

자료구조



리스트의 구현과 활용



한국기술교육대학교
온라인평생교육원

학습내용

- **ArrayList** 클래스 구현
- 리스트의 응용: 집합의 구현

학습목표

- 배열 구조의 리스트 클래스를 구현할 수 있다.
- 리스트를 이용해 집합을 구현할 수 있다.



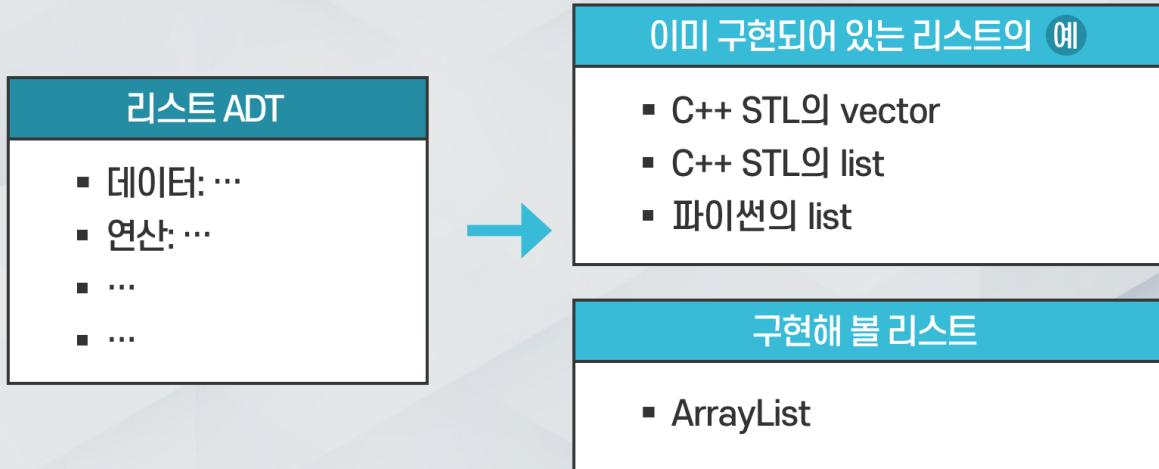
ArrayList 클래스 구현

리스트의 구현과 활용

ArrayList 클래스 구현

1 리스트 클래스의 구현

● 리스트 ADT의 구현



리스트의 구현과 활용

ArrayList 클래스 구현

1 리스트 클래스의 구현

● 리스트 ADT 추상 자료형

연산
<ul style="list-style-type: none"> ▪ <code>isFull()</code>: 리스트가 가득 차 있는지를 검사 ▪ <code>size()</code>: 리스트의 요소의 개수를 반환 ▪ <code>clear()</code>: 리스트를 초기화 ▪ <code>getEntry(pos)</code>: pos 위치에 있는 요소를 반환 ▪ <code>find(e)</code>: 리스트에서 항목 e가 있는지 찾아 인덱스를 반환 ▪ <code>replace(pos, e)</code>: pos에 있는 항목을 e로 바꿈 ▪ <code>append(e)</code>: 리스트의 맨 뒤에 새로운 항목을 추가 ▪ <code>pop()</code>: 리스트의 맨 뒤 항목을 꺼내고 반환 ▪ <code>display()</code>: 리스트를 화면에 보기 좋게 출력



ArrayList 클래스 구현

리스트의 구현과 활용

ArrayList 클래스 구현

1 리스트 클래스의 구현

◎ ArrayList 설계

배열 구조의 파이썬 “리스트”로 구현

객체(데이터)	연산(알고리즘)
<ul style="list-style-type: none"> ▪ 멤버 변수 ▪ 지정된 크기의 배열 	<ul style="list-style-type: none"> ▪ 멤버 함수 ▪ 파이썬 리스트의 고급 연산들을 사용하지 않고 구현 <p>예 insert(), pop(), append() 등</p>

