가천대학교
Gachon University

02

Algorithm

자료구조 기초

2025-03-14

조윤실

목 차

※ 가천대학교 컴퓨터공학과 '알고리즘' 과정

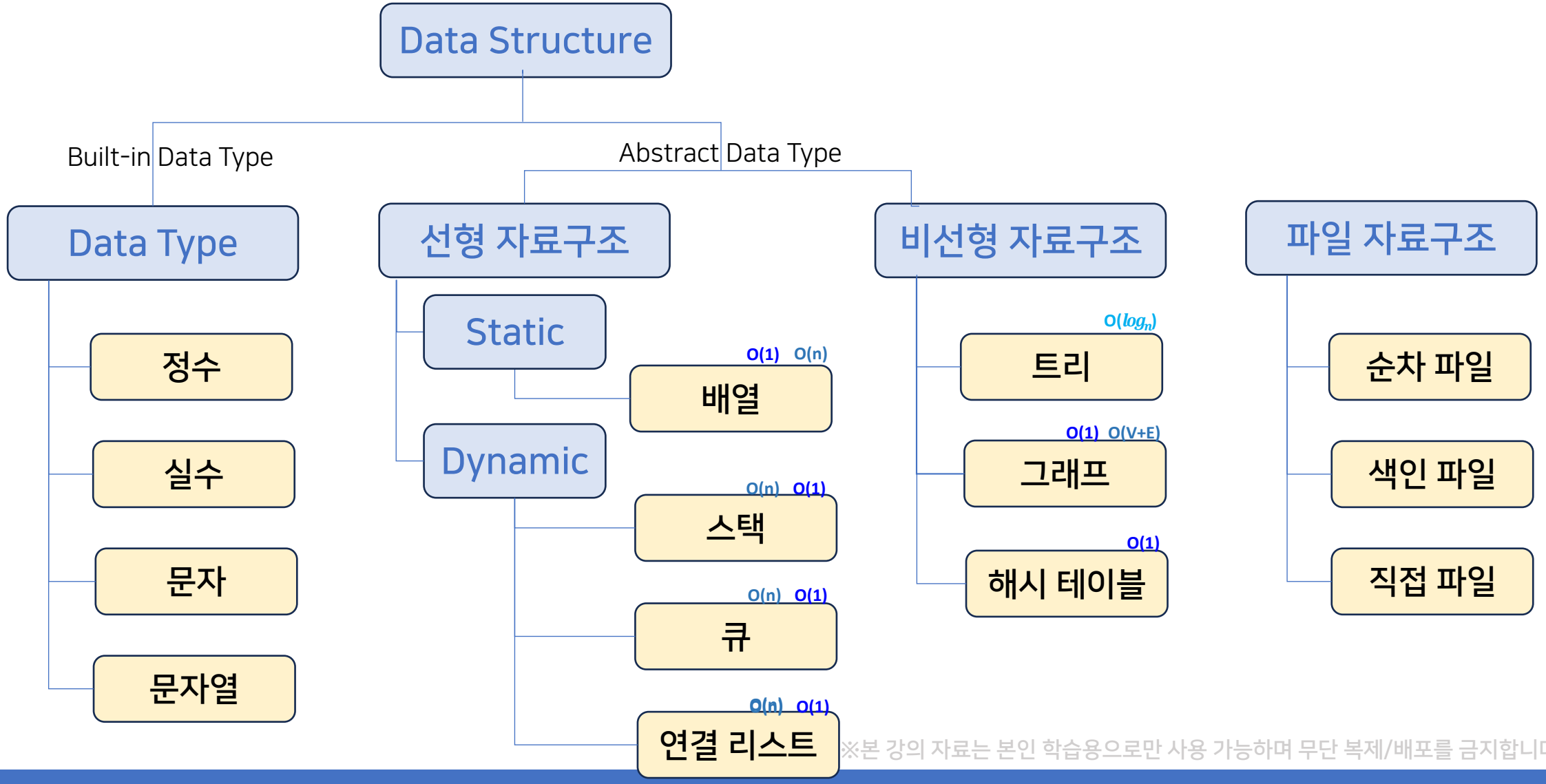
자료구조의 이해

자료구조 개요

자료구조란?

- 자료구조(Data Structure)
  - 데이터를 효율적으로 저장하고 관리하는 방식
  - 컴퓨터에서 데이터를 조직화하고 조작하는 방법을 정의하는 개념

자료구조의 종류



자료구조의 종류

- 단순 자료구조(Built-in Data Type)
  - 프로그래밍 언어의 데이터 형식에 해당하는 정수, 실수, 문자, 문자열 등
  - 정수(Integer)
  - 실수(Float)
  - 문자(Character)
  - 문자열(String)

```
정수  0, 100, 1234, -27 등
```

```
실수  0.1, 3.14, 1.234567 등
```

```
문자  'A', '채', '3' 등
```

```
문자열  "안녕", "1234", "한" 등
```

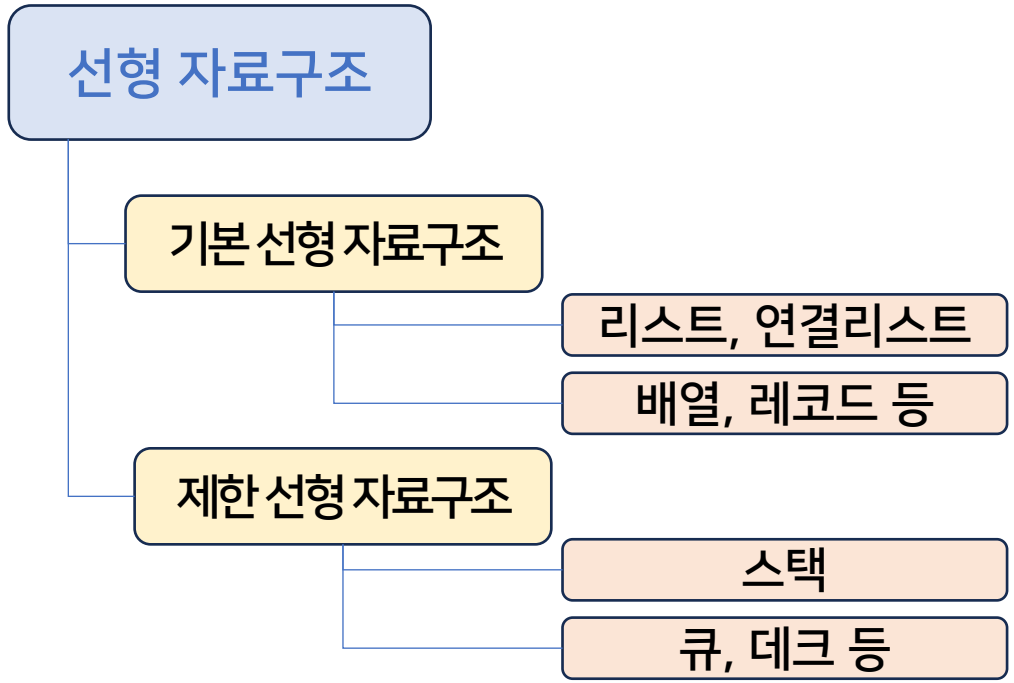
자료구조의 종류

- **추상 자료형(ADT, Abstract Data Type)**
  - **데이터**와 해당 데이터를 조작하는 **연산(Operation)**을 **추상적으로 정의한 자료형**
  - ADT는 "**무엇을 할 수 있는지**"만 정의하고, "어떻게 구현하는지"는 명시하지 않음

추상자료형	설명	주요 연산
리스트(List)	순서가 있는 요소의 집합	Insert, Delete, Search
스택(Stack)	LIFO(Last-In, First-Out) 구조	push(), pop(), peek()
큐(Queue)	FIFO(First-In, First-Out) 구조	enqueue(), dequeue()
덱(Deque)	양방향 큐 (앞뒤 삽입/삭제 가능)	append(), appendleft(), pop(), popleft()
우선순위 큐(Priority Queue)	우선순위가 높은 요소가 먼저 처리됨	insert(), extract_min()
그래프(Graph)	노드(Node)와 간선(Edge)으로 구성	add_node(), add_edge(), search()

자료구조의 종류

- 선형 자료구조(Linear Data Structure)
  - 자료들이 직선 형태로 나열되어 있는 구조
  - 전후/인접/선후 원소들 간에 1:1 관계로 나열 됨

