) :
    before = self.getNode(pos-1)          # 삭제할 위치 이전 노드 탐색
    if before == None :
        before = self.head
        if self.head is not None :
            self.head = self.head.link
        return before
    else: return before.popNext()          # before의 다음 노드 삭제
```

← 머리노드를 삭제하면 head가 다음 노드로 변경됨.

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

단순 연결 리스트 구현하기

- 단순 연결 리스트 클래스 정의하기
  - 전체 요소의 수 : size()




```
def size( self ) :
    ptr = self.head
    count = 0;
    while ptr is not None :
        ptr = ptr.link
        count += 1
    return count
```

머리노드부터 링크를 따라
None이 될 때까지 이동하면서
이동 횟수를 기록함.

```
    # ptr이 None이 아닌 동안
    # 링크를 따라 ptr 이동
    # 이동할 때마다 count 증가
    # count 반환
```

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

실습 : 단순 연결 리스트 vs 파이썬 리스트 비교

- 앞에서 작성한 단순 연결 리스트와 파이썬의 리스트 연산(함수)을 각각 사용하여 아래와 같이 결과를 출력 하시오.

```
# 단순연결리스트(SinglyLinkedList) 사용
s = SinglyLinkedList()
s.display('연결리스트( 초기 ): ')
s.insert(0, 10)
s.insert(0, 20)
s.insert(1, 30)
s.insert(s.size(), 40)
s.insert(2, 50)
s.display("연결리스트(삽입x5): ")
s.replace(2, 90)
s.display("연결리스트(교체x1): ")
s.delete(2)
s.delete(3)
s.delete(0)
s.display("연결리스트(삭제x3): ")
```

```
연결리스트( 초기 ): None
연결리스트(삽입x5): 20->30->50->10->40->None
연결리스트(교체x1): 20->30->90->10->40->None
연결리스트(삭제x3): 30->10->None
```

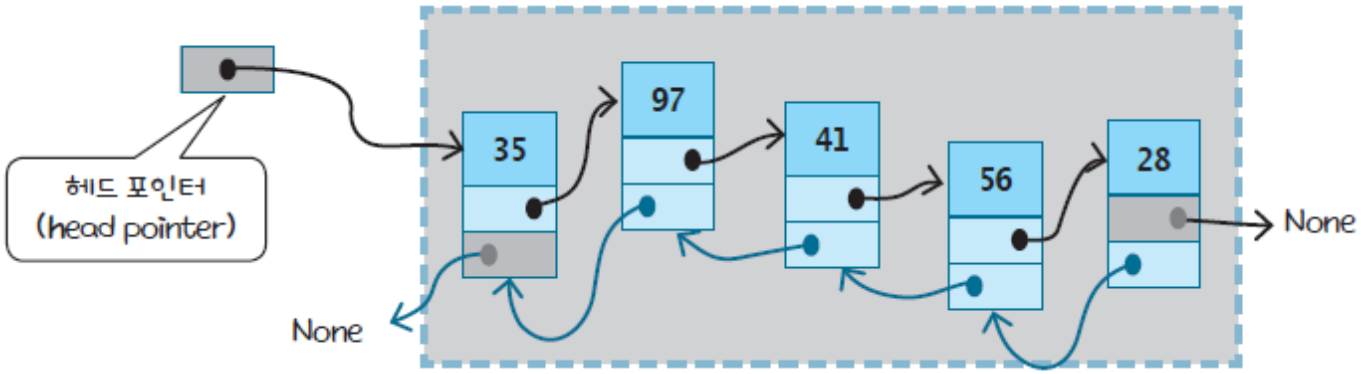
파이썬 연산 동작 결과