리스트의 구현과 활용

ArrayList 클래스 구현

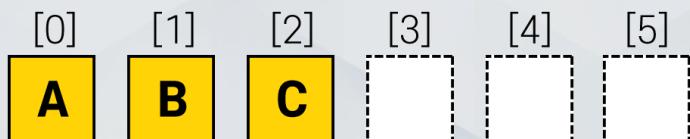
1 리스트 클래스의 구현

◎ 클래스 정의와 생성자

▶ 멤버 변수와 생성자

items[]	리스트 항목들을 저장하는 배열
capacity	리스트의 최대 용량(배열의 크기)
nItems	현재 항목의 개수

예 capacity가 6인 리스트에서 nItems가 3인 경우



ArrayList 클래스 구현



리스트의 구현과 활용

ArrayList 클래스 구현

1 리스트 클래스의 구현

● 클래스 정의와 생성자

▶ 클래스 선언

```
class ArrayList:

    def __init__( self, capacity = 10 ):           # 생성자 정의
        self.capacity = capacity                  # 리스트의 용량
        self.items = [None]*self.capacity         # 객체(데이터) : 인스턴스 변수
        self.nItems = 0                           # 리스트의 현재 원소 개수
```

리스트의 구현과 활용

ArrayList 클래스 구현

1 리스트 클래스의 구현

● 기본적인 연산들

▶ size(): 전체 항목의 수

```
def size( self ):                                # 항목의 개수
    return self.nItems
```

▶ clear(): 리스트 초기화

```
def clear( self ) :
    self.nItems = 0                               # 리스트의 현재 항목 개수
```

▶ getEntry(): pos 위치의 항목 반환

```
def getEntry(self, pos) :
    return self.items[pos]                      # pos 위치의 항목 반환
```

ArrayList 클래스 구현



리스트의 구현과 활용

ArrayList 클래스 구현

1 리스트 클래스의 구현

● 공백, 포화상태 검사

▶ isEmpty(): 공백상태 검사

```
def isEmpty( self ):  
    return self.nItems == 0 # 공백 상태 검사
```

▶ isFull(): 포화상태 검사

```
def isFull( self ):  
    return self.nItems >= self.capacity # 포화 상태 검사
```

리스트의 구현과 활용

ArrayList 클래스 구현

1 리스트 클래스의 구현

● 삽입 연산

▶ insert(): pos 위치에 원소 elem 삽입

- 포화상태를 먼저 검사해야 함
- 항목들의 이동이 필요함



```
def insert(self, pos, elem) : # pos 위치에 원소 elem 삽입  
    if not self.isFull() :  
        for i in range(self.nItems-1, pos-1, -1) :  
            self.items[i+1] = self.items[i]  
        self.items[pos] = elem  
        self.nItems += 1
```

ArrayList 클래스 구현



리스트의 구현과 활용

ArrayList 클래스 구현

1 리스트 클래스의 구현

● 삭제 연산

▶ `delete()`: pos 위치의 원소 제거

- pos가 유효한 위치인지 먼저 검사해야 함
- 항목들의 이동이 필요함



```
def delete(self, pos) :           # pos 위치의 원소 제거
    if 0 <= pos < nItems :        # 0 <= pos and pos < nItems
        for i in range(pos, self.nItems-1):
            self.items[i] = self.items[i+1]
        self.nItems -= 1
```

리스트의 구현과 활용

ArrayList 클래스 구현

1 리스트 클래스의 구현

● 후단 삽입 / 삭제 연산

▶ `append()`: 후단에 항목 삽입

```
def append(self, elem) :          # 맨 뒤에 원소 elem 삽입
    if not self.isEmpty() :
        self.items[self.nItems] = elem
        self.nItems += 1
```

▶ `pop()`: 후단 항목 삭제

```
def pop(self) :                  # 맨 마지막 원소 삭제
    if not self.isEmpty() :
        self.items[self.nItems-1] = None
        self.nItems -= 1
```

ArrayList 클래스 구현



리스트의 구현과 활용

ArrayList 클래스 구현

1 리스트 클래스의 구현

● 탐색, 대체 연산

- ▶ `find()`: 리스트에서 어떤 항목의 위치 찾기

```
def find(self, item) :          # 리스트에서 item의 위치 찾기
    for i in range(self.nItem):
        if self.items[i] == item:
            return i
    return -1
```

- ▶ `replace()`: pos번째 항목을 다른 항목으로 교체

```
def replace(self, pos, elem) :      # pos 위치의 원소 교체
    self.items[pos] = elem
```

리스트의 구현과 활용

ArrayList 클래스 구현

1 리스트 클래스의 구현

● 화면 출력 연산

- ▶ `display()`: 리스트를 보기 좋게 화면에 출력하기

```
def display(self, msg='ArrayList:'):    # 리스트를 화면에 출력
    print(msg, end='[')
    for i in range(self.nItems):
        print(self.items[i], end=', ')
    print("]")
```



