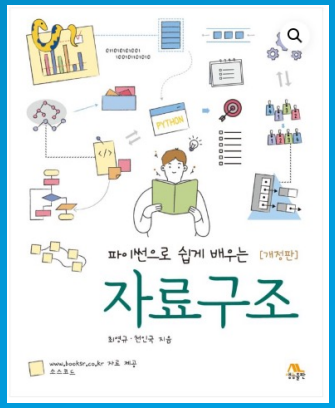


CHAPTER 03

리스트와 집합

3장. 리스트와 집합

3.1 리스트란?

3.2 파이썬의 리스트

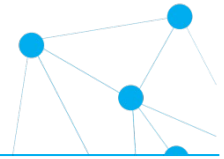
3.3 배열로 구현한 리스트

3.4 리스트의 응용: 라인 편집기

3.5 집합이란?

3.6 집합의 구현

3.1 리스트란?



- 가장 자유로운 선형 자료구조

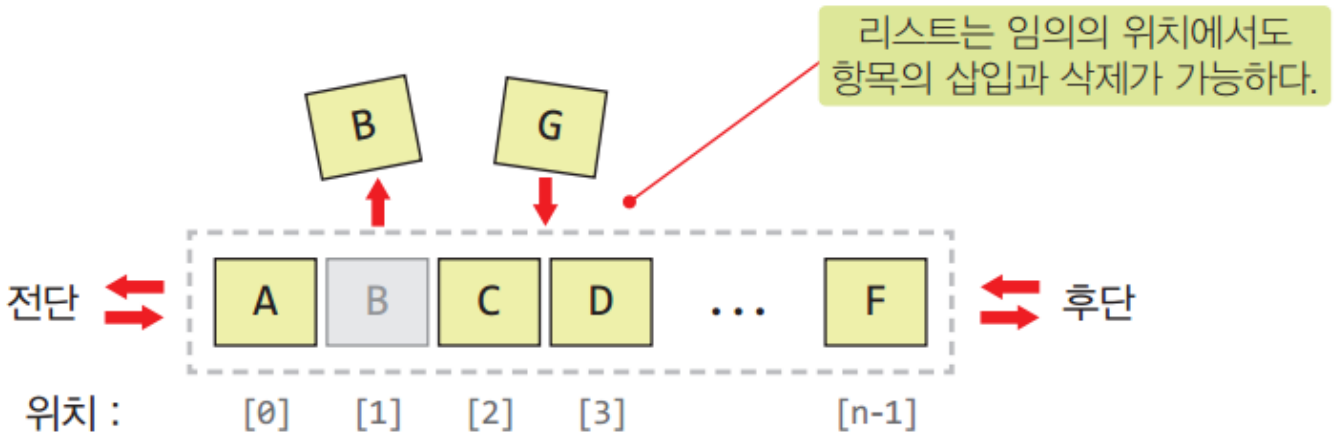
$$L = [item_0, item_1, item_2, \ldots, item_{n-1}]$$

- 리스트(list), 선형 리스트(linear list)
  - 순서를 가진 항목들의 모임
  - 항목들은 "위치"를 가짐
  - $L = [item_0, item_1, item_2, \ldots, item_{n-1}]$

- 집합은?
  - 항목간의 순서의 개념이 없음
  - 항목의 중복을 허용하지 않음
  - ➔ 선형 자료구조가 아님

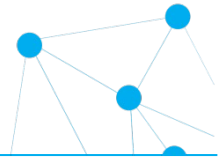
리스트의 구조

- 리스트
  - 항목들이 순서대로 나열되어 있고, 각 항목들 은 위치를 갖는다.



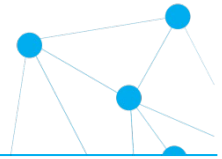
- Stack, Queue, Deque과의 차이점
  - 자료의 접근 위치

리스트의 추상 자료형

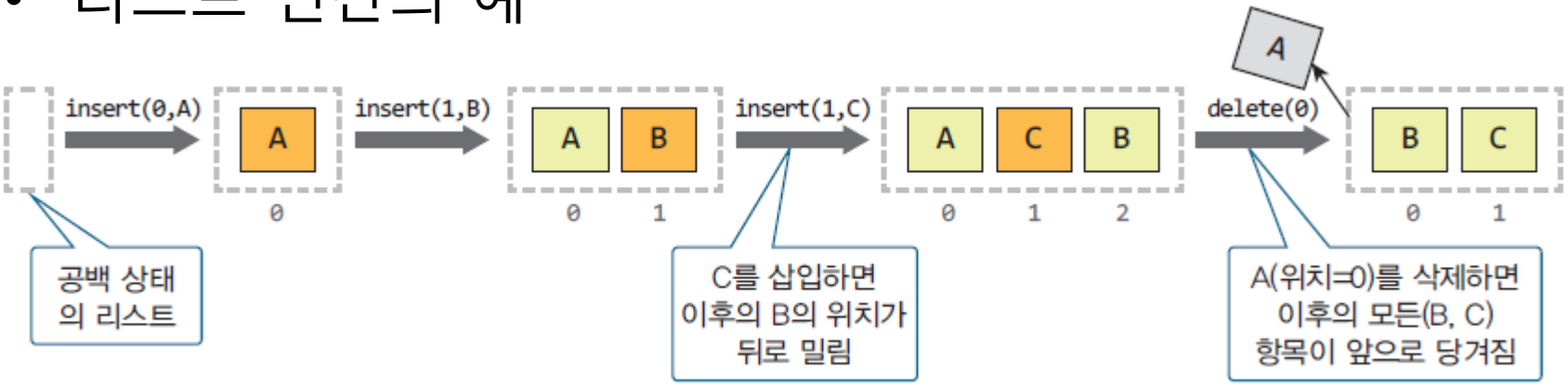


- 데이터
  - 같은 유형의 요소들의 순서 있는 모임
- 연산
  - insert(pos, e): pos 위치에 새로운 요소 e를 삽입한다.
  - delete(pos): pos 위치에 있는 요소를 꺼내고(삭제) 반환한다.
  - isEmpty(): 리스트가 비어 있는지를 검사한다.
  - isFull(): 리스트가 가득 차 있는지를 검사한다.
  - getEntry(pos): pos 위치에 있는 요소를 반환한다.