```
파이썬list( 초기 ): []
파이썬list(삽입x5): [20, 30, 50, 10, 40]
파이썬list(교체x1): [20, 30, 90, 10, 40]
파이썬list(삭제x3): [30, 10]
```

이중 연결 리스트 구현

이중 연결 리스트

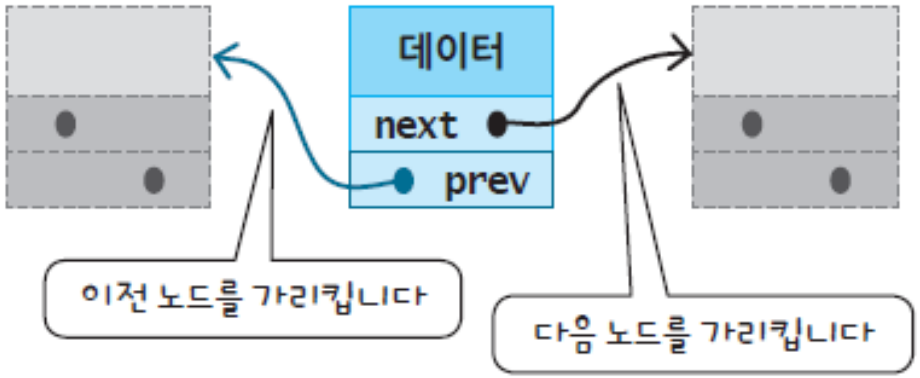
- 단순 연결 리스트 → 이중 연결 리스트가 더 좋은 이유
  - 양방향 탐색 가능
  - 노드 삭제 및 삽입이 용이
  - 원형 리스트 구현 용이
  - 역순 순회 가능



[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

이중 연결 리스트 구현하기

- 노드 클래스 정의하기
  - 생성자 : data + 이전링크 +다음링크
  - append()
  - popNext()



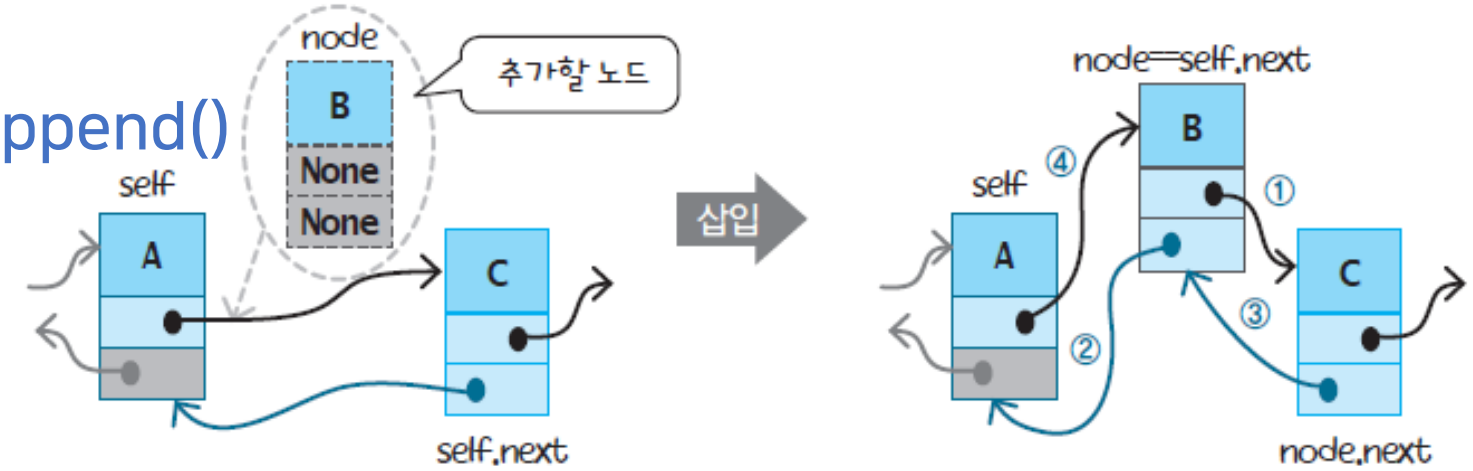

```
class DNode:
    def __init__ (self, elem, prev=None, next=None):
        self.data = elem  # 노드의 데이터 필드(요소)
        self.next = next  # 다음 노드를 위한 링크
        self.prev = prev  # 이전 노드를 위한 링크(추가됨)
```

← 이중 연결 노드의 생성자

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

이중 연결 리스트 구현하기

- 노드 클래스 정의하기
  - 새로운 노드를 뒤에 추가 : append()

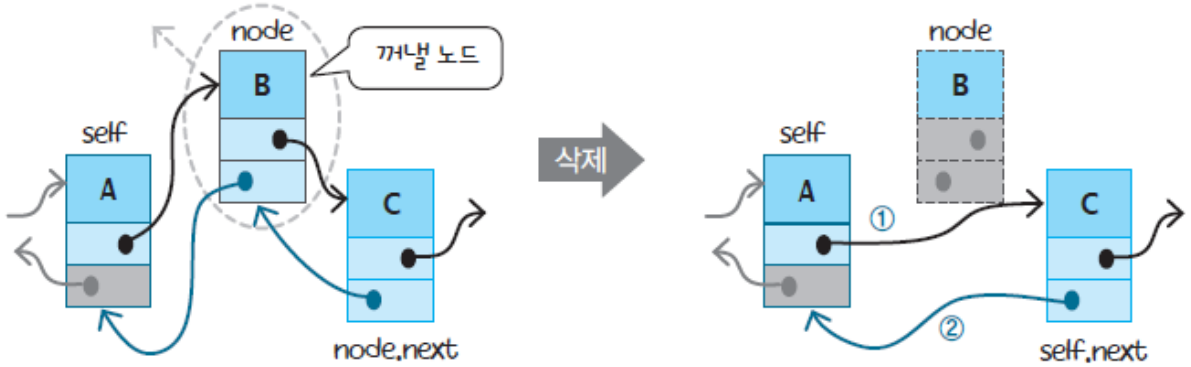