ArrayList 클래스 구현

리스트의 구현과 활용

ArrayList 클래스 구현

1 리스트 클래스의 구현

● 테스트 프로그램

```
s = ArrayList()
s.display("클래스로 구현한 배열 구조의 리스트(ArrayList) 테스트")
s.insert(0, 10)
s.insert(0, 20)
s.insert(1, 30)
s.insert(s.size(), 40)
s.insert(2, 50)
s.display("클래스로 구현한 ArrayList(삽입x5): ")
s.replace(2, 90)
s.display("클래스로 구현한 ArrayList(교체x1): ")
s.delete(2)
s.delete(s.size() - 1)
s.delete(0)
s.display("클래스로 구현한 ArrayList(삭제x3): ")
s.clear()
s.display("클래스로 구현한 ArrayList(정리후): ")
```

C:\WINDOWS\system32\cmd.exe

```
클래스로 구현한 배열 구조의 리스트(ArrayList) 테스트= []
클래스로 구현한 ArrayList(삽입x5): = [20, 30, 50, 10, 40, ]
클래스로 구현한 ArrayList(교체x1): = [20, 30, 90, 10, 40, ]
클래스로 구현한 ArrayList(삭제x3): = [30, 10, ]
클래스로 구현한 ArrayList(정리후): = []
계속하려면 아무 키나 누르십시오 . . .
```

ArrayList 클래스 구현

The screenshot shows the PyCharm IDE interface with the code for the `ArrayList` class. The code is annotated with comments explaining its purpose and methods:

```

1 # 자료구조-6회차 : ArrayListClassBasic.py
2 #
3 # [6-1] ArrayList 클래스 구현하기
4 # - 용량이 미리 정해진 리스트 사용
5 =====
6
7 class ArrayList:
8     def __init__( self, capacity = 10 ):           # 생성자 정의
9         self.capacity = capacity                  # 리스트의 용량
10        self.items = [None]*self.capacity          # 객체(데이터) : 인스턴스 변수
11        self.nItems = 0                            # 리스트의 현재 원소 개수
12
13    def size( self ):                           # 원소의 개수
14        return self.nItems
15
16    def isEmpty( self ):                      # 공백 상태 검사
17        return self.nItems == 0
18
19    def isFull( self ):                       # 포화 상태 검사
20        return self.nItems >= self.capacity
21
22    def clear( self ):                         :

```

실습 단계

용량이 정해진 `ArrayList` 사용

생성자 함수에서 데이터 멤버 설정

ArrayList 크기는 사용자가 배열의 크기 자유롭게 지정

초기 설정 시 공백이기 때문에 `insert`는 0으로 설정

`class` 멤버 함수는 모두 `self`를 인자로 설정

공백 상태는 `nItems`가 0일 때, 포화 상태는 `nItems`가 용량보다 크거나 같을 때

모든 자료구조에서 삽입, 삭제 연산 중요

직접 코딩하여 하나하나 과정 확인

먼저, 리스트가 포화 상태인지 검사

먼저, 삭제할 항목의 위치가 유효한지 검사

`append()`, `pop()`은 위치 지정이 필요 없음

이 코드의 `ArrayList`는 리스트 ADT를 배열 구조로 구현한 예

Test 코드이기 때문에 if문을 통해 파일을 바로 실행(`Ctrl+F5`) 했을 때만 결과 출력

ArrayList 클래스 구현



리스트의 구현과 활용

ArrayList 클래스 구현

2 ArrayList의 복잡도 분석

● 기초적인 연산들: O(1)

```

def size( self ):
    return self.nItems # 원소의 개수

def isEmpty( self ):
    return self.nItems == 0 # 공백 상태 검사

def isFull( self ):
    return self.nItems >= self.capacity # 포화 상태 검사

def clear( self ) :
    self.nItems = 0 # 리스트의 현재 원소 개수

def getEntry(self, pos) :
    return self.items[pos] # pos 위치의 원소 반환
  
```

리스트의 구현과 활용

ArrayList 클래스 구현

2 ArrayList의 복잡도 분석

● 삽입 연산, 삭제 연산: O(n)

▶ 항목들의 이동이 필요함: 항목의 수 n의 비례

```

def insert(self, pos, elem) : # pos 위치에 원소 elem 삽입
    if not self.isFull() :
        for i in range(self.nItems-1, pos-1, -1) :
            self.items[i+1] = self.items[i]
        self.items[pos] = elem
        self.nItems += 1

def delete(self, pos) : # pos 위치의 원소 제거
    if 0 <= pos < nItems :
        for i in range(pos, self.nItems-1):
            self.items[i] = self.items[i+1]
        self.nItems -= 1
  