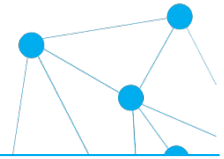
리스트의 연산



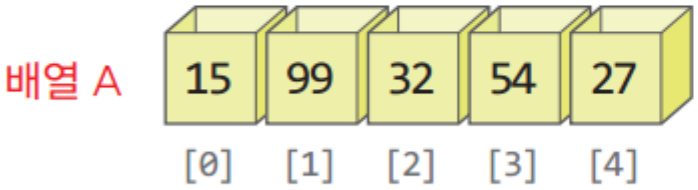
- 리스트의 연산
  - 삽입과 삭제: 리스트의 상태를 변경시킴
  - 연산의 위치를 지정해야 함
  - 위치(인덱스)는 보통 0부터 시작
  - 어떤 위치에 항목을 삽입/삭제하면 이후의 모든 자료들의 위치가 한 칸씩 밀리거나 당겨짐
- 리스트 연산의 예



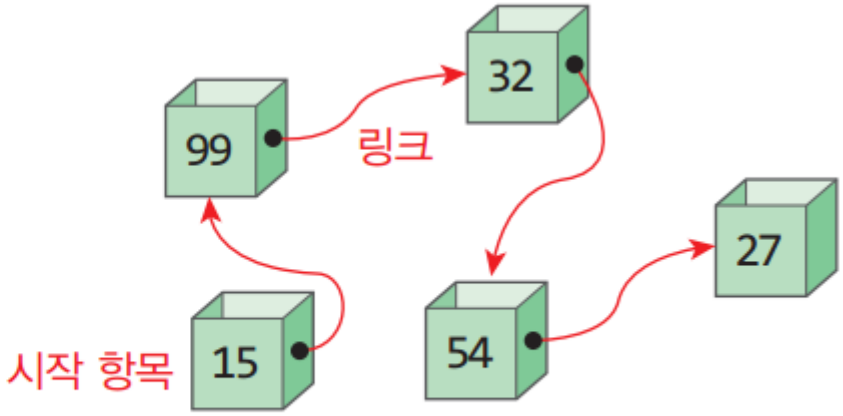
리스트 구현 방법



- 배열 구조
  - 구현이 간단
  - 항목 접근이 $O(1)$
  - 삽입, 삭제시 오버헤드
  - 항목의 개수 제한
- 연결된 구조
  - 구현이 복잡
  - 항목 접근이 $O(n)$
  - 삽입, 삭제가 효율적
  - 크기가 제한되지 않음

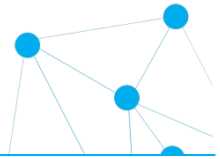


배열 구조의 리스트

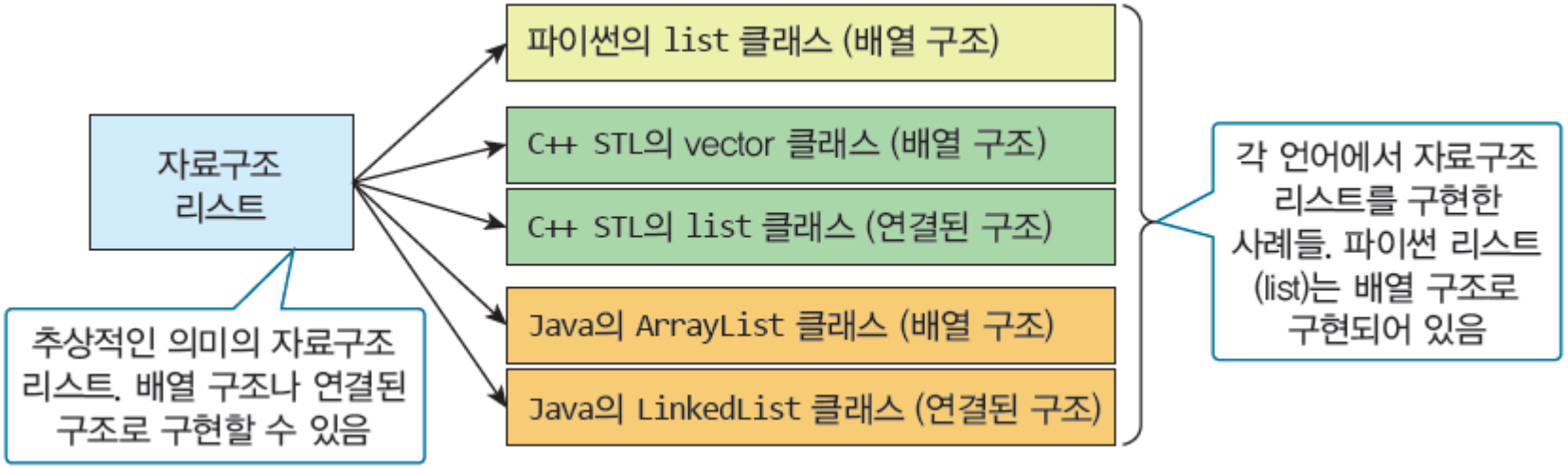


연결된 구조의 리스트

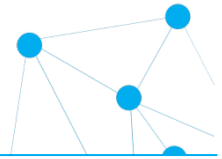
배열과 파이썬 리스트



- 파이썬의 리스트는 자료구조 리스트를 배열구조로 구현한 하나의 사례임.
- 자료구조 리스트를 구현한 다양한 사례들



3.2 파이썬 리스트



- 파이썬의 리스트는 스마트한 배열
- 파이썬 리스트 선언

```
A = [ 1, 2, 3, 4, 5 ]
B = [ 0 ] * 5
```

C 언어의 배열 선언

```
int A[5] = { 1, 2, 3, 4, 5 };
int B[5] = { 0, 0, 0, 0, 0 };
```

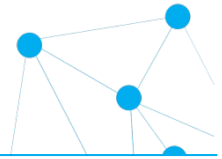
- 리스트의 크기 구하기
  - 내장 함수 len() 사용

```
print('파이썬 리스트 A의 크기는 ', len(A))
```

- 크기를 늘릴 수 있다!

