# 5.2 Linear Data Structure

## 5.2 선형 자료 구조

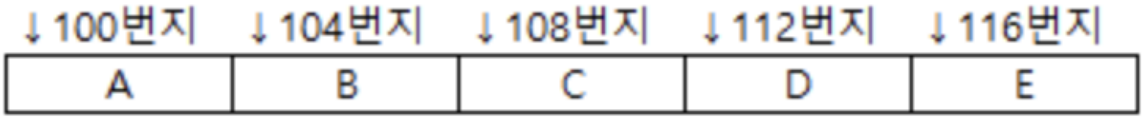
요소가 일렬로 나열되어 있는 자료 구조

### 선형 리스트(Linear List)

- 장점
  - 배열에 구성했기 때문에 단순하다.
  - 노드의 번지가 순차적이다.
  - 물리적 순서, 논리적 순서가 동일하여 데이터를 찾기 간단하다.
  - 프로그램 구현이 쉽다.
- 단점
  - 데이터 삽입, 삭제에 많은 작업이 필요하다.

    **eg.**

    100만 개의 선형 리스트 맨 앞에 데이터를 삽입하려면 100만개를 뒤로 이동시켜야 한다.
    ⇒ 오버헤드 발생!

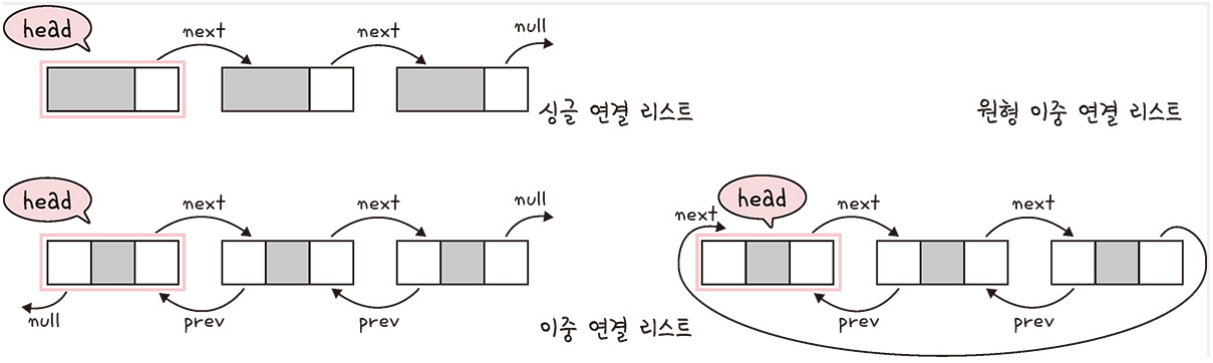



## 5.2.1 연결 리스트

💡 **연결 리스트**
**정의**
-
각 노드가 데이터와 포인터를 가지고 한 줄로 연결되어 있는 방식으로 데이터 저장
- 동적인 데이터 추가/삭제에 유리
- 삽입, 삭제 O(1)
  탐색 O(n)

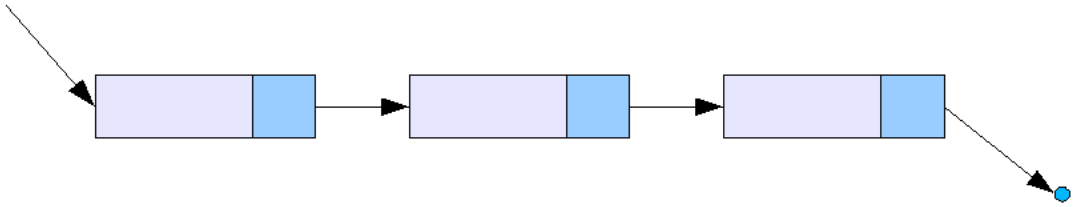
**종류**
- 단일 연결 리스트
- 이중 연결 리스트
- 원형 연결 리스트