```

ArrayList 클래스 구현



리스트의 구현과 활용

ArrayList 클래스 구현

2 ArrayList의 복잡도 분석

● 후단 삽입, 삭제 연산

► O(1)

```
def append(self, elem) :           # 맨 뒤에 원소 elem 삽입
    if not self.isEmpty() :
        self.items[self.nItems] = elem
        self.nItems += 1
```

```
def pop(self) :                   # 맨 마지막 원소 삭제
    if not self.isEmpty() :
        self.items[self.nItems-1] = None
        self.nItems -= 1
```

리스트의 구현과 활용

ArrayList 클래스 구현

2 ArrayList의 복잡도 분석

● 기타 연산들

► 탐색 연산: O(n)

```
def find(self, item) :           # 리스트에서 item의 위치 찾기
    for i in range(self.nItem):
        if self.items[i] == item:
            return i
    return -1
```

► 교체 연산: O(1)

```
def replace(self, pos, elem) :    # pos 위치의 원소 교체
    self.items[pos] = elem
```

ArrayList 클래스 구현



리스트의 구현과 활용

ArrayList 클래스 구현

2 ArrayList의 복잡도 분석

● 기타 연산들

▶ 화면 출력 연산: $O(n)$

```
def display(self, msg='ArrayList:'):
    print(msg, end=' = [ ')
    for i in range(self.nItems):
        print(self.items[i], end=', ')
    print("]")
```

리스트의 구현과 활용

ArrayList 클래스 구현

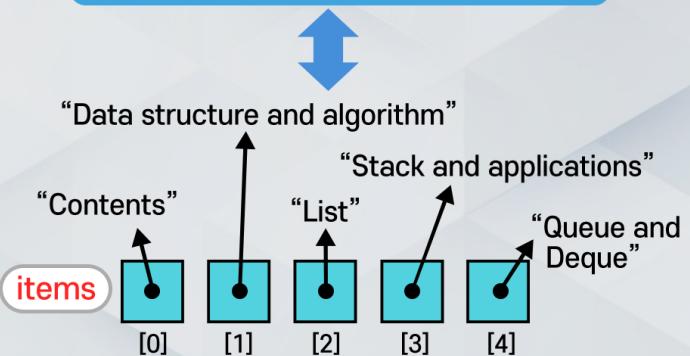
2 ArrayList의 복잡도 분석

● 리스트의 활용

다양한 문제 해결을 위해 사용

- 목록, 계획표 만들기
- 라인 편집기
- 다항식의 연산 프로그램
- 버킷 리스트

- Contents
- Data structure and algorithm
- List
- Stack and applications
- Queue and Deque



ArrayList 클래스 구현



리스트의 구현과 활용

ArrayList 클래스 구현

2 ArrayList의 복잡도 분석

● 리스트의 활용

다른 자료 구조를
구현하기 위해 사용

- Bag
- Set
- 트리나 그래프 등

리스트의 응용: 집합의 구현



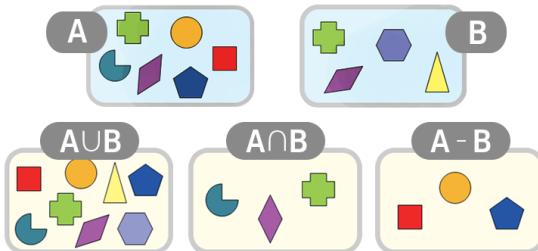
리스트의 구현과 활용

리스트의 응용: 집합의 구현

1 집합의 개념

집합

- 리스트와 유사하지만 원소의 중복을 허용하지 않는 자료구조
- 원소들 사이에 순서가 없음: 선형 자료구조가 아님
- $S = \{ item_0, item_1, item_2, \dots item_{n-1} \}$
- 특징적인 연산들



리스트의 구현과 활용

리스트의 응용: 집합의 구현

1 집합의 개념

● 집합의 추상 자료형

객체(데이터)

- 같은 유형의 **유일한** 원소들의 모임

연산

- **Set()**: 비어 있는 새로운 집합을 만듦
- **insert(e)**: 새로운 원소 e를 삽입함. 이미 e가 있다면 삽입하지 않음
- **delete(e)**: 원소 e를 집합에서 꺼내고(삭제) 반환
- **isEmpty()**: 공집합인지를 검사