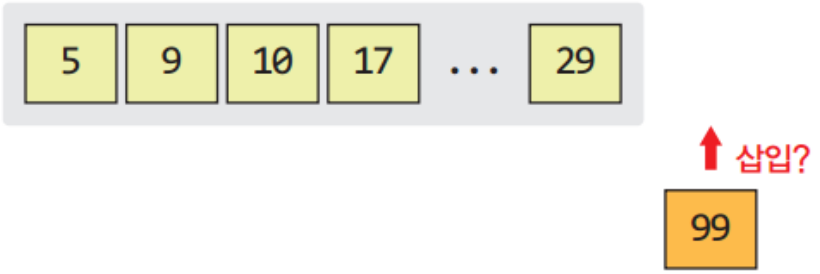
```
A.append(6)          # A = [1, 2, 3, 4, 5, 6]
A.append(7)          # A = [1, 2, 3, 4, 5, 6, 7]
A.insert(0, 0)       # A = [0, 1, 2, 3, 4, 5, 6, 7]
B.insert(9)          # B = [0, 0, 0, 0, 0, 9]
```

파이썬 리스트는 동적 배열



- 필요한 양보다 넉넉한 크기의 메모리를 사용!

(크기 〈 용량) 인 상황에서의 항목 삽입

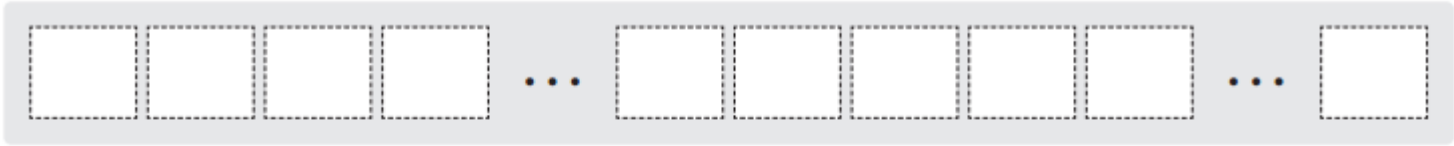


(크기 == 용량) 인 상황에서의 항목 삽입

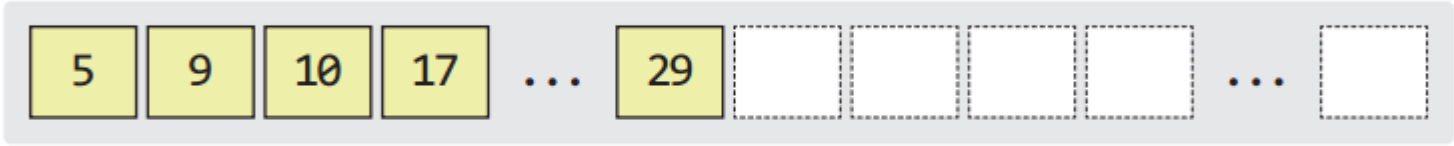
- 남은 공간이 없으면 어떻게 삽입할까?

동적 배열 구조에서의 용량 증가 과정

Step1: 용량을 확장한 새로운 배열 할당. (예: 기존 배열 용량의 2배)



Step2: 기존의 배열을 새로운 배열에 복사

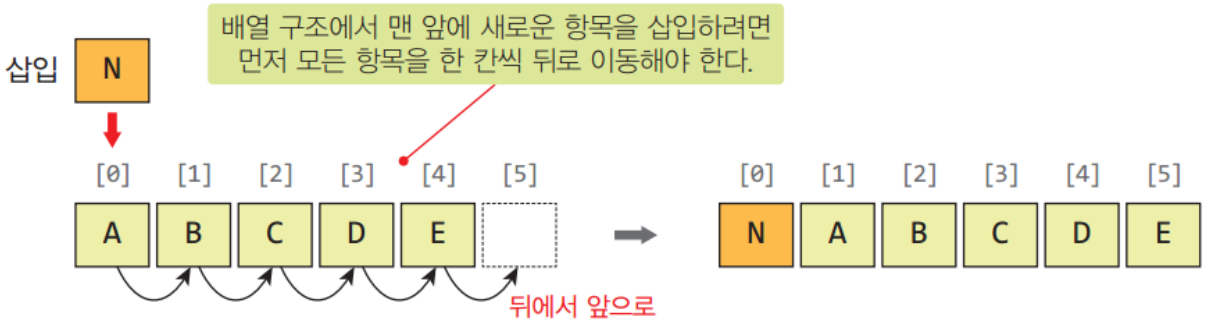


Step3: 항목을 삽입

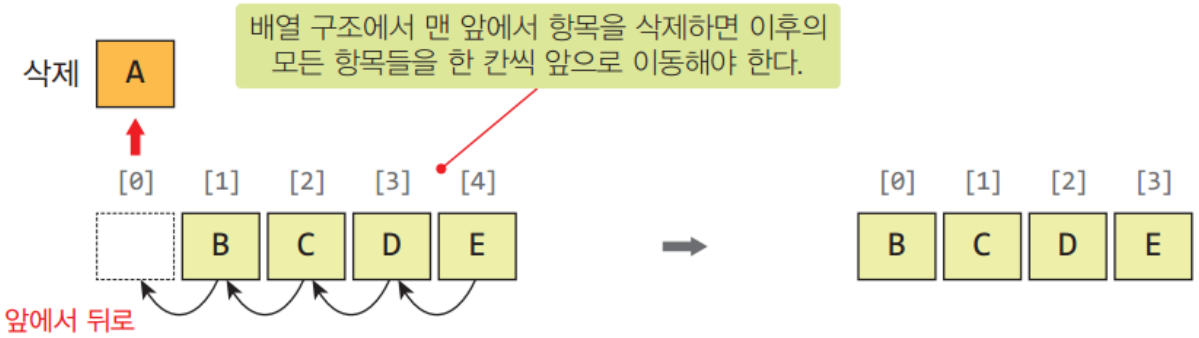
Step4: 기존 배열 해제, 리스트로 새 배열 사용

연산들의 시간 복잡도

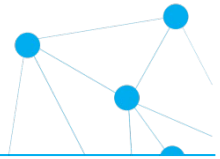
- append(e) : 대부분의 경우 $O(1)$
- insert(pos, e): $O(n)$



