# 선수 지식 - 자료구조
## 배열

배열 | 다양한 알고리즘의 기본이 되는 자료구조 이해하기

강사 나동빈

# 선수 지식 - 자료구조

## 배열

# 배열(Array)



- **가장 기본적인 자료구조**다.
- 여러 개의 변수를 담는 공간으로 이해할 수 있다.
- 배열은 **인덱스(index)**가 존재하며, 인덱스는 0부터 시작한다.
- 특정한 인덱스에 직접적으로 접근 가능 → 수행 시간: $O(1)$

| 인덱스 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| 값 | 25 | 75 | 83 | 59 | 24 | 72 | 55 | 17 | 42 |

# 배열의 특징



- 컴퓨터의 메인 메모리에서 배열의 공간은 연속적으로 할당된다.

**장점**: **캐시(cache)** 히트 가능성이 높으며, 조회가 빠르다.

**단점**: 배열의 크기를 미리 지정해야 하는 것이 일반적이므로, 데이터의 추가 및 삭제에 한계가 있다.

| 메모리 주소 | 0x0000 | 0x0004 | 0x0008 |
|---|---|---|---|
| 인덱스 | 0 | 1 | 2 |
| 값 | 25 | 75 | 83 |

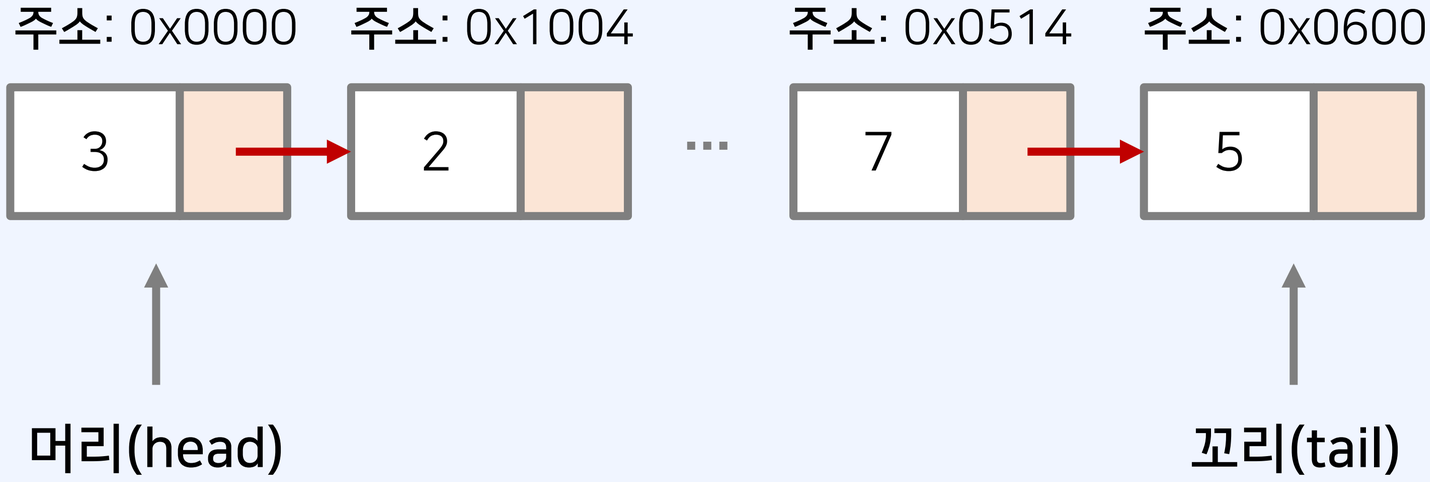
# 연결 리스트(Linked List)



- 컴퓨터의 메인 메모리상에서 주소가 연속적이지 않다.
- 배열과 다르게 크기가 정해져 있지 않고, 리스트의 크기는 동적으로 변경 가능하다.