리스트의 응용: 집합의 구현



리스트의 구현과 활용

리스트의 응용: 집합의 구현

1 집합의 개념

● 집합의 추상 자료형

연산

- `size()`: 집합의 원소의 개수를 반환
- `contains(e)`: 집합이 원소 e를 포함하는지를 검사
- `equals(setB)`: setB와 같은 집합인지를 검사
- `union(setB)`: setB와의 합집합을 만들어 반환
- `intersect(setB)`: setB와의 교집합을 만들어 반환
- `difference(setB)`: setB와의 차집합을 만들어 반환
- `display()`: 집합을 화면에 출력

리스트의 구현과 활용

리스트의 응용: 집합의 구현

1 집합의 개념

● 집합의 구현 방법들

▶ 집합은 다양한 방법으로 구현할 수 있음

리스트, 비트 벡터, 트리, 해싱 구조 등

▶ 파이썬에서 시퀀스 자료형으로 집합(set)을 제공

```
s1 = {1,2,3}           # 집합 객체
s2 = {2,3,4,5}         # 집합 객체
s3 = s1.union(s2)      # 합집합
s4 = s1.intersection(s2) # 교집합
s5 = s1 - s2          # 차집합
print( "s1" , s1)
print( "s2" , s2)
print( "s3" , s3)
print( "s4" , s4)
```

```
print( "s5" , s5)
s5 = {3.14}            # 원소가 하나인 집합
map = {3.4: 'phi'}    # 엔트리가 하나인 딕셔너리
```

C:\#WIND... — ×
s1: {1, 2, 3}
s2: {2, 3, 4, 5}
s3: {1, 2, 3, 4, 5}
s4: {2, 3}
s5: {1}



리스트의 응용: 집합의 구현

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

리스트를 이용해 객체(데이터) 표현

- 데이터 멤버로 리스트를 가짐
- 앞에서 구현한 배열 구조의
리스트 클래스(ArrayList) 사용

집합의 연산(알고리즘)

- 리스트의 연산들(메소드)을
이용해서 구현

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

● 클래스 정의와 생성자

▶ 다른 모듈(파일)에 구현된 ArrayList 클래스를 불러옴

- 구현된 리스트 클래스 이름: `ArrayList`
- 코드가 저장된 파일 이름: `ArrayListClassBasic.py`

```
from ArrayListClassBasic import ArrayList
```

▶ 클래스 정의와 집합의 데이터 멤버

- 집합의 원소를 리스트(ArrayList)에 저장 → `self.list`
- 집합의 최대 용량을 지정할 수 있도록 함

```
class Set:
    def __init__( self, capacity=10 ):      # 생성자
        self.list = ArrayList(capacity)     # 최대 항목수를 지정한 리스트
```



리스트의 응용: 집합의 구현

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

● 클래스의 연산들

▶ 집합의 연산들: 리스트의 메소드를 이용해 구현

▶ 집합의 원소 개수

```
def size( self ):
    return self.list.size()
```

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

● 기본 연산들

▶ 원소의 포함 검사

리스트의 find() 연산 사용: O(n)

- 원소가 있으면: 인덱스 반환 (0이상) → True
- 원소가 없으면: -1 반환 → False

```
def contains(self, elem) :
    if self.list.find(elem) >= 0 :
        return True
    else : return False
    # elem이 원소인지 검사
    # elem이 리스트에 있으면
    # 이 집합에 포함됨→True
    # 포함되지 않음→False
```



리스트의 응용: 집합의 구현

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

◎ 기본 연산들

- ▶ 공백 상태 검사

```
def isEmpty( self ):
    return self.size()==0
```

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

◎ 삽입 연산

- ▶ 원소가 이미 있는지 먼저 검사: $O(n)$
- ▶ 없는 경우 리스트의 맨 뒤에 삽입: $O(1)$

```
def insert(self, elem) :
    if not self.contains(elem) :          # 삽입 연산
        self.list.insert(self.size(), elem) # 포함하고 있지 않으면
                                            # 리스트의 맨 뒤에 삽입
```



리스트의 응용: 집합의 구현

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

● 삭제 연산

▶ 원소를 먼저 찾아보고, 있는 경우 ‘리스트’에서 삭제: $O(n)$

```
def delete(self, elem) :          # 집합의 삭제 연산
    id = self.list.find(elem)     # 리스트에 있는지 검사
    if id >= 0 :                 # 있으면
        self.list.delete(id)      # 리스트에서 삭제
```