```
def append (self, node):              # self 다음에 node를 넣는 연산
    if node is not None :             # node가 None이 아니면
        node.next = self.next         # ①
        node.prev = self              # ②
        if node.next is not None:     # ③ self의 다음 노드가 있으면
            node.next.prev = node     # 그 노드의 이전 노드는 node
        self.next = node              # ④
```

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

이중 연결 리스트 구현하기

- 노드 클래스 정의하기
  - 다음 노드를 연결 구조에서 꺼내기 : popNext()




```
def popNext (self):                 # self 다음 노드 삭제 연산
    node = self.next                # 삭제할 노드
    if node is not None :           # next가 None이 아니면
        self.next = node.next       # ①
        if self.next is not None:   # ② 다음 노드가 있으면
            self.next.prev = self   #   그 노드의 이전 노드는 self
    return node                     # 다음 노드를 반환
```

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

이중 연결 리스트 구현하기

- 이중 연결 리스트 클래스 정의하기
  - 생성자 : 헤더 포인터만 관리
  - 연산
    - isEmpty()
    - *isFull()*
    - getNode(pos)
    - insert(pos, e)
    - delete(pos)
    - size()

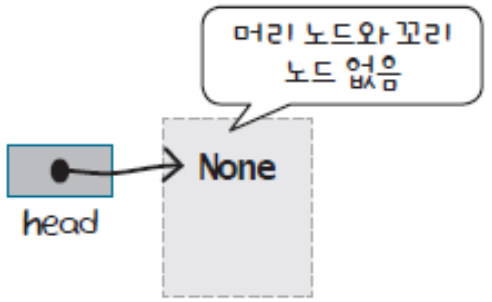
```
class DblLinkedList:
    def __init__( self ):
        self.head = None
```

대부분의 연산들은 단순 연결 리스트 클래스와 거의 유사
- Node를 DNode로 수정
- .link를 .next로 수정

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

이중 연결 리스트 구현하기

- 단순 연결 리스트 클래스 정의하기
  - 공백 상태 검사 : isEmpty()
  - 포화 상태 검사 : *isFull()*



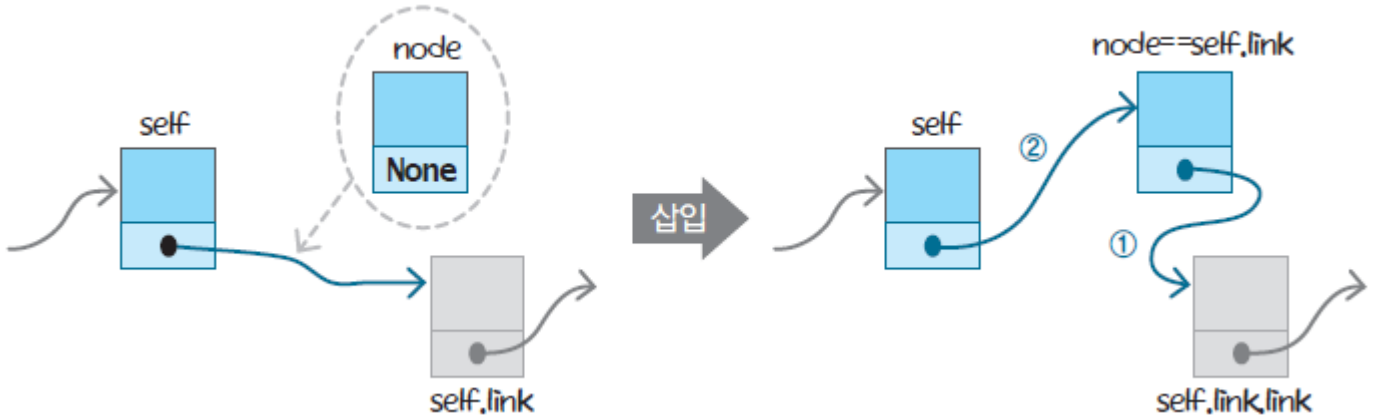
공백 상태의 연결리스트