**장점**: **포인터(pointer)**를 통해 다음 데이터의 위치를 가리킨다는 점에서 **삽입과 삭제가 간편**하다.

**단점**: 원소를 검색할 때는 앞에서부터 원소를 찾아야 하므로, 데이터 검색 속도가 느리다.



# 파이썬의 리스트(List) 자료형



- 파이썬에서는 **리스트 자료형**을 제공한다.

[알아 둘 점]

- 컴퓨터 공학에서의 연결 리스트와 다른 의미를 가진다.
- 일반적인 프로그래밍 언어에서의 **배열**로 이해할 수 있다.
- 파이썬의 리스트는 배열처럼 임의의 인덱스를 이용해 직접적인 접근이 가능하다.

# 파이썬의 리스트(List) 자료형



- 파이썬의 리스트 자료형은 **동적 배열**이다.
- 배열의 용량이 가득 차면, 자동으로 크기를 증가시킨다.
- 내부적으로 포인터(pointer)를 사용하여, 연결 리스트의 장점도 가지고 있다.
- **배열(array) 혹은 스택(stack)**의 기능이 필요할 때 리스트 자료형을 그대로 사용할 수 있다.
- 큐(queue)의 기능을 제공하지 못한다. (비효율적)

# 리스트 컴프리헨션(List Comprehension)



- 파이썬에서는 임의의 크기를 가지는 배열을 만들 수 있다.
- 일반적으로 **리스트 컴프리헨션(list comprehension)**을 사용한다.
- 크기가 $N$인 1차원 배열을 만드는 방법은 다음과 같다.

```
# [0, 0, 0, 0, 0]
n = 5
arr = [0] * n
print(arr)

# [0, 1, 2, 3, 4]
n = 5
arr = [i for i in range(n)]
print(arr)
```

**[실행 결과]**

```
[0, 0, 0, 0, 0]
[0, 1, 2, 3, 4]
```

# 크기가 N X M인 2차원 리스트(배열) 만들기 ①



- 2차원 배열이 필요할 때는 다음과 같이 초기화한다.

```
n = 3
m = 5
arr = [[0] * m for i in range(n)]
print(arr)
```

[실행 결과]

```
[
  [0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0]
]
```

# 크기가 N X M인 2차원 리스트(배열) 만들기 ②



- 2차원 배열이 필요할 때는 다음과 같이 초기화한다.

```
n = 3
m = 5
arr = [[i * m + j for j in range(m)] for i in range(n)]
print(arr)
```

[실행 결과]

```
[
  [0, 1, 2, 3, 4],
  [5, 6, 7, 8, 9],
  [10, 11, 12, 13, 14]
]
```

# 배열을 초기화할 때 유의할 점



- 리스트는 기본적으로 메모리 주소를 반환한다.
- 따라서 단순히 $[[0] * m] * n$ 형태로 배열을 초기화하면 안 된다.

→ 이렇게 초기화를 하게 되면, $n$개의 $[0] * m$ 리스트는 모두 같은 객체로 인식된다.

→ 다시 말해, 같은 메모리를(동일한 리스트를) 가리키는 $n$개의 원소를 담는 리스트가 된다.

# 배열을 초기화할 때 유의할 점



- 2차원 배열을 초기화할 때는 리스트 컴프리헨션을 이용하는 것이 일반적이다.

```
n = 3
m = 5
arr1 = [[0] * m] * n
arr2 = [[0] * m for i in range(n)]

arr1[1][3] = 7
arr2[1][3] = 7

print(arr1)
print(arr2)
```

[실행 결과]

```
[
  [0, 0, 0, 7, 0],
  [0, 0, 0, 7, 0],
  [0, 0, 0, 7, 0]
]
[
  [0, 0, 0, 0, 0],
  [0, 0, 0, 7, 0],
  [0, 0, 0, 0, 0]
]
```

# 배열 직접 초기화



- 자신이 원하는 임의의 값을 넣어 곧바로 사용할 수 있다.

```
arr = [0, 1, 2, 3, 4, 5, 6, 7, 8]
print(arr)
```

[실행 결과]

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```