# 구조



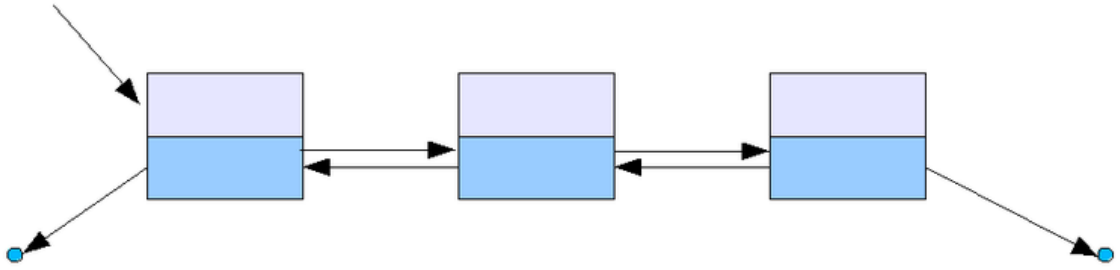

## 단일 연결 리스트

- 각 노드에 자료 공간과 한 개의 포인터 공간이 있다.
- 포인터는 다음 노드를 가리킨다.



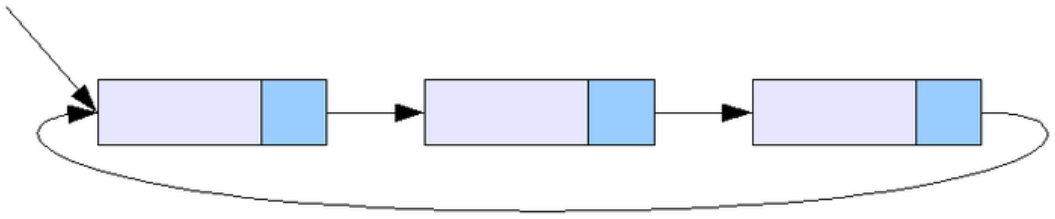
## 이중 연결 리스트의 구조

- 단일 연결 리스트와 비슷하지만, 포인터 공간이 두 개가 있다.
- 각각의 포인터는 앞, 뒤의 노드를 가리킨다.



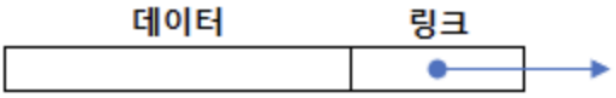
## 원형 연결 리스트의 구조

- 일반적인 연결 리스트에 마지막 노드와 처음 노드를 연결시켜 원형으로 만든 구조



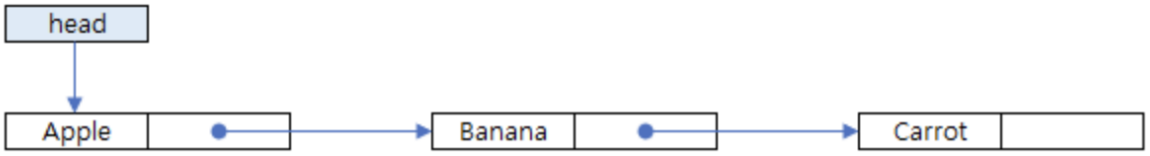
## 설명 - 단순 연결 리스트(Singly Linked List)

- 선형 리스트와 다르게 노드들이 물리적으로 떨어진 곳에 위치한다.
    - 노드의 번지도 100, 200, 130 등 순차적이지 않다.
- 데이터와 링크로 구성된 **노드(Node)**로 이루어진다.