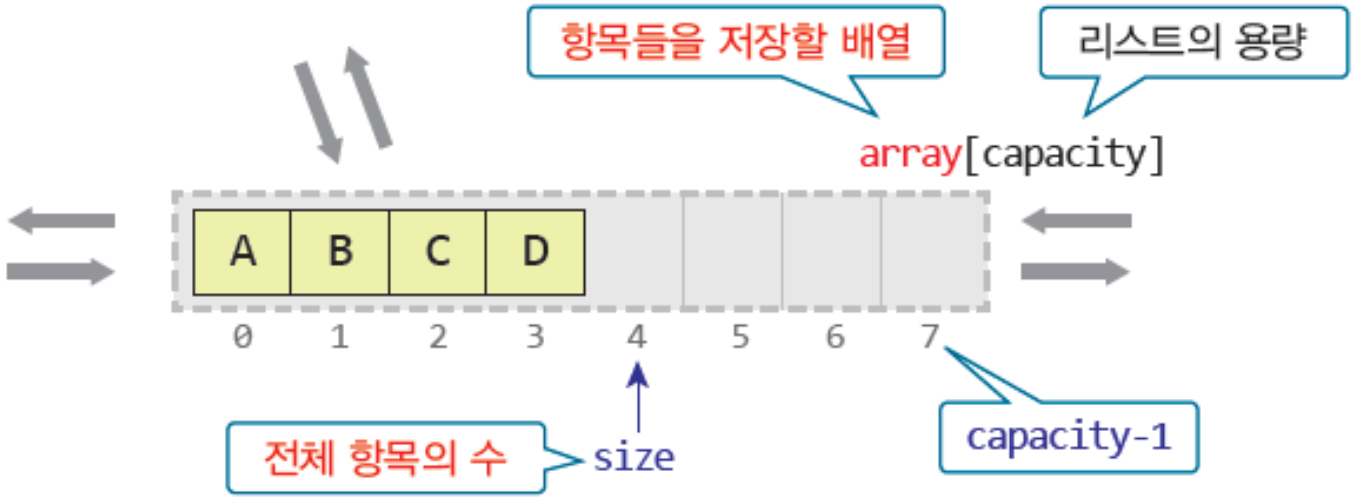
- pop(pos): $O(n)$



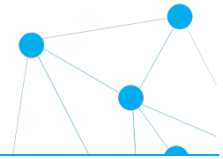
3.3 배열로 구현한 리스트



- 배열을 이용한 리스트의 구조
  - 용량이 고정된 리스트의 구조



배열로 구현한 리스트(함수 버전)

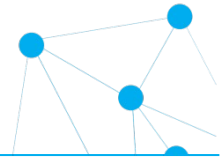


- 리스트 ADT를 배열 구조(파이썬 리스트 이용)로 구현

- 구현 방법: 전역 변수와 함수 이용 / 클래스 이용
- 함수 버전
  - 리스트의 데이터를 전역 변수로 선언
  - 리스트의 연산은 일반 함수로 구현

- 클래스 버전
  - 리스트의 데이터를 클래스의 멤버 변수
  - 리스트의 연산은 클래스의 메소드(멤버 함수)

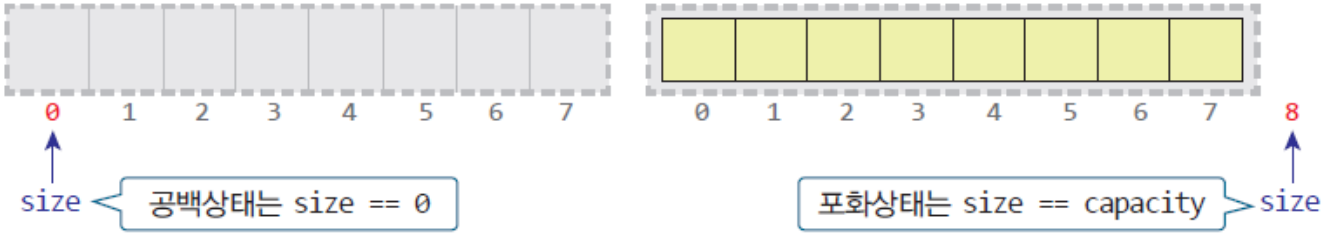
방법 1: 함수로 구현



- 리스트의 데이터: 전역 변수

```
capacity = 100
array = [None]*capacity
size = 0
```

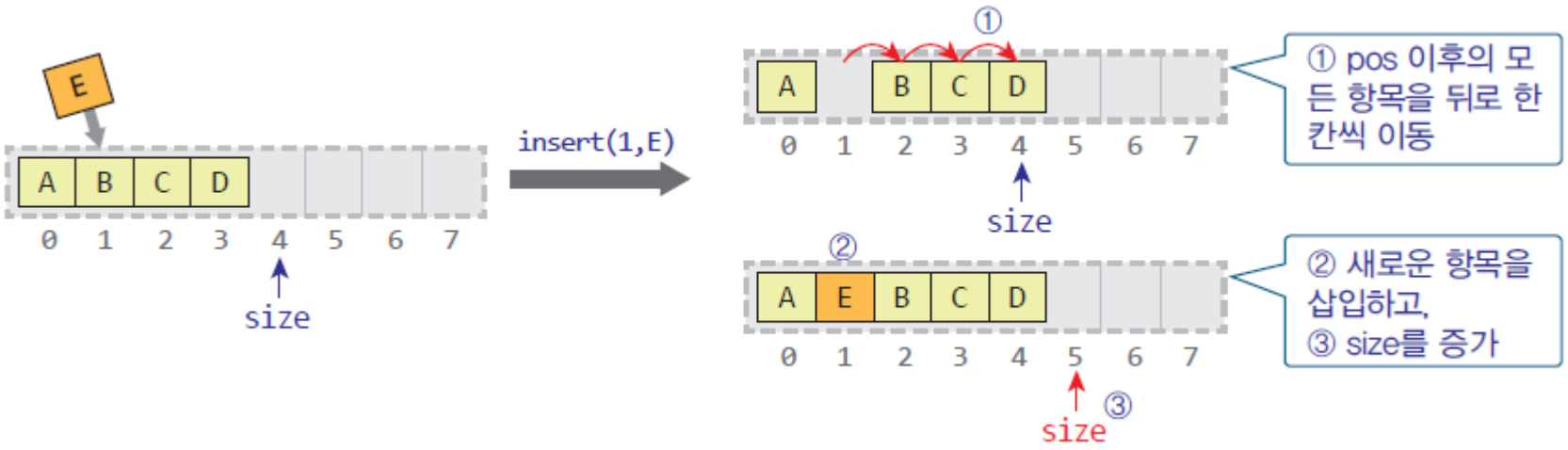
- 포화상태와 공백 상태 검사




```
def isEmpty( ):
    if size == 0 : return True
    else : return False
```