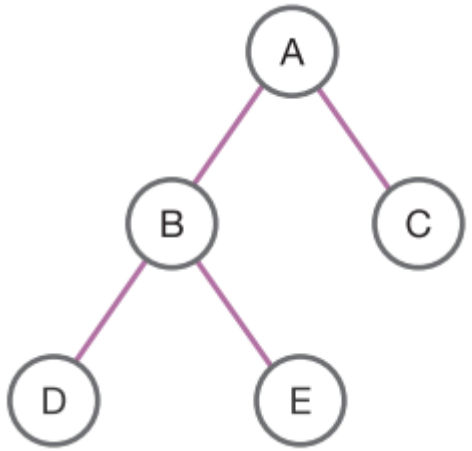


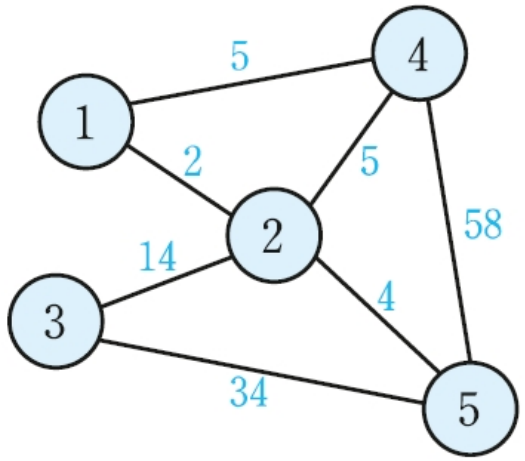
: 자료의 삽입 및 삭제가 어느 위치에서도 가능

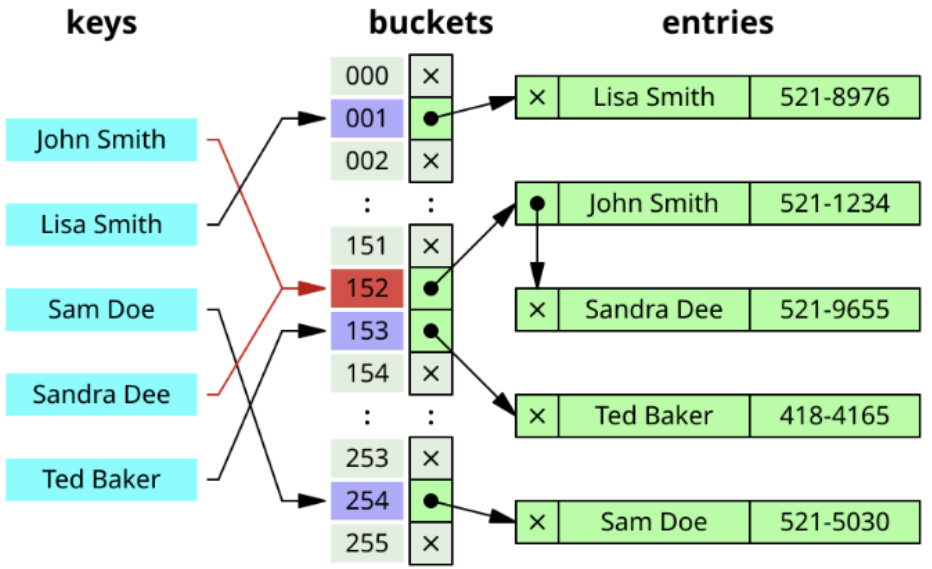
: 자료의 삽입 및 삭제가 정해진 위치에서만 가능

자료구조의 종류

- 비선형 자료구조(Non-linear Data Structure)
  - 하나의 데이터 뒤에 여러 개가 이어지는 형태.
  - ex: 트리, 그래프, 키-값 테이블 형태 등





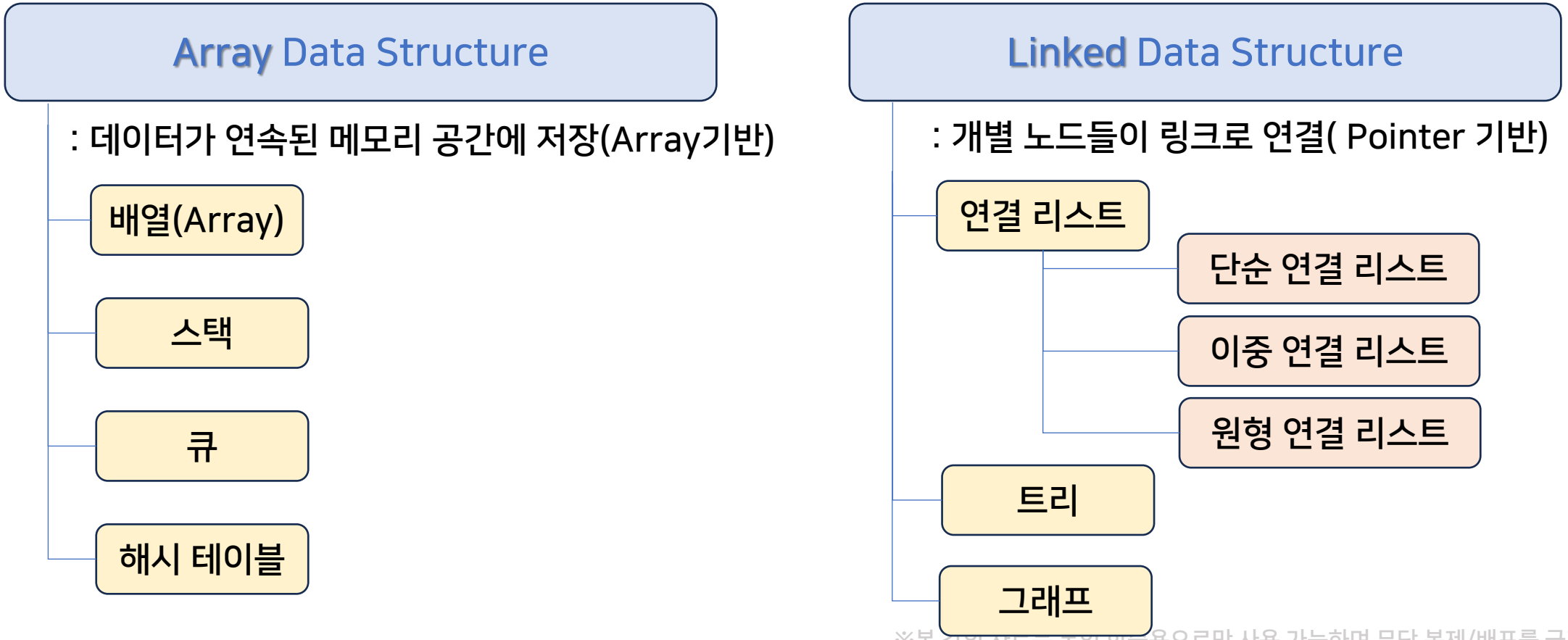


자료구조의 종류

- **파일 자료구조 예 : (디스크에 저장되는 방식에 따라)**
  - 순차 파일(Sequential File)
    - 파일 내용을 논리적인 처리 순서에 따라 연속해서 저장하는 것을 말함
    - log파일 등
  - 직접 파일 (Direct File)
    - 파일 내용을 임의의 물리적 위치에 기록하는 방식으로 직접 접근 방식(Direct Access Method)
    - DBMS 등
  - 색인 순차 파일 (Indexed Sequential File, ISAM)
    - 순차 파일과 직접 파일이 결합된 형태

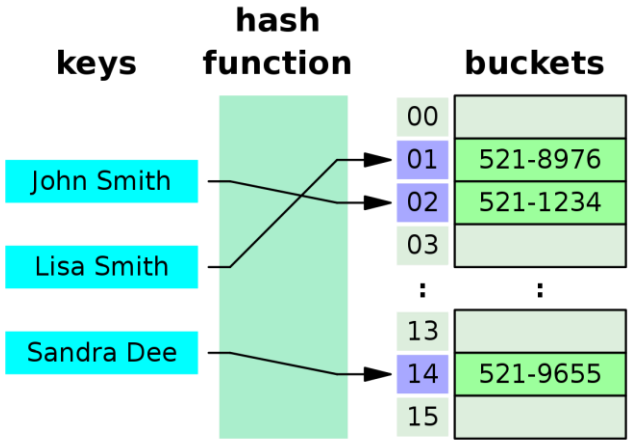
Array Data Structure vs Linked Data Structure

- 자료구조의 종류 (연결 형태에 따라 분류)



Array Data Structure vs Linked Data Structure

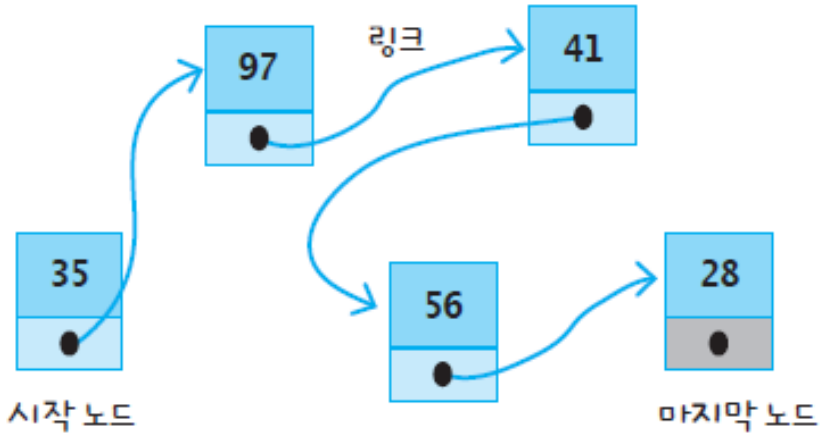
- Array Data Structure
  - 데이터를 연속된 메모리 공간에 저장하는 자료구조
  - 개별 요소들이 서로 연결되어 있지 않고, 인덱스를 통해 요소에 접근
  - e.g.: Array, Stack, Queue, Hash table 등