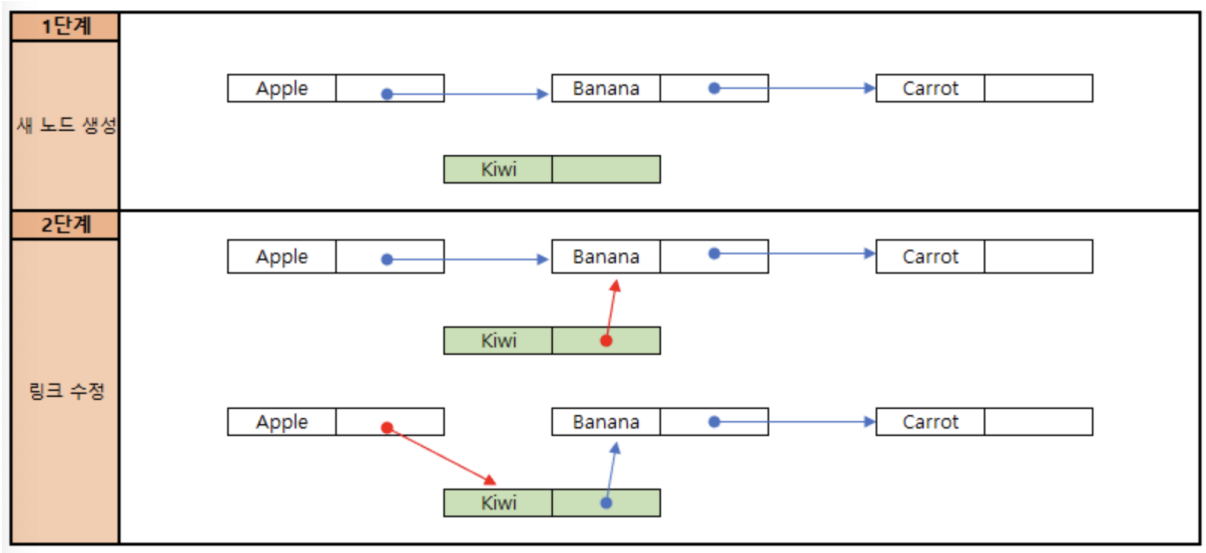

노드의 구조

- 첫 번째 노드를 가리키는 헤드(Head)를 사용한다.



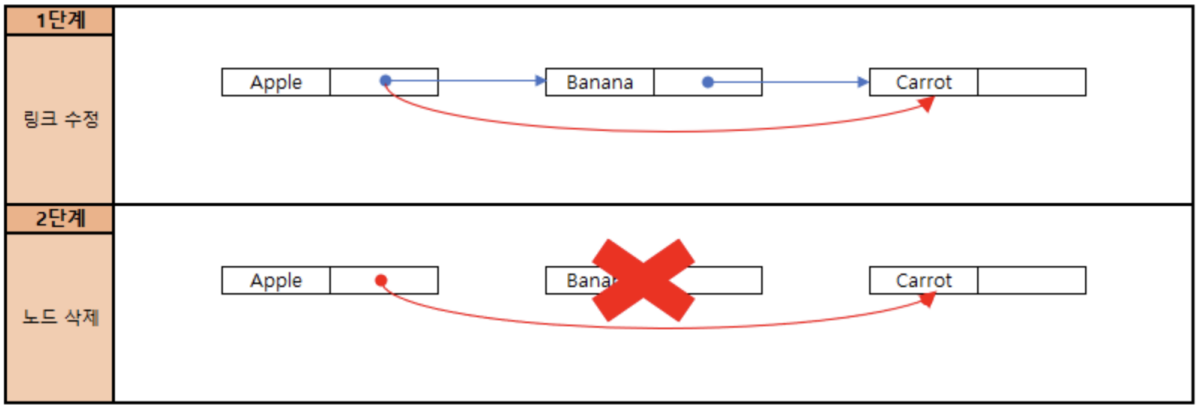

## 노드(데이터) 삽입

1. 삽입할 노드 생성
2. 순서에 맞게 링크 생성
    a. 새 노드의 링크에 이전 노드의 링크 복사
    b. 이전 노드의 링크에 새 노드 지정




## 노드(데이터) 삭제

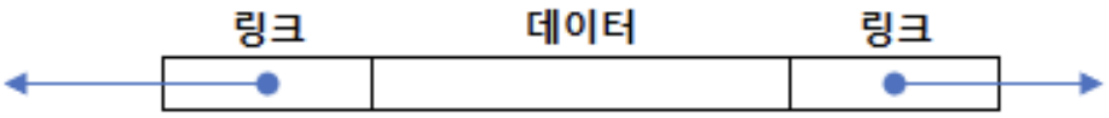
1. 삭제할 노드의 바로 앞 노드 링크 수정
    a. 삭제할 노드의 링크를 바로 앞 노드의 링크로 복사
    b. 앞 노드가 삭제할 노드의 다음 노드를 가리키게 한다.
2. 노드 삭제




## 설명 - 이중 연결 리스트(Doubly Linked List)

- 양방향으로 링크가 연결되어 노드의 양방향으로 접근이 용이한 리스트

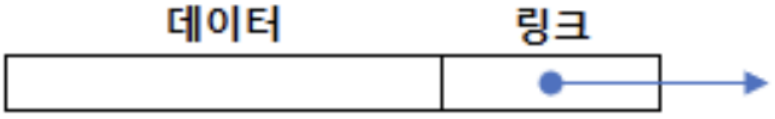

- 2개의 링크와 1개의 데이터가 한 노드로 구성
  - 이전 노드를 가리키는 링크 : plink
  - 다음 노드를 가리키는 링크 : nlink