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

● 동일집합 검사

▶ 원소의 개수가 같으면서 하나가 다른 하나의 부분 집합인 경우

```
def __eq__( self, setB ) :          # 같은 집합인지 검사
    if self.size() != setB.size() :   # 원소의 개수가 같고
        return False
    else :
        return self.isSubsetOf( setB ) # 부분집합이면 동일집합
```



리스트의 응용: 집합의 구현

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

● 부분집합 검사

▶ 모든 원소가 모두 포함되어 있어야 함

```
def isSubsetOf( self, setB ):
    for i in self.size():
        elem = self.list.getEntry()
        if not setB.contains(elem):
            return False
    return True
```

부분집합 검사
self의 원소 elem이
setB에 없으면
부분집합이 아님
모든 원소가 있으면 부분집합

▶ 시간 복잡도는?

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

● 집합의 복사

▶ 내부의 리스트를 복사해야 함: $O(n)$

```
def clone( self ):
    newSet = Set()
    for i in range(self.size()):
        newSet.insert(self.list.getEntry(i))
    return newSet
```

동일한 집합 만들기



리스트의 응용: 집합의 구현

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

● 합집합

```
def union( self, setB ):
    newSet = self.clone()
    for i in range(setB.size()) :
        newSet.insert(setB.list.getEntry(i))
    return newSet
```

▶ 복잡도: $O(n^2)$

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

● 교집합

```
def intersect( self, setB ):
    newSet = Set()
    for i in range(self.size()) :
        e1 = self.list.getEntry(i)
        for j in range(setB.size()):
            e2 = setB.list.getEntry(j)
            if e1 == e2 :
                newSet.insert(e1)
    return newSet
```

C = self ^ B
self의 원소 e1과
같은 원소 e2가 있으면
그 원소를 교집합에 추가

리스트의 응용: 집합의 구현



리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

◎ 차집합

```
def difference( self, setB ) :           # C = self - B
    newSet = self.clone()
    for i in range(setB.size()) :
        newSet.delete(setB.list.getEntry(i))
    return newSet
```

리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

◎ 리스트로 구현한 집합 테스트

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

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

```
# 빗, 자료구조, 야구공, 지갑

setA.delete('손수건')
setA.delete('발수건')
setA.display('Set A:')
# 휴대폰, 지갑

setB.insert('빗')
setB.display('Set B:')
# 빗, 자료구조, 야구공, 지갑
```

리스트의 응용: 집합의 구현



리스트의 구현과 활용

리스트의 응용: 집합의 구현

2 리스트를 이용한 집합의 구현

● 리스트로 구현한 집합 테스트

```
s0 = setA.union(setB)
s0.display('A U B:')
# 휴대폰, 지갑, 빗, 자료구조, 야구공

s1 = setA.intersect(setB)
s1.display('A ^ B:')
# 지갑

s2 = setA.difference(setB)
s2.display('A - B:')
# 휴대폰
```

C:\WINDOWS\system32\cmd.exe

```
Set A:= [휴대폰, 지갑, 손수건, ]
Set B:= [빗, 자료구조, 야구공, 지갑, ]
Set A:= [휴대폰, 지갑, ]
Set B:= [빗, 자료구조, 야구공, 지갑, ]
A U B:= [휴대폰, 지갑, 빗, 자료구조, 야구공, ]
A ^ B:= [지갑, ]
A - B:= [휴대폰, ]
```

리스트의 응용: 집합의 구현

```

# 자료구조-6회차 : SetByArrayList.py
#
# [6-2] 집합 자료구조 만들기
# - 앞에서 구현한 ArrayList를 이용해 집합 자료구조를 구현함
#=====

from ArrayListClassBasic import ArrayList

# Set.
class Set:
    def __init__( self, capacity=10 ): # 생성자
        self.list = ArrayList(capacity) # 최대 항목수를 지정한 리스트

    def size( self ):
        return self.list.size()

    def contains(self, elem) : # elem이 원소인지 검사
        if self.list.find(elem) >= 0 : # 원소가 이미 있으면
            return True
        else : return False

```

실습 단계

집합이라는 자료구조를 구현하기 위해 ArrayList 사용

ArrayList를 바로 사용 가능

이 list는 파이썬의 list가 아닌 우리가 만든 ArrayList

find(elem): 항목이 리스트에 있으면 인덱스 반환, 없으면 -1 반환

집합에서 원소는 중복될 수 없음

insert에서는 맨 뒤에 삽입

delete에서는 삭제할 원소가 집합에 존재해야 함

합집합은 복제하여 새로운 집합을 생성해 시작

교집합은 공백 집합으로 시작