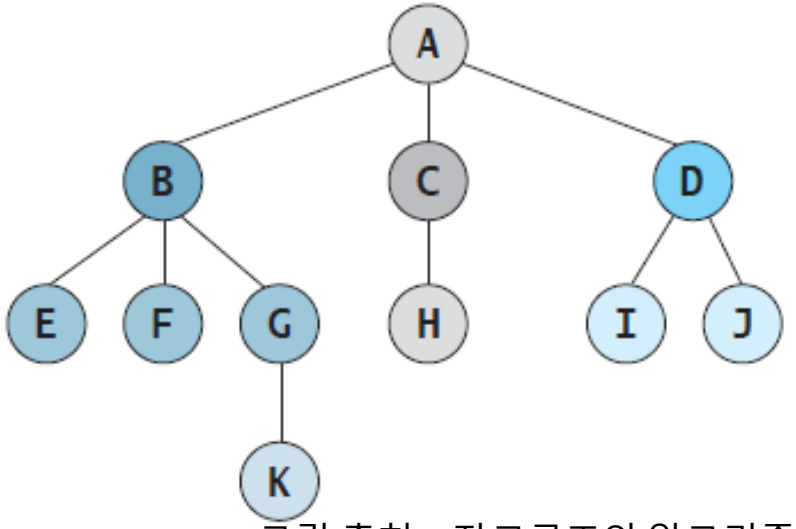


[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

Array Data Structure vs Linked Data Structure

- Linked Data Structure
  - 개별적인 노드(node)들이 서로 연결되어 있는 자료구조
  - 각 노드는 데이터와 다음 노드를 가리키는 포인터(또는 링크)를 가지고 있음
  - e.g.: Linked List, Tree 등





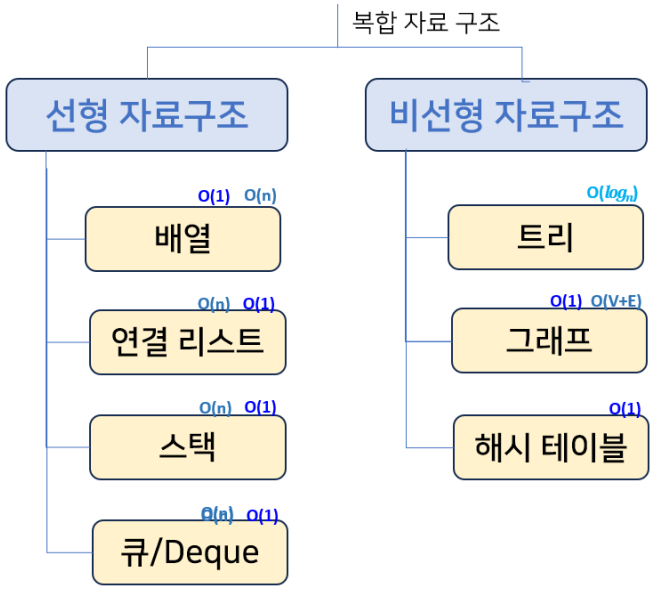
[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

Array Data Structure vs Linked Data Structure

특징	Array Data Structure	Linked Data Structure
메모리 할당	정적 할당 (연속된 메모리 블록)	동적 할당 (노드 단위)
접근 방식	인덱스를 통한 임의 접근	링크를 따라가며 순차 접근
크기 변경	어려움 (크기가 고정됨)	용이 (노드 추가/삭제)
메모리 효율성	높음 (메모리 연속성)	낮음 (포인터 오버헤드)
접근 시간	임의 접근 시 O(1)	순차 접근 시 O(n)
삽입/삭제	요소 이동이 필요함	상대적으로 쉬움
예시	배열, 행렬, 해시 테이블	연결 리스트, 트리, 그래프

Quiz

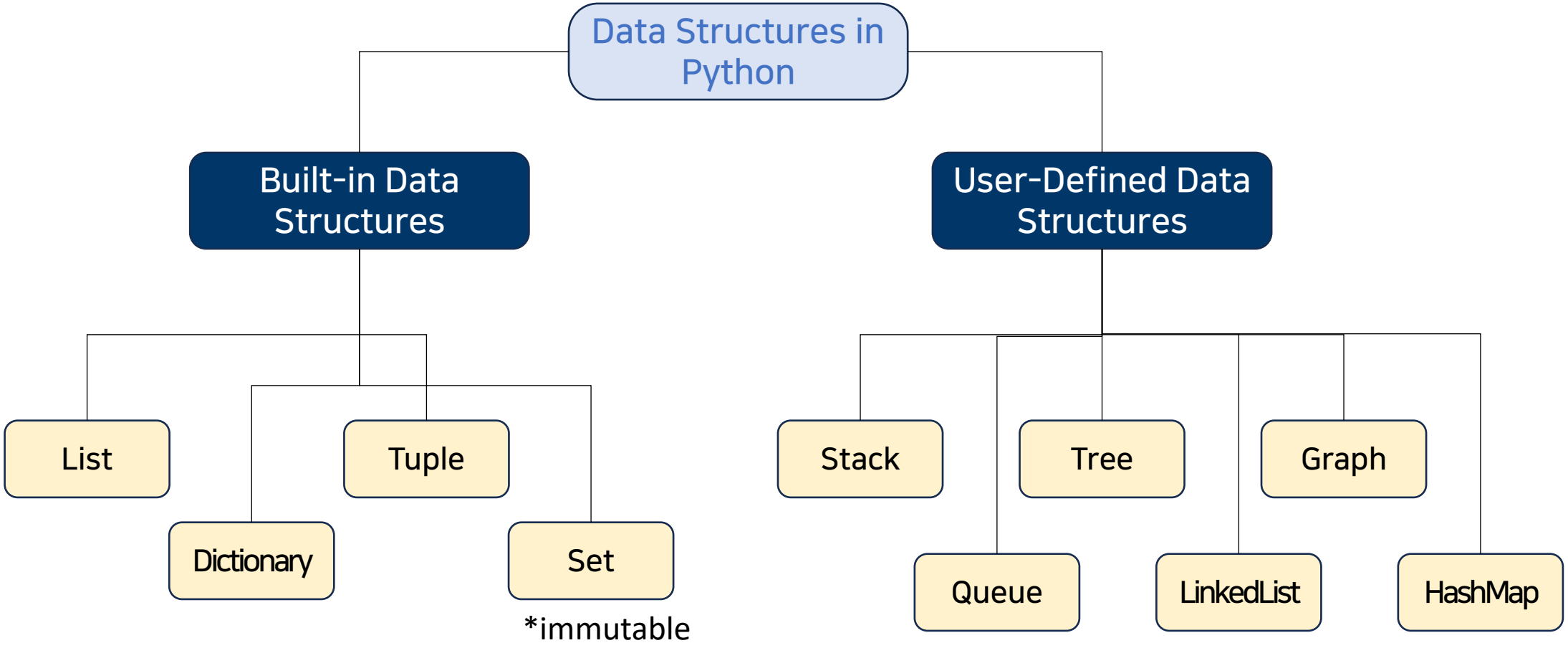
- 자료구조 선택할 때...
  - 빠른 데이터 접근이 필요하면?
  - 자주 삽입/삭제해야 한다면?
  - 후입선출(LIFO) 방식이 필요하면?
  - 선입선출(FIFO) 방식이 필요하면?
  - 빠른 탐색과 정렬이 필요하면?
  - 우선순위 기반의 작업이 필요하면?
  - 키-값 기반의 빠른 데이터 검색이 필요하면?



다양한 자료구조

파이썬 자료구조

- 파이썬 자료구조 https://docs.python.org/ko/3/tutorial/datastructures.html



자료구조의 종류

- 고급 자료구조의 예 :

자료구조	특징	주요 활용 사례
B-트리 (B-Tree) $O(\log_n)$	**균형 잡힌 다진 트리**, 여러 개 자식	데이터베이스 인덱싱, 파일 시스템
B+ 트리 (B+ Tree) $O(\log_n)$	B-트리의 변형, 모든 키가 리프 노드에 저장	데이터베이스, 파일 시스템
트라이 (Trie, Prefix Tree) $O(m)$	**문자열 검색** 최적화, 노드에 문자 저장	자동 완성, 사전 검색, IP 라우팅
세그먼트 트리 (Segment Tree) $O(\log_n)$	구간 합, 최댓값/최솟값을 빠르게 계산	구간 질의, 최적화 문제
스플레이 트리 (Splay Tree) $O(\log_n)$	최근 접근한 노드를 루트로 이동	캐싱 시스템, 메모리 할당
스킵 리스트 (Skip List) $O(\log_n)$	정렬된 연결 리스트에 여러 개의 레벨 추가	데이터베이스, 분산 시스템
쿠크-해싱 (Cuckoo Hashing) $O(1)$	충돌 없는 해시 테이블 구현	빠른 검색, 해시 테이블

자료구조의 종류

- 데이터 형태별 자료구조의 예 :