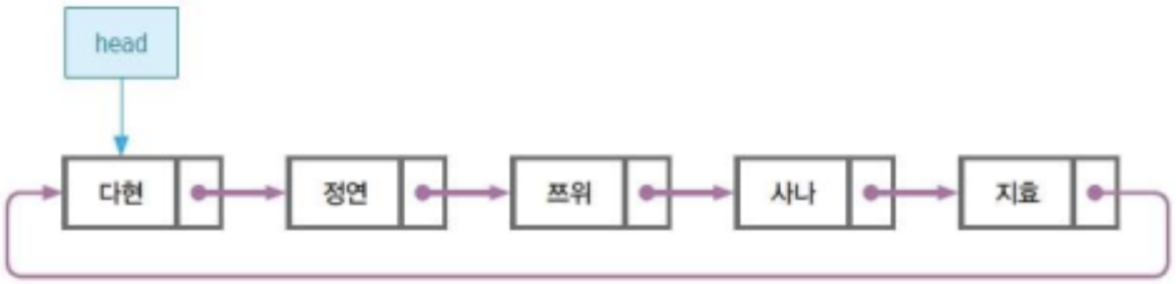

## 설명 - 원형 연결 리스트(Circular Linked List)

- 단순 연결 리스트는 끝까지 방문한 뒤에는 더 이상 방문할 곳이 없어서 다시 방문하려면 헤드부터 재시작해야 한다.
- 원형 연결 리스트는 마지막 노드가 다시 첫 번째 노드를 기리키도록 설정되어 형태가 원형태로 구성된다.

- 구조



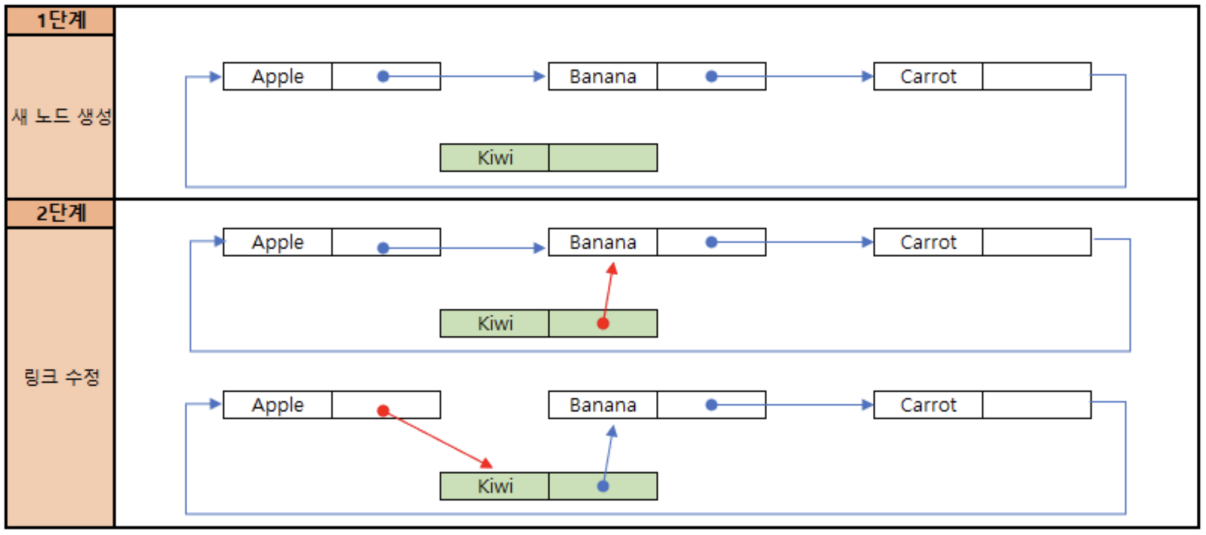

노드의 구조




노드로 구성된 원형 연결 리스트

## 노드(데이터) 삽입

1. 삽입할 노드 생성
2. 순서에 맞게 링크 수정
    a. 새 노드의 링크에 이전노드의 링크 복사
    b. 이전 노드의 링크에 새 노드 지정