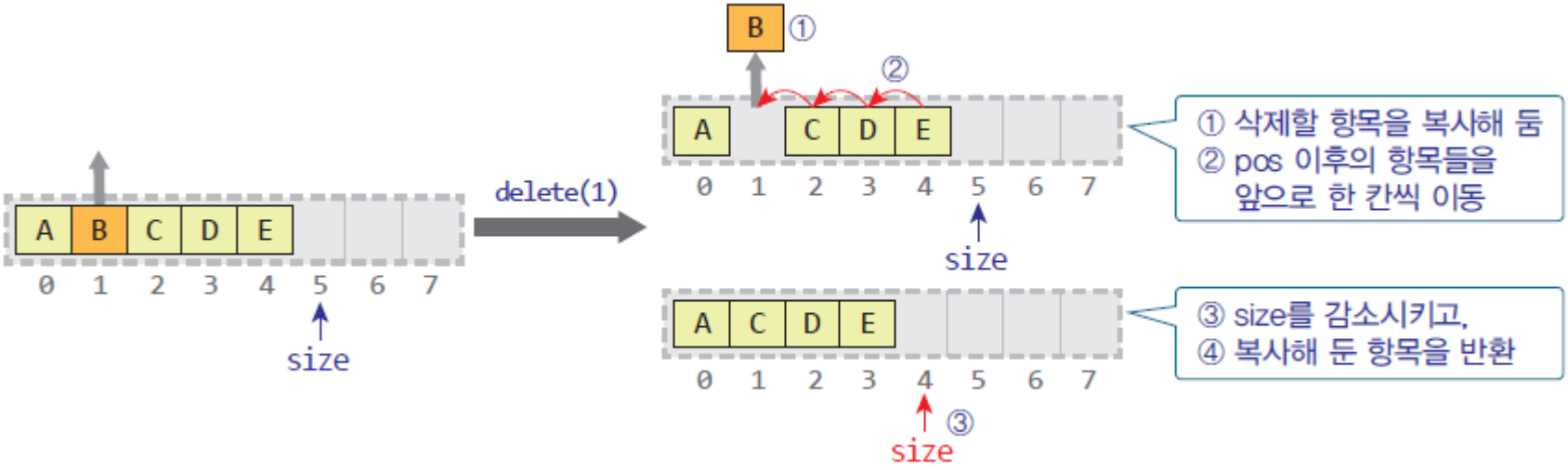
```
def isFull( ):
    return size == capacity
```

- 삽입 연산



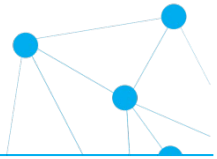
  - 코드 3.1의 19~28행
  - O(n)

- 삭제 연산



  - 코드 3.1의 30~40행
  - O(n)

테스트 프로그램

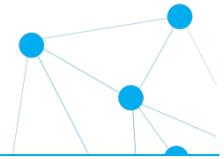


```
print("최초   ", array[0:size])
insert(0, 10)
insert(0, 20)
insert(1, 30)
insert(3, 40)
insert(2, 50)
print("삽입x5 ", array[0:size])
delete(2)
print("삭제(2)", array[0:size])
delete(3)
print("삭제(3)", array[0:size])
delete(0)
print("삭제(0)", array[0:size])
```

리스트의 0~size-1까지의 항목들만 뽑음

```
C:₩WINDOWS₩system32₩cmd.exe
최초      []
삽입x5    [20, 30, 50, 10, 40]
삭제(2)   [20, 30, 10, 40]
삭제(3)   [20, 30, 10]
삭제(0)   [30, 10]
```

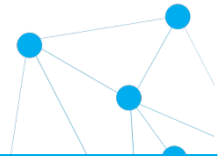
방법 2: 클래스로 구현



- 여러 개의 리스트를 사용하려면?
  - 전역 변수와 함수로는 어려움
  - 클래스가 자료구조를 구현하는 가장 좋은 방법

- 함수를 클래스로 변경하는 과정
  - 필요한 클래스를 선언한다.
  - 전역변수로 선언된 데이터 → 클래스의 멤버 변수(생성자에서)
  - 일반 함수로 구현된 연산 → 클래스의 멤버 함수
    - 첫 번째 매개변수로 self 추가
  - 멤버 함수에서 멤버 변수나 멤버 함수를 호출하기 위해 self. 사용