데이터 형태	대표적인 자료구조	활용 분야
숫자 (정수, 실수 등)	**배열** (Array), **연결 리스트** (Linked List)	기본적인 수치 데이터 저장, 연산 처리
	힙 (Heap)	우선순위 연산 (예: 최대/최소값 관리)
	해시 테이블 (Hash Table)	키-값 매핑을 통한 빠른 검색
	스킵 리스트 (Skip List)	동적 데이터 정렬 및 빠른 탐색
텍스트 (문자열, 문서 등)	**트라이** (Trie)	문자열 검색, 사전 데이터 관리
	접미사 트리 (Suffix Tree)	문자열 패턴 검색, NLP, DNA서열분석
	해시 테이블 (Hash Table)	중복 단어 검색, 문서 색인화
	B-트리, B+트리	대량의 문서 데이터 인덱싱

자료구조의 종류

데이터 형태	대표적인 자료구조	활용 분야
이미지 (사진, 영상 등)	**행렬** (Matrix)	픽셀 단위 데이터 저장
	KD-트리 (KD-Tree)	이미지 검색, 컴퓨터 비전
	쿼드트리 (Quadtree)	이미지 압축, GIS(지리정보시스템)
	텐서 (Tensor)	딥러닝 기반 이미지 처리 (CNN)
	히스토그램 (Histogram)	색상 분석, 밝기 조정
소리 (음성, 음악 등)	**시간-주파수 행렬** (Spectrogram Matrix)	음향 데이터 분석, 주파수 변환
	웨이블릿 트리 (Wavelet Tree)	오디오 압축, 신호 처리
	해시 테이블 (Fingerprint Hash)	오디오 매칭, 음성 검색 (Shazam)
	R-트리 (R-Tree)	음악 특징 벡터 검색

자료구조의 종류

데이터 형태	대표적인 자료구조	활용 분야
비정형 데이터 (웹, 로그, 센서 데이터 등)	LSM-Tree (Log-Structured Merge Tree)	대량의 로그 데이터 처리
	블룸 필터 (Bloom Filter)	빠른 데이터 존재 여부 확인
	그래프 (Graph)	소셜 네트워크, 추천 시스템
	시계열 데이터 구조 (Time-Series Structures)	센서 데이터, 금융 데이터

시대별 자료구조 방법의 발전

▪ 자료구조 관리 방법의 발전

초기 단계 (1940년대~1950년대)	파일 시스템 활용 (1960년대~1970년대)	데이터베이스관리시스템 (1970년대~1980년대)	객체지향프로그래밍 (1980년대~1990년대)	분산시스템및빅데이터 (1990년대~현재)
• 자료구조는 **컴퓨터 메모리에 직접 저장** • **고정 길이** 배열, 연결 리스트 등 단순 구조 사용 • 구현 방식이 제한적이고 메모리 관리가 어려웠음	• **외부 보조기억**장치인 디스크를 활용하여 **파일 형태로 자료구조** 구현 • 직접 접근 파일, 순차 접근 파일 등을 활용 • 대용량 데이터 처리가 가능해지고 영구 저장 가능	• **관계형 데이터베이스** 모델이 등장 • **테이블 형태**로 데이터 저장 및 조작 • 데이터 중복 최소화, 데이터 무결성, 보안성 향상	• **클래스와 객체 개념**을 활용한 자료구조 구현 • 데이터와 연산을 하나의 단위로 **캡슐화** • 재사용성, 확장성, 유지보수성 향상	• 분산 파일 시스템, NoSQL, 하둡 등장 • **대규모 데이터** 집합 저장 및 처리 • **클라우드, 분산 컴퓨팅** 환경에 적합한 자료구조 • **AI/ML을 위한 자료구조**

AI/ML에서 자주 사용하는 자료구조 예:

- AI/ML에서 자주 사용하는 자료구조 예 :

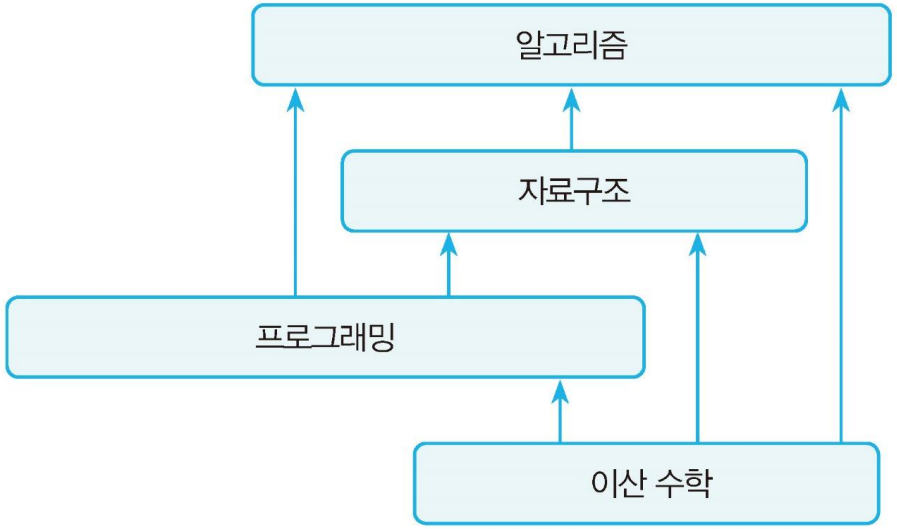
자료구조	AI/ML 활용분야	관련 라이브러리
배열 (Array)	딥러닝 가중치, 이미지 픽셀	NumPy, TensorFlow, PyTorch
해시 테이블 (Hash Table)	단어 임베딩, 추천 시스템	Python dict, HashMap (Java, C++)
스택 & 큐 (Stack & Queue)	탐색 알고리즘 (DFS, BFS)	collections.deque, Queue (Java, C++)
트리 (Tree)	의사결정 트리, XGBoost	Scikit-learn, XGBoost, LightGBM
그래프 (Graph)	소셜 네트워크 분석, 추천 시스템	NetworkX, DGL
힙 (Heap)	최적화 알고리즘, A* 탐색	heapq (Python), priority_queue (C++)
트라이 (Trie)	자동완성, NLP	collections.defaultdict, DAWG
KD-트리 (KD-Tree)	다차원 데이터의 검색, 최근접 이웃 검색 (NN)	SciPy KDTree, Scikit-learn NearestNeighbors

자료구조와 알고리즘

자료구조와 알고리즘

- 자료구조
  - 데이터를 효율적으로 저장하고 관리하는 방식
  - 컴퓨터에서 데이터를 조직화하고 조작하는 **방법**을 정의하는 개념

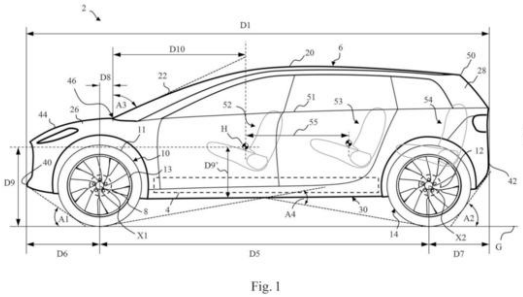
- 알고리즘은 자료구조의 확장이다



자료구조와 알고리즘

- 자료구조와 알고리즘, 프로그램의 관계

자동차 부품
(자료구조)



자동차 조립 방법
(알고리즘)

자동차 조립
(프로그래밍 언어)

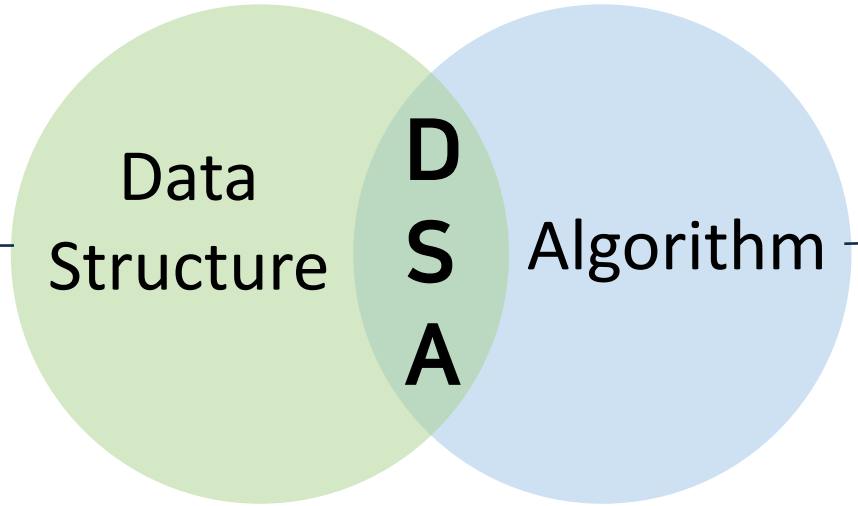
완성 자동차
(소프트웨어)

Input Data

Processing Method

자료구조와 알고리즘

- Linked List
- Matrix/Grid
- Queue
- Stack
- Array
- Hash
- Heap
- Graph
- String
- Tree