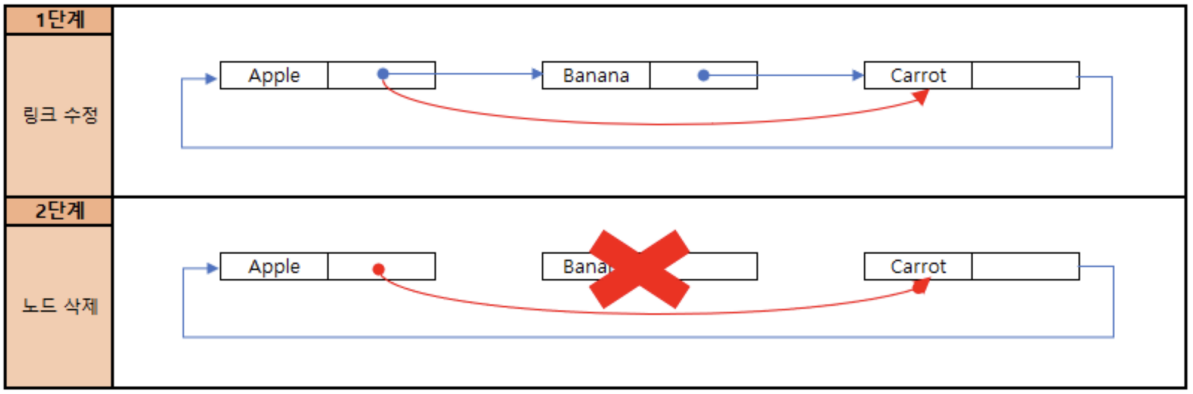



원형 연결 리스트에서 중간 위치에 노드를 삽입하는 과정

## 노드(데이터) 삭제

1. 삭제할 노드의 바로 앞 노드 링크 수정
    a. 삭제할 노드의 링크를 바로 앞 노드의 링크로 복사, 앞 노드가 삭제할 노드의 다음 노드를 가리키게 한다.

2. 삭제할 노드 삭제



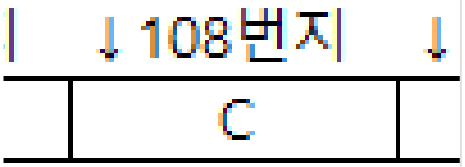

원형 연결 리스트에서 중간 위치의 데이터를 삭제하는 과정

참고)

[Python] 단순 연결 리스트(Singly Linked List)

단순 연결 리스트(Singly Linked List) 단순 연결 리스트의 개념 선형 리스트(Linear List) 장점 배열에 구성하였기 때문에 단순하다. 물리적인 순서와 논리적인 순서가 동일하여 데이터를 찾기 간단하다. 프로

https://starrykss.tistory.com/1837




## 5.2.2 배열

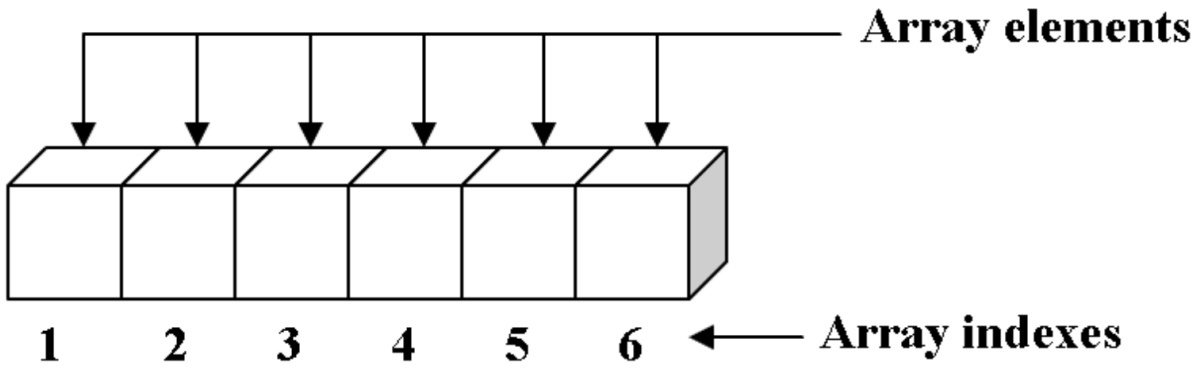
💡 **배열**
-
**같은 타입의 데이터**를 여러개 나열한 선형 자료구조
(파이썬의 배열은 리스트, 튜플 같은 자료형)
(파이썬은 다른 데이터 타입도 한 배열에 담을 수 있다.)
- 연속적인 메모리 공간에 순차적으로 데이터 저장
- 삽입, 삭제 O(n)
탐색O(1)