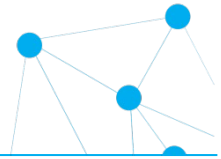
ArrayList 클래스



```
01  class ArrayList:
02      # 리스트의 데이터: 생성자에서 정의 및 초기화
03      def __init__( self, capacity=100 ):
04          self.capacity = capacity
05          self.array = [None]*capacity
06          self.size = 0
07
08      # 리스트의 연산: 클래스의 메소드
09      def isEmpty( self ):
10          return self.size == 0
11
12      def isFull( self ):
13          return self.size == self.capacity
14
15      def getEnty(self, pos) :
16          if 0 <= ps < self.size :
17              return self.array[pos]
18          else : return None
```

```
20      def insert( self, pos, e ) :
21          if not self.isFull() and 0 <= pos <= self.size :
22              for i in range(self.size, pos,-1) :
23                  self.array[i] = self.array[i-1]
24              self.array[pos] = e
25              self.size += 1
26          else : pass
27
28      def delete( self, pos ) :
29          if not self.isEmpty() and 0 <= pos < self.size :
30              e = self.array[pos]
31              for i in range(pos, self.size-1) :
32                  self.array[i] = self.array[i+1]
33              self.size -= 1
34              return e
35          else : pass
36
37      def __str__( self ) :
38          return str(self.array[0:self.size])
```

테스트 프로그램(클래스 버전)



```
from ArrayList import ArrayList
L = ArrayList(50)

print("최초   ", L)
L.insert(0, 10)
L.insert(0, 20)
L.insert(1, 30)
L.insert(L.size, 40)
L.insert(2, 50)
print("삽입x5 ", L)
L.delete(2)
print("삭제(2)", L)
L.delete(L.size-1)
print("삭제(3)", L)
L.delete(0)
print("삭제(0)", L)
```

ArrayList.py 모듈의 ArrayList를 사용함

용량이 50인 리스트 객체를 만듦. 이제 리스트를 여러 개 만들어 사용할 수 있음

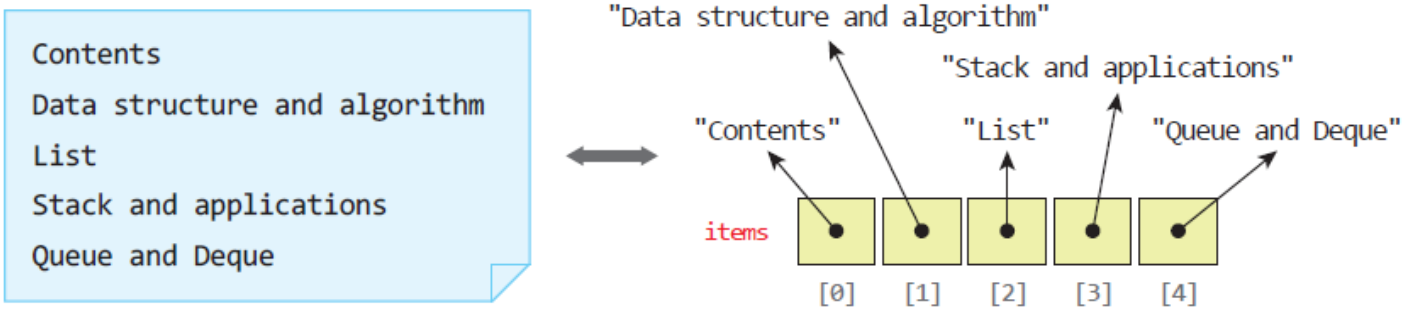
리스트의 메소드는 반드시 객체를 통해 호출해야 함. 예를 들어, L.insert(1, 30)는 리스트 객체 L에서 메소드 insert(1, 30)를 호출하도록 함

리스트 객체 L을 print()로 출력할 문자열로 변환하기 위해 ArrayList의 __str__() 메소드를 자동으로 호출해 줌

```
C:\WINDOWS\system32\cmd.exe
최초     []
삽입x5   [20, 30, 50, 10, 40]
삭제(2)  [20, 30, 10, 40]
삭제(3)  [20, 30, 10]
삭제(0)  [30, 10]
```

3.4 리스트의 응용 : 라인 편집기

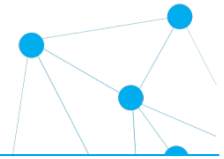
- 라인 단위로 입력이나 삭제를 할 수 있는 문서 편집기



[그림 3.13] 리스트를 이용한 문서의 내부적인 표현

- 명령 i: 라인 삽입. 행 번호와 문자열을 입력하면 그 행에 문자열을 추가함
- 명령 d: 한 라인 삭제. 행 번호를 입력하면 그 행을 삭제
- 명령 r: 한 라인 변경. 행 번호와 문자열을 입력하면 그 행의 내용을 변경
- 명령 p: 현재 내용 출력. 현재 문서의 모든 내용을 라인 번호와 함께 출력
- 명령 l: 파일 입력. 지정된 (test.txt) 파일로부터 라인을 읽어 들임
- 명령 s: 파일 출력. 지정된 (test.txt) 파일로 편집 내용을 저장

구현 코드



- ArrayList 클래스를 이용한 구현 예(코드 3.5)

```
01  from ArrayList import ArrayList
02
03  # 배열구조의 리스트를 이용한 라인 편집기 프로그램
04  list = ArrayList()
05  while True :
06      command = input("[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> ")
07
08      if command == 'i' :
09          pos = int( input("  입력행 번호: ") )
10          str = input("  입력행 내용: ")
11          list.insert(pos, str)
12
```