- Pattern Searching
- Divide & Conquer
- Searching
- Sorting
- Bitwise
- Greedy
- Recursion
- Backtracking
- Mathematical
- Dynamic Programming

※ 가천대학교 컴퓨터공학과 '알고리즘' 과정

기초 자료구조

배열(Array)

배열(Array)

- 배열(Array)란?
  - 동일한 타입의 요소들이 연속적인 메모리 공간에 저장된 자료구조
  - 인덱스(Index)를 사용하여 요소에 빠르게 접근
  - 데이터 검색과 순차적인 데이터 처리를 위해 최적화된 자료구조

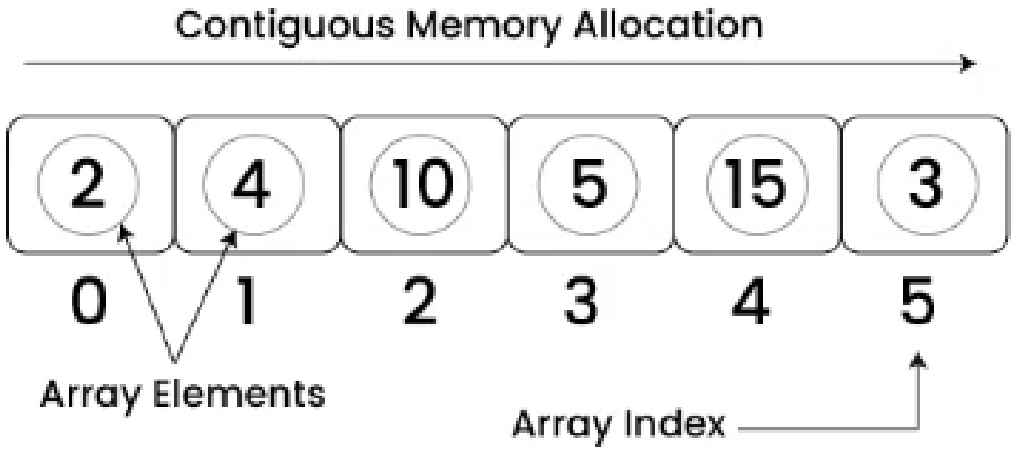


image by geeksforgeeks.org

배열(Array) 특징

- **고정된 크기 (Static Size)**
  - 배열은 일반적으로 선언 시 크기가 고정됨 (동적 배열은 예외)
- **연속적인 메모리 할당 (Contiguous Memory Allocation)**
  - 모든 요소가 연속된 메모리 주소에 저장됨 → 빠른 접근 가능 ($O(1)$)
- **인덱스를 이용한 빠른 접근 (Index-Based Access)**
  - 특정 요소에 $O(1)$ 의 시간 복잡도로 접근 가능.
- **데이터 추가/삭제가 비효율적**
  - 배열 크기가 고정되어 있으며, 요소 삽입/삭제 시 비효율적($O(n)$)

배열의 주요 연산 및 시간 복잡도

연산	설명	시간 복잡도
접근 (Access)	특정 인덱스의 요소를 가져오기	$O(1)$
검색 (Search)	특정 요소 찾기 (선형 검색)	$O(n)$
삽입 (Insert)	특정 위치에 요소 삽입	$O(n)$
삭제 (Delete)	특정 위치 요소 삭제	$O(n)$

배열의 종류

- **1차원 배열(One-Dimensional Array)** [10, 20, 30, 40, 50]
  - 가장 기본적인 형태의 배열
- **2차원 배열 (Two-Dimensional Array** [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
  - 행(row)과 열(column)로 구성된 배열 (행렬)
- **다차원 배열 (Multi-Dimensional Array)**
  - 3차원 이상의 배열, 이미지 처리, 그래픽 연산, 머신러닝에서 활용
- **동적 배열 (Dynamic Array)**
  - 가장 크기가 가변적으로 조정됨(Python list, C++ vector)
  - 요소 추가 시 자동으로 메모리 재할당 및 확장

실습 : 배열 구조 출력하기

- **파이썬으로 배열 구조 출력하기**

  임의의 정수 12개를 발생시켜

  1) 1차원 배열 구조로 출력하기
  2) 2차원(3x4) 배열 구조로 출력하기
  3) 3차원(2x2x3) 배열 구조로 출력하기

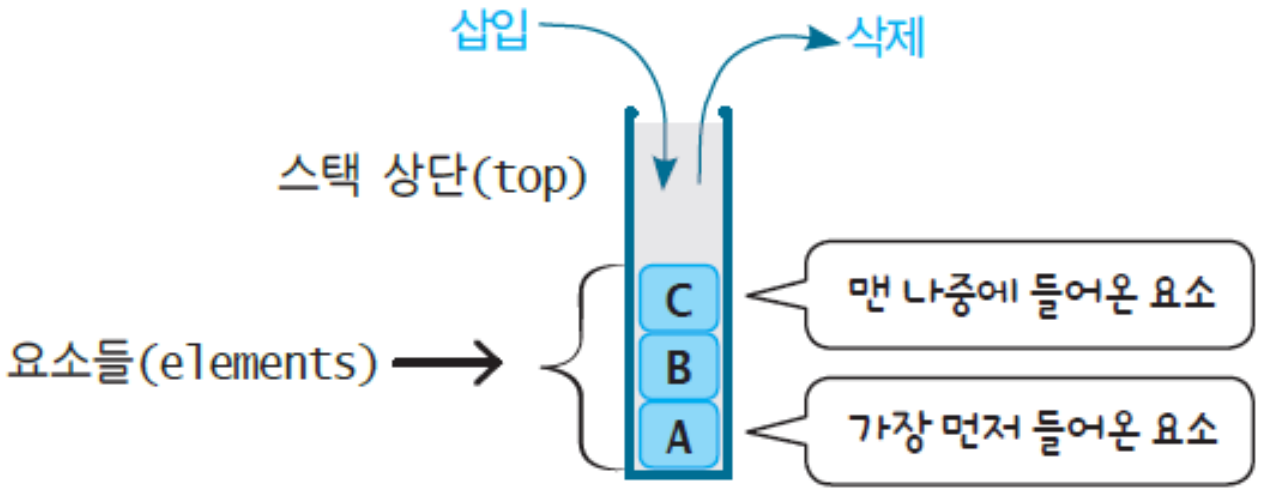
스택(Stack)

스택(Stack)

- 스택(Stack)이란?
  - “stack”: 쌓아놓은 더미
  - 후입선출(LIFO:Last-In First-Out)/선입후출(FILO)의 자료구조
  - 가장 최근에 들어온 데이터가 가장 먼저 나감

스택의 구조

- 스택의 구조
  - 다른 통로들은 모두 막고 한쪽만을 열어둔 구조
  - 이때 열린 곳을 보통 스택 상단(stack top)
  - 스택은 요소의 삽입과 삭제가 상단에서만 가능한 자료구조