* 데이터 추가, 삭제를 많이 한다면 연결리스트
* 탐색을 많이 한다면 배열

1. 랜덤 접근
2. 순차적 접근

## 구성

- 인덱스(Index)
- 요소(Element)




## 특징

- 최초에 메모리 크기를 정적으로 할당
- 삽입 순서대로 저장
- 중복 데이터 삽입 가능
- 데이터 수정 가능

- 배열 안에 배열 배치 가능(다차원 배열)

## 장점

- 구현이 쉽다.
- 추가적인 메모리가 필요하지 않다.
- 인덱스를 통해 빠른 접근이 가능하다.
    - 인덱스를 알고 있다면 접근의 **시간 복잡도는 O(n)**
- 연속 데이터로 메모리 관리가 편하다.

## 단점

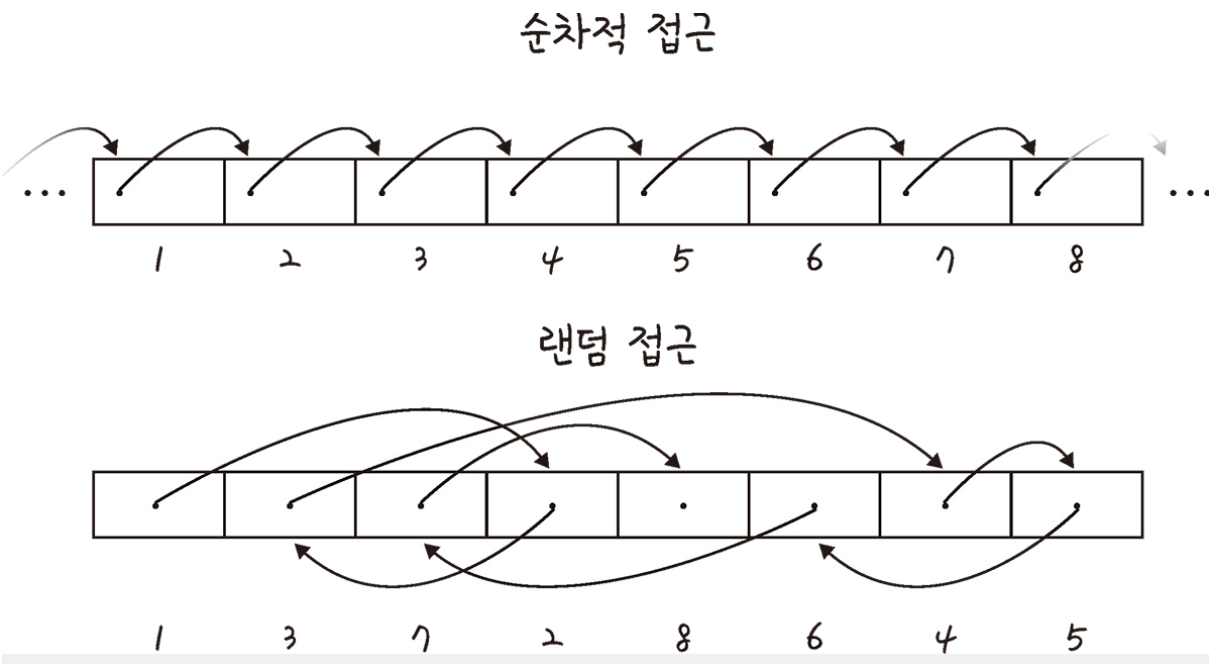
- 최초 정의된 크기를 변경할 수 없다.
- 사용하지 않는 공간도 메모리를 차지해서 낭비가 발생한다.
- 중간 요소의 삽입, 삭제가 오래 걸린다.
    - 기존 데이터를 모두 한 칸씩 밀어내고 데이터를 삽입
    - **시간복잡도 O(n)**

## 랜덤 접근(직접 접근)

- 순차적인 데이터가 있을 때 임의의 인덱스에 해당하는 데이터에 접근할 수 있는 기능

## 순차적 접근

- 데이터를 저장된 순서대로 검색




## 연결 리스트와 배열

### 탐색

- 연결리스트는 순차적으로 탐색해야 하기 때문에 느리다.
- 배열은 인덱스를 알면 바로 접근할 수 있기 때문에 빠르다.

### 삽입, 삭제

- 연결리스트는 연결된 노드만 바꿔주면 되기 때문에 빠르다.
- 배열은 모든 요소들을 옮겨줘야 하기 때문에 느리다.