```
def isEmpty( self ):            # 공백 상태 검사
    return self.head == None    # head가 None이면 공백

def isFull( self ):             # 포화 상태 검사
    return False                # 연결된 구조에서는 포화 상태 없음
```


[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

단순 연결 리스트 구현하기

- 노드 클래스 정의하기
  - 새로운 노드를 뒤에 추가 : append()



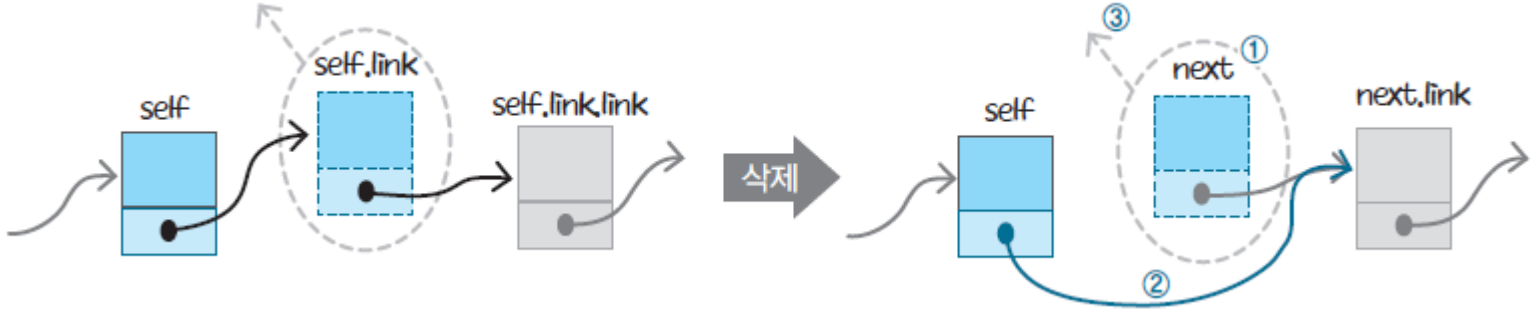

```
def append (self, node):      # self 다음에 node를 넣는 연산
    if node is not None :
        node.link = self.link
        self.link = node
```

← 삽입할 노드가 None이 아니면 ①과 ②단계를 통해 node를 다음 노드로 연결

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

단순 연결 리스트 구현하기

- 노드 클래스 정의하기
  - 다음 노드를 연결 구조에서 꺼내기 : popNext()




```
def popNext (self):                    # self의 다음 노드를 삭제하는 연산
    next = self.link                   # 현재 노드(self)의 다음 노드
    if next is not None :
        self.link = next.link
    return next                        # 다음 노드를 반환
```

① 단계

next가 None이 아니면 ②단계 처리

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

실습 : 이중 연결 리스트 vs 파이썬 리스트 비교

- 앞에서 작성한 이중 연결 리스트와 파이썬의 리스트 연산(함수)을 각각 사용하여 아래와 같이 결과를 출력 하시오.

```
# 단순연결리스트(SinglyLinkedList) 사용
s = SinglyLinkedList()
s.display('연결리스트( 초기 ): ')
s.insert(0, 10)
s.insert(0, 20)
s.insert(1, 30)
s.insert(s.size(), 40)
s.insert(2, 50)
s.display("연결리스트(삽입x5): ")
s.replace(2, 90)
s.display("연결리스트(교체x1): ")
s.delete(2)
s.delete(3)
s.delete(0)
s.display("연결리스트(삭제x3): ")
```

```
연결리스트( 초기 ): None
연결리스트(삽입x5): 20->30->50->10->40->None
연결리스트(교체x1): 20->30->90->10->40->None
연결리스트(삭제x3): 30->10->None
```

파이썬 연산 동작 결과

```
파이썬list( 초기 ): []
파이썬list(삽입x5): [20, 30, 50, 10, 40]
파이썬list(교체x1): [20, 30, 90, 10, 40]
파이썬list(삭제x3): [30, 10]
```

실습 : 음악 재생 목록 출력하기

- 앞에서 기술한 이중 연결 구조 리스트를 참고하여 음악 재생 목록 클래스를 만들고 출력하기 (단, 출력 결과가 완전 동일하지 않아도 됨)
  - 클래스명 : MusicPlaylist
  - 필요한 연산
    - 곡 추가 : add_song(song)
    - 곡 삭제 : remove_song(song)
    - 곡 목록 출력 : show_playlist()

실습 : 음악 목록 관리 프로그램

```
# 사용 예시
playlist = MusicPlaylist()

playlist.add_song("Butter")
playlist.add_song("Permission to Dance")
playlist.add_song("Life Goes On")
playlist.show_playlist()
playlist.remove_song("Permission to Dance")
playlist.show_playlist()
playlist.remove_song("Dynamite")
```

```
곡 'Butter'이(가) 재생 목록에 추가되었습니다.
곡 'Permission to Dance'이(가) 재생 목록에 추가되었습니다.
곡 'Life Goes On'이(가) 재생 목록에 추가되었습니다.

--재생 목록--
Butter
Permission to Dance
Life Goes On

곡 'Permission to Dance'이(가) 재생 목록에서 제거되었습니다.

--재생 목록--
Butter
Life Goes On

곡 'Dynamite'이(가) 재생 목록에 없습니다.
```

Q & A

Next Topic

- 점화식과 재귀 알고리즘(Recursive Algorithm)

Keep learning, see you soon!