[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

스택의 응용

- 스택의 응용 예 :
  - **웹 브라우저의 방문기록 기능**

    웹브라우저에서 뒤로가기, 앞으로 가기 기능

  - **실행취소(Undo)**

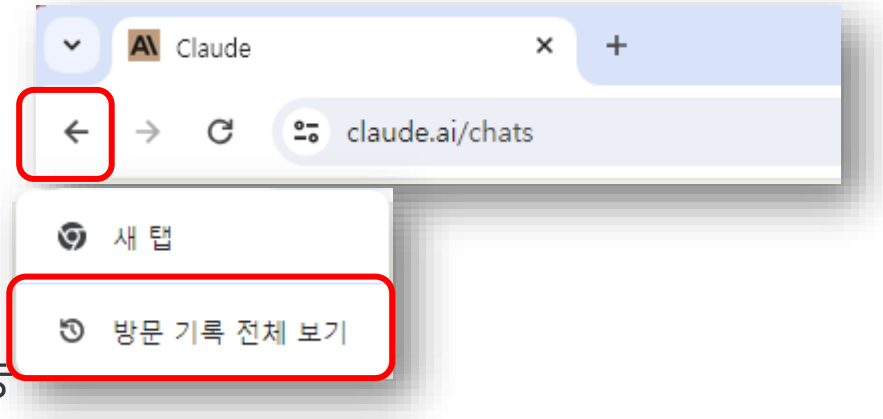
    ‘문서 편집기나 그래픽 편집 프로그램에서 실행 취소 기능

  - **수식 계산기**

    수식을 계산할 때 스택 활용,

    연산자가 나올 때마다 스택에 저장된 피연산자들을 꺼내 연산 수행하고 결과를 다시 스택에 저장

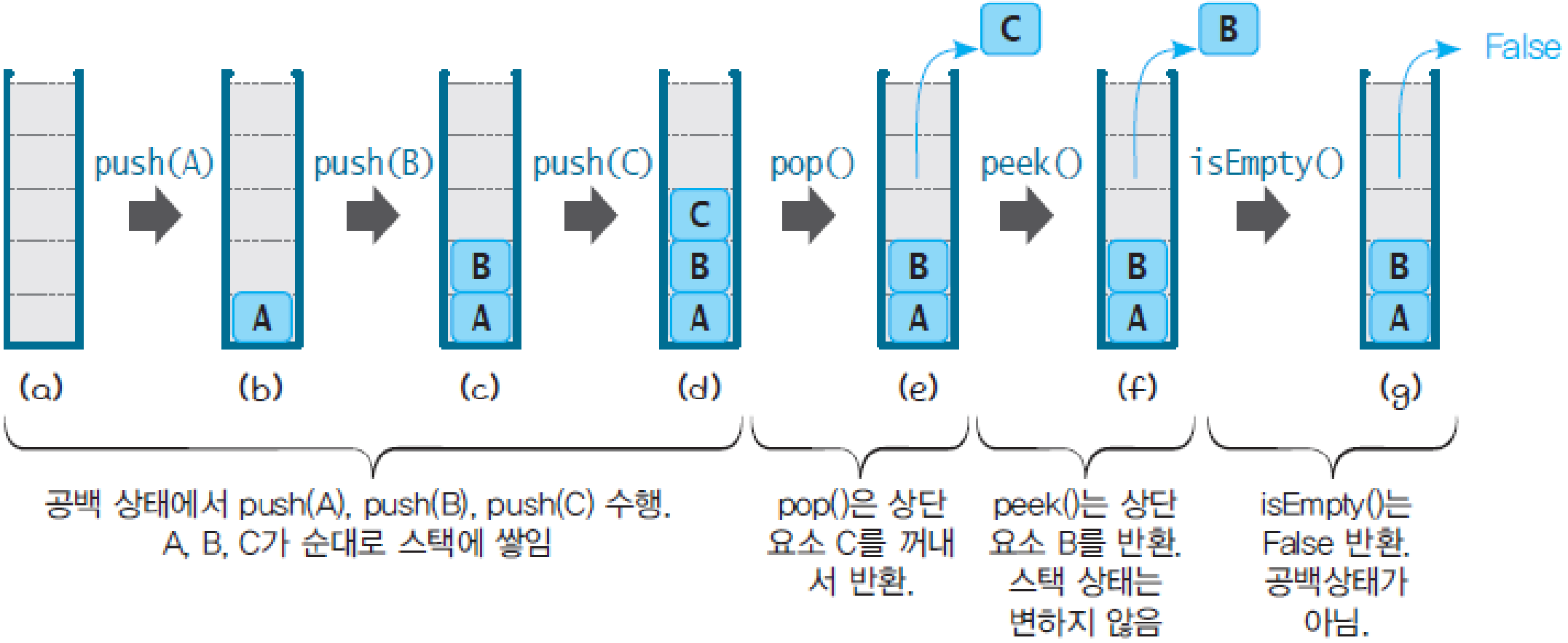
  - **괄호 검사**



스택의 연산(Operations)

- 스택의 연산
  - push(e): 스택의 맨 위에 데이터(요소) 삽입
  - pop(): 스택의 맨 위에 있는 요소 제거
  - peek/top(): 스택의 맨 위에 있는 항목을 삭제하지 않고 반환, 데이터 참조
  - isEmpty(): 스택이 비어 있는지 확인, True/False 반환
  - isFull(): 스택이 가득 차 있는지 확인, True/False 반환
  - size(): 스택에 들어 있는 전체 요소의 수 반환

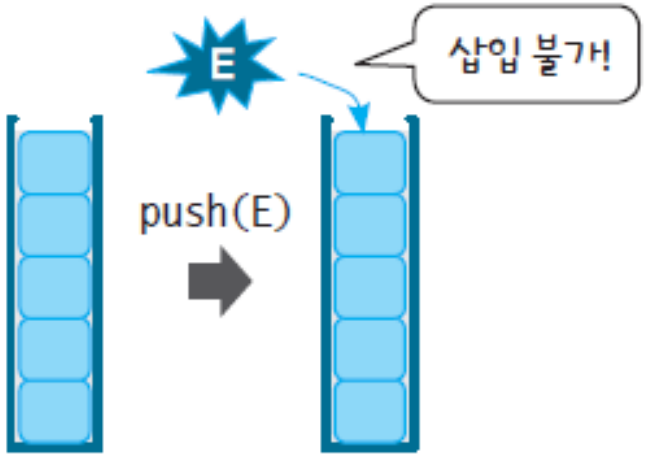
스택의 연산 동작



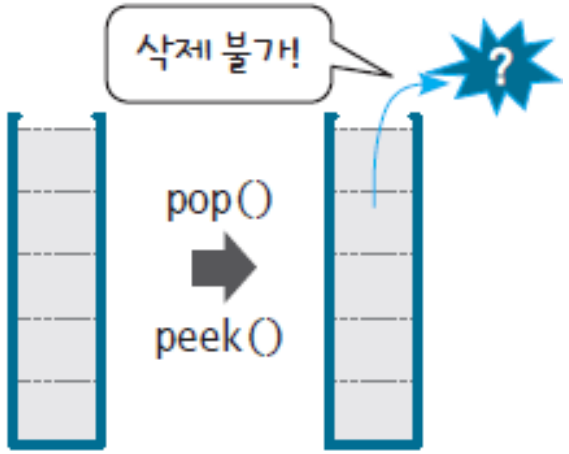
[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

스택의 오류 상황

- overflow
  - 포화 상태인 스택에 새로운 요소를 삽입(push)할 때 발생
- underflow
  - 공백 상태의 스택에서 pop( )이나 peek( )을 호출하면 발생



(a) 오버플로 오류

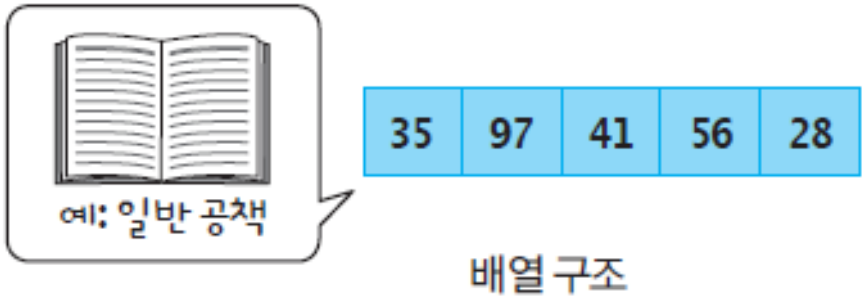


(b) 언더플로 오류

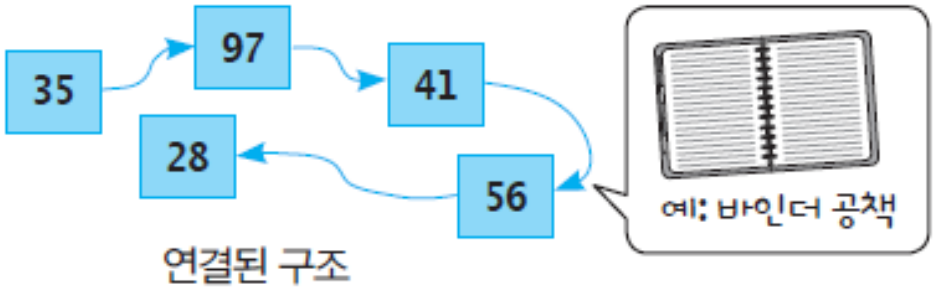
[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

스택의 구현 방법

- 배열을 이용한 구현
  - 자료를 배열에 모아 저장
  - (예) 일반 공책: 편리하지만 공책이 가득 차면 저장 불가 구현



- 연결 리스트를 이용한 구현
  - 요소들을 연결하여 하나로 관리
  - (예) 바인더 공책: 페이지의 위치를 바꾸거나 페이지를 쉽게 추가, 삭제 가능.



쉽게 추가/삭제 가능한 만큼 관리하기 복잡

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

배열로 스택 구현하기

- 전역변수 데이터
  - 배열 구조 → 용량이 고정된 스택
  - 변수 사용 → 전역 변수

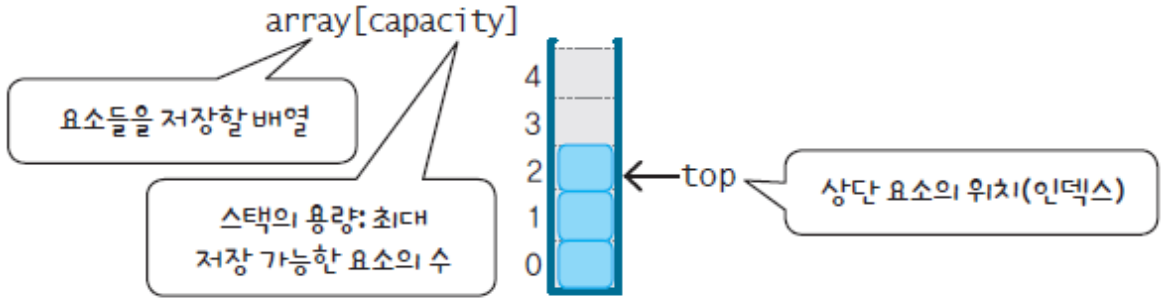


그림 1.7 | 배열을 이용한 스택의 구조

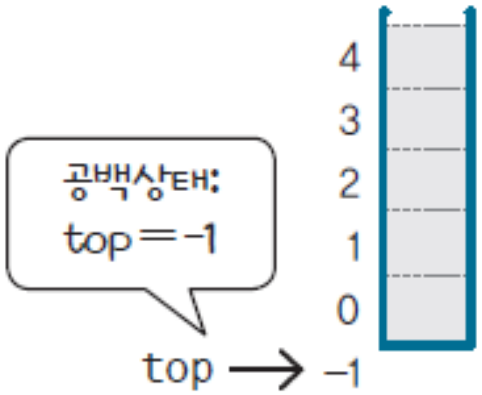
- array[] : 스택 요소들을 저장할 배열
- capacity : 스택의 최대 용량. 저장할 수 있는 요소의 최대 개수(상수)
- top : 상단 요소의 위치(변수, 인덱스)