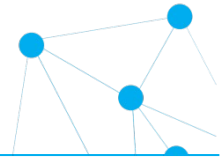
라인 편집기로 사용할 리스트 객체를 만듦

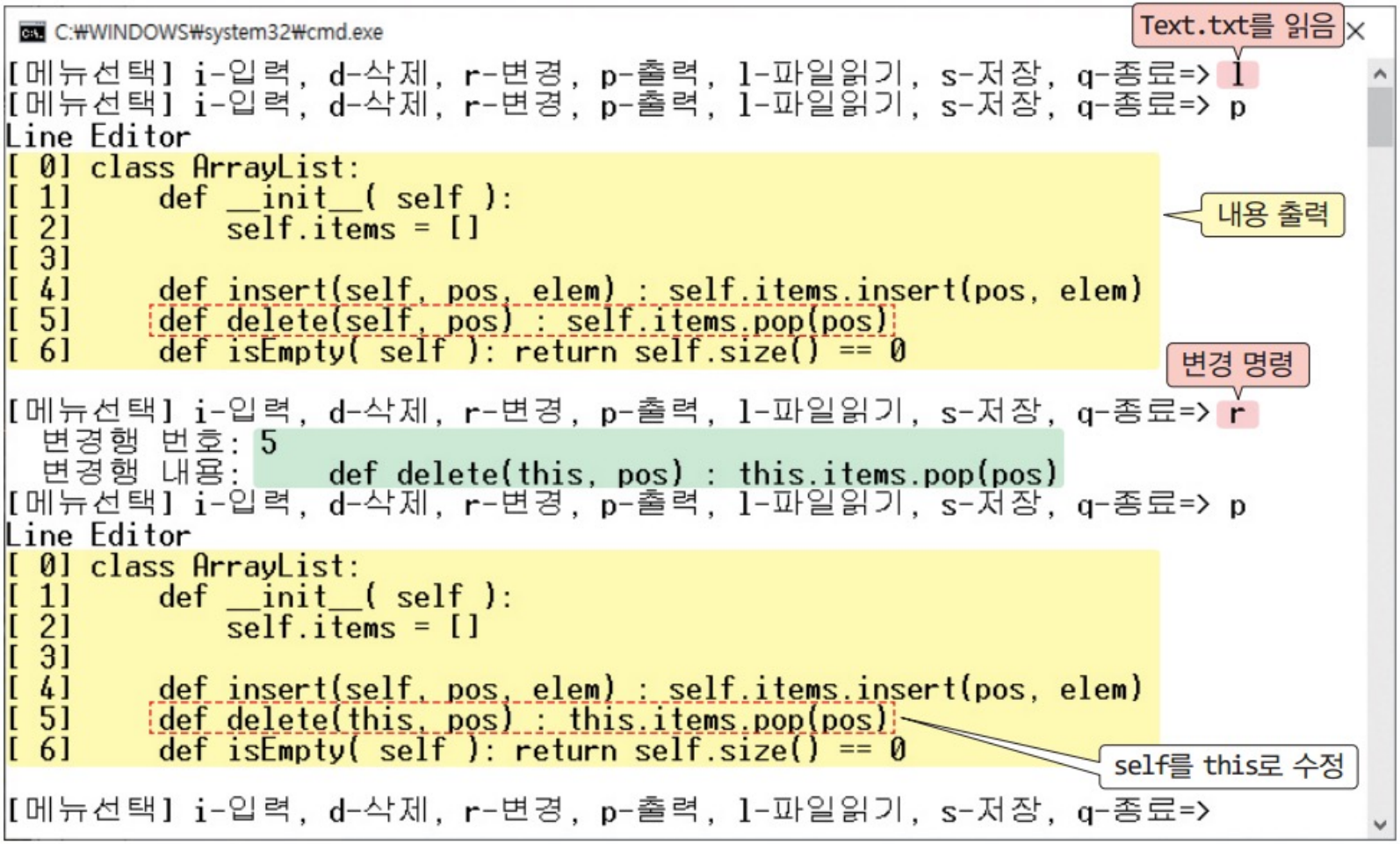
삽입 명령이면, 추가로 입력행 번호 pos와 내용 str을 순서대로 입력받고, insert() 연산을 이용해 pos행에 str을 삽입

실행 결과 예


C:₩WINDOWS₩system32₩cmd.exe
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> i
입력행 번호: 0
입력행 내용: Contents
0번 줄에 삽입
라인 추가 명령
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> i
입력행 번호: 1
입력행 내용: 자료구조와 알고리즘
1번 줄에 삽입
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> i
입력행 번호: 2
입력행 내용: 리스트
2번 줄에 삽입
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> i
입력행 번호: 3
입력행 내용: 스택, 큐, 덱
3번 줄에 삽입
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> p
화면 출력 명령
Line Editor
[0] Contents
[1] 자료구조와 알고리즘
[2] 리스트
[3] 스택, 큐, 덱
내용 출력
삭제 명령
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> d
삭제행 번호: 1
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> p
Line Editor
[0] Contents
[1] 리스트
[2] 스택, 큐, 덱
1번 줄 삭제
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=>