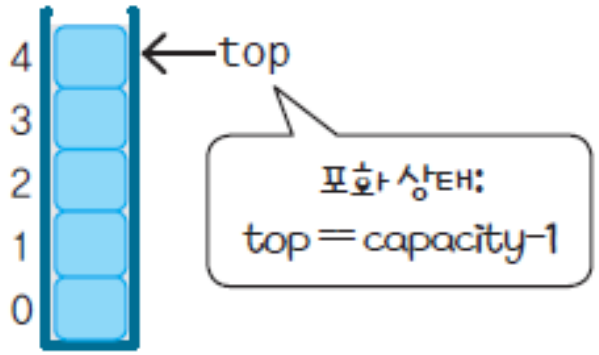
```
capacity = 10                 # 스택 용량: 10으로 지정
array = [None]*capacity       # 요소 배열: [None,..,None] 길이(capacity)
top = -1                      # 스택 상단의 인덱스 : 공백 상태(-1)로 초기화
```

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

배열로 스택 구현하기

- 스택 연산
  - 공백 상태 : isEmpty()
  - 포화 상태 : isFull()





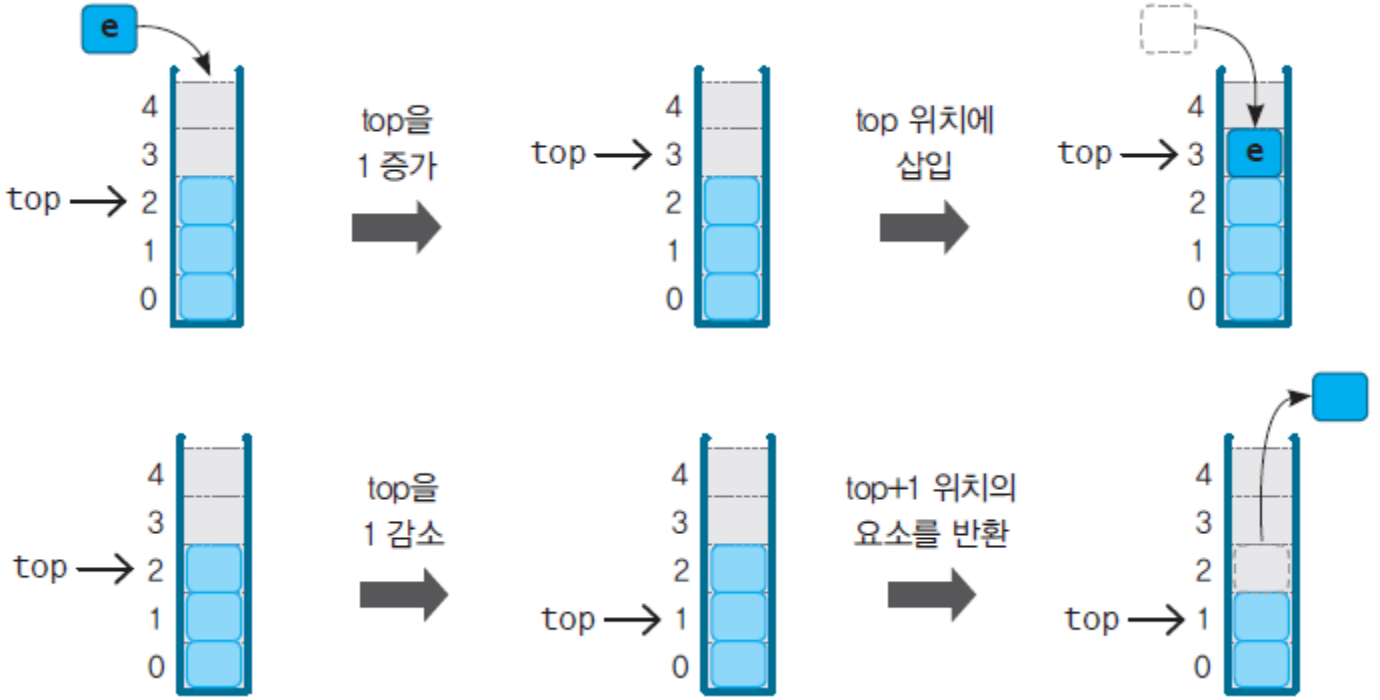

```
def isEmpty():
    if top == -1: return True  # 공백이면 True
    else: return False         # 아니면 False
#     return top == -1
```

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

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

배열로 스택 구현하기

- 스택 연산
  - 추가 상태 : push(e)
  - 삭제 상태 : pop()



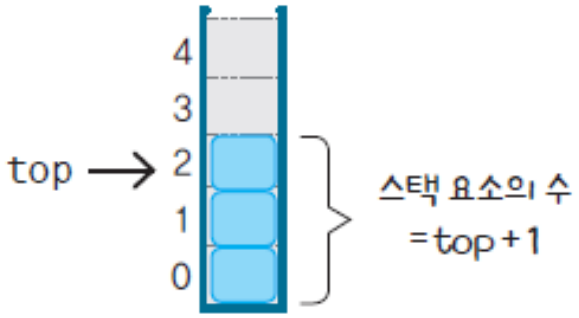

```
def push(e):
    #global top
    if not isFull():
        top += 1
        array[top] = e
    else:
        print('stack overflow!')
        exit()
```

```
def pop():
    #global top
    if not isEmpty():
        top -= 1
        return array[top+1]
    else:
        print('stack underflow!')
        exit()
```

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

배열로 스택 구현하기

- 스택 연산
  - 상단 요소 참조 : peek()
  - 현재 스택 크기 : size()




```
def peek():
    if not isEmpty():
        return array[top]
    else:
        pass
```

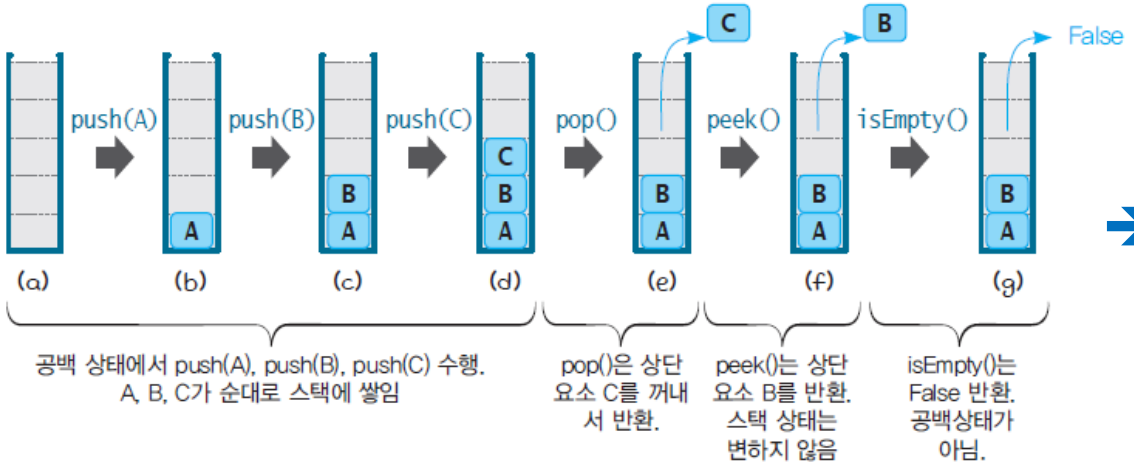
```
def size():
    return top+1
```

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

실습 : 스택 클래스 구현하기

- 스택 클래스 구현하기

1) ArrayStack 이란 이름의 빈 스택 클래스 작성하기 (사이즈 제한 없는)
2) Stack에 push(), pop(), peek(), isEmpty(), size() 메서드 추가하기
3) 앞에서 기술된 스택 연산 실행하고 아래와 같은 결과 출력하기



➔ 출력결과

```
(a) []
(b) ['A']
(c) ['A', 'B']
(d) ['A', 'B', 'C']
(e) C
(f) B
(g) False
```