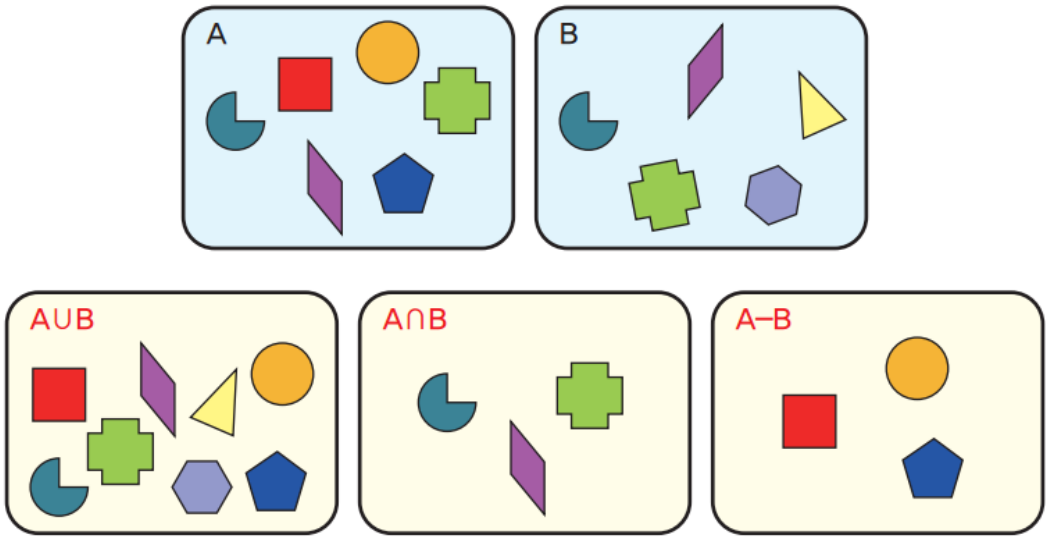
실행 결과 예



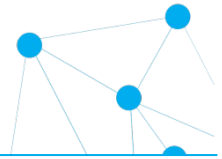

C:₩WINDOWS₩system32₩cmd.exe
Text.txt를 읽음
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> l
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> p
Line Editor
[0] class ArrayList:
[1]   def __init__(self):
[2]   self.items = []
[3]
[4]   def insert(self, pos, elem) : self.items.insert(pos, elem)
[5]   def delete(self, pos) : self.items.pop(pos)
[6]   def isEmpty(self): return self.size() == 0
내용 출력
변경 명령
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> r
변경행 번호: 5
변경행 내용: def delete(this, pos) : this.items.pop(pos)
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> p
Line Editor
[0] class ArrayList:
[1]   def __init__(self):
[2]   self.items = []
[3]
[4]   def insert(self, pos, elem) : self.items.insert(pos, elem)
[5]   def delete(this, pos) : this.items.pop(pos)
[6]   def isEmpty(self): return self.size() == 0
self를 this로 수정
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=>

3.5 집합이란?

- 집합
  - 원소의 중복을 허용하지 않음
  - 원소들 사이에 순서가 없음 → 선형 자료구조가 아님
  - $S = \{item_0, item_1, item_2, \ldots, item_{n-1}\}$



리스트의 추상 자료형

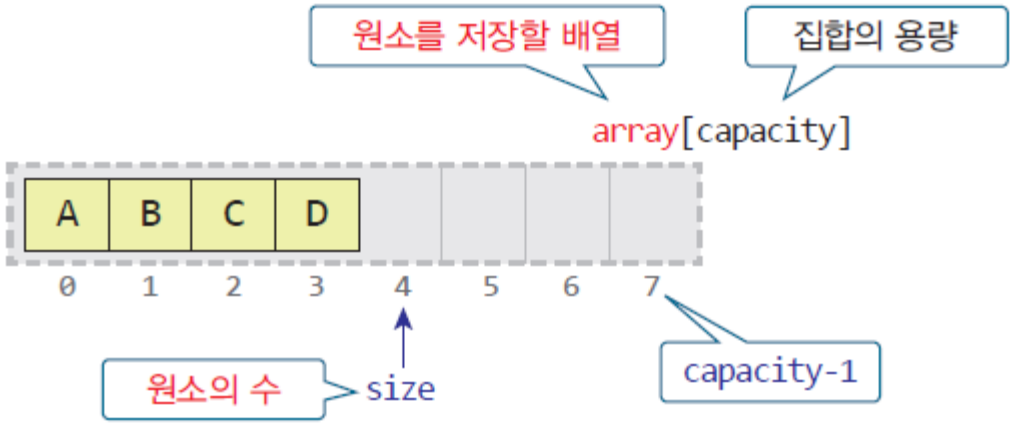


- 데이터
  - 같은 유형의 유일한 요소들의 모임. 원소들은 순서는 없지만 서로 비교할 수는 있어야 함
- 연산
  - contains(e): 집합이 원소 e를 포함하는지를 검사한다.
  - insert(e): 새로운 원소 e를 삽입한다. 중복은 허용하지 않는다.
  - delete(e): 원소 e를 집합에서 꺼내고(삭제) 반환한다.
  - isEmpty(): 공집합인지 검사한다.
  - isFull(): 집합이 가득 차 있는지를 검사한다.
  - union(setB): setB와의 합집합을 만들어 반환한다.
  - intersect(setB): setB와의 교집합을 만들어 반환한다.
  - difference(setB): setB와의 차집합을 만들어 반환한다.

3.6 집합의 구현

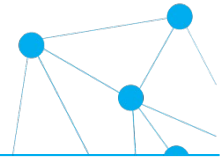
- 집합은 다양한 방법으로 구현할 수 있음
  - 배열, 비트 벡터, 트리, 해싱 구조 등
  - 사용 방법에 따라 연산들의 성능이 달라짐

- 배열을 이용한 집합의 구조



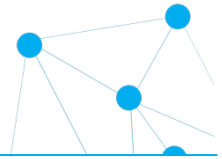
  - 원소들은 정렬하지 않음. 정렬한 경우 → 7.3절

집합의 연산들



- 원소 e가 집합에 있는지 검사하는 contains(e) 연산
  - 순차 탐색, O(n)
- 원소를 삽입하는 insert(e) 연산
  - 중복 검사, O(n)
  - 맨 뒤에 저장
- 원소를 삭제하는 delete(e) 연산
  - 삭제할 원소를 찾기, O(n)
  - 있다면 그 원소 삭제 → 이후의 모든 원소 이동
  - 개선: 맨 뒤의 원소를 삭제할 원소 위치로 옮김
- 기타 연산
  - 합집합 C = A.union(B) → $O(n^2)$
  - 교집합 C = A.intersection(B) → $O(n^2)$
  - 차집합 C = A. difference(B) → $O(n^2)$
- 클래스 코드: 코드 3.6

테스트 프로그램



```
setA = Set()
setA.insert('휴대폰')
setA.insert('지갑')
setA.insert('손수건')
setA.display('Set A:')

setB = Set()
setB .insert('빗')
setB .insert('파이썬 자료구조')
setB .insert('야구공')
setB .insert('지갑')
setB.display('Set B:')
```

```
setB.insert('빗')
setA.delete('손수건')
setA.delete('발수건')
setA.display('Set A:')
setB.display('Set B:')
```

```
setA.union(setB).display('A U B:')
setA.intersect(setB).display('A ^ B:')
setA.difference(setB).display('A - B:')
```

```
C:₩WINDOWS₩system32₩cmd.exe
Set A: ['휴대폰', '지갑', '손수건']
Set B: ['빗', '파이썬 자료구조', '야구공', '지갑']
Set A: ['휴대폰', '지갑']
Set B: ['빗', '파이썬 자료구조', '야구공', '지갑']
A U B: ['휴대폰', '지갑', '빗', '파이썬 자료구조', '야구공']
A ^ B: ['지갑']
A - B: ['휴대폰']
```

'빗'을 중복해서 넣었지만 하나만 들어 있음

합집합, 교집합, 차집합

감사합니다!