4) ArrayStack 에 사이즈 제한 있도록 isFull() 매서드 추가하기

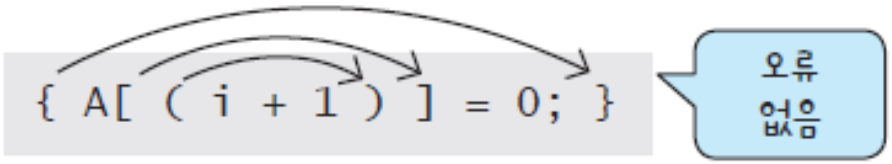
스택의 응용

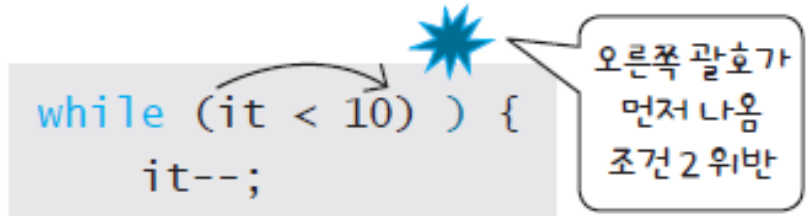
- ## 응용 문제 : 괄호 검사
  - 소스코드나 주어진 문자열에서 괄호들이 올바르게 사용되었는지를 검사하는 문제

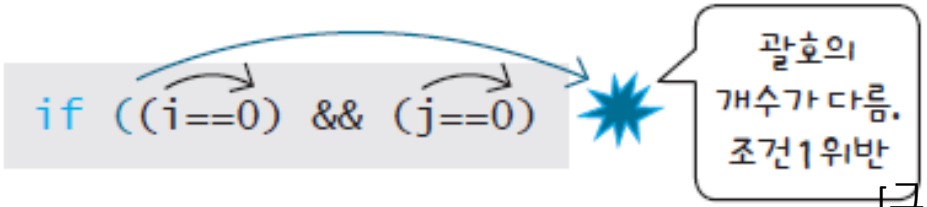
올바른 괄호 사용을 위한 조건

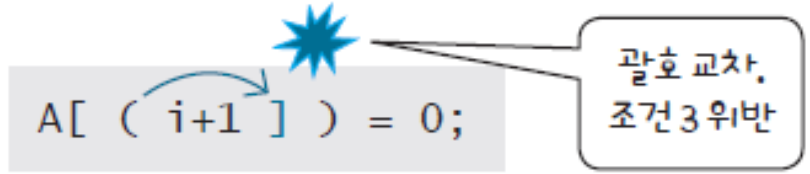
- 조건 1 : 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야 합니다.
- 조건 2 : 같은 종류인 경우 왼쪽 괄호가 오른쪽보다 먼저 나와야 합니다.
- 조건 3 : 다른 종류의 괄호 쌍이 서로 교차하면 안 됩니다.

괄호가 아닌 문자들은 모두 무시하고 처리함







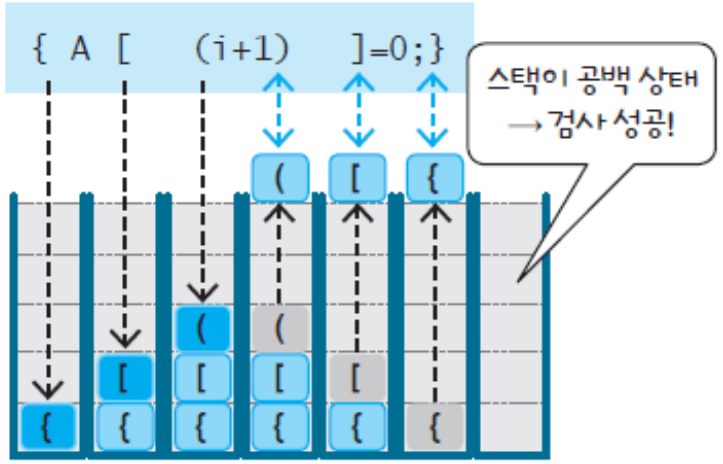


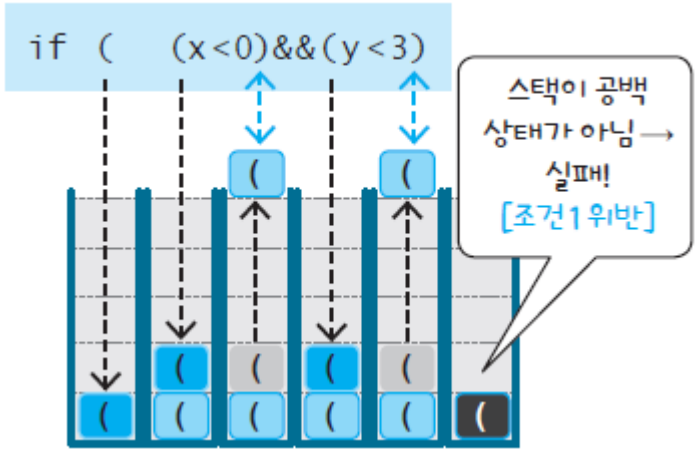
[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

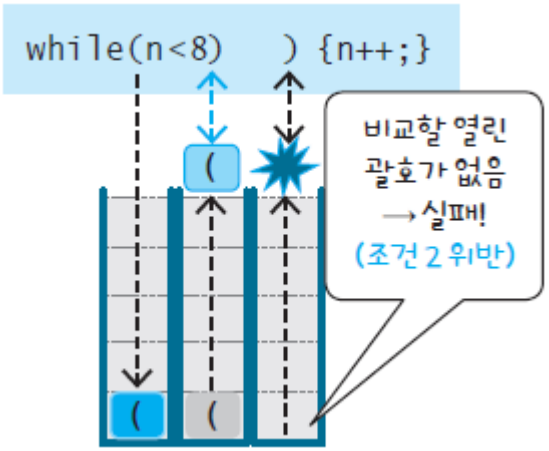
괄호 검사 알고리즘

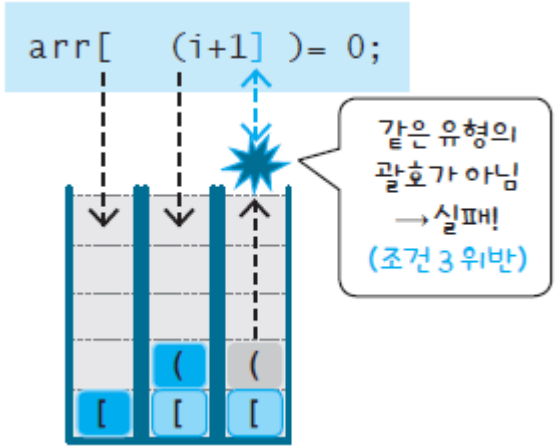
- 빈 스택을 준비합니다.
- 입력된 문자를 하나씩 읽어 왼쪽 괄호를 만나면 스택에 삽입합니다.
- 오른쪽 괄호를 만나면 가장 최근에 삽입된 괄호를 스택에서 꺼냅니다. 이때 스택이 비었다면 오른쪽 괄호가 먼저 나온 상황이므로 조건 2에 위배됩니다.
- 꺼낸 괄호가 오른쪽 괄호와 짝이 맞지 않으면 조건 3에 위배됩니다.
- 입력 문자열을 끝까지 처리했는데 스택에 괄호가 남아 있으면 괄호의 개수가 같지 않으므로 조건 1에 위배됩니다. 모든 문자를 처리하고 스택이 공백 상태이면 검사 성공입니다.

괄호 검사 알고리즘









[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

실습 : 괄호 검사 알고리즘 구현하기

- 괄호 검사 알고리즘을 구현하고 테스트하기

1) 괄호 검사 알고리즘 만들기

```
def checkBrackets(statement):
```

2) 괄호 검사 알고리즘 테스트하기

```
str1 = "{ A[(i+1)]=0; }"
str2 = "if ((x<0) && (y<3)"
str3 = "while (n<8)) {n++;}"
str4 = "arr[(i+1])=0;"

print(str1, " ---> ", checkBrackets(str1))
print(str2, " ---> ", checkBrackets(str2))
print(str3, " ---> ", checkBrackets(str3))
print(str4, " ---> ", checkBrackets(str4))
```

파이썬에서 스택 사용하기

- 파이썬에서 스택 사용하는 방법
  - 방법 1) Stack 클래스 직접 구현해서 사용하기
  - 방법 2) 파이썬 리스트 함수 사용해서 스택으로 사용하기
  - 방법 3) 파이썬의 queue 모듈 LifoQueue 사용하기

파이썬에서 스택 사용하기

- 방법 2) 파이썬 리스트 함수 사용해서 스택으로 사용하기

연산	ArrayStack	파이썬 list	사용 예
삽입	push( )	메서드 append( ) 사용	L.append(e)
삭제	pop( )	메서드 pop( ) 사용	L.pop( )
요소의 수	size( )	내장 함수 len( )을 사용	len(L)
공백 상태 검사	isEmpty( )	요소의 수가 0인지 검사	len(L)==0
포화 상태 검사	isFull( )	list는 용량이 무한대라 포화 상태는 의미가 없음. 항상 False	False
상단 들여다보기	peek( )	맨 마지막 요소 참조	L[len(L)-1] 또는 L[-1]

파이썬에서 스택 사용하기

- 방법 3) 파이썬의 queue 모듈 LifoQueue 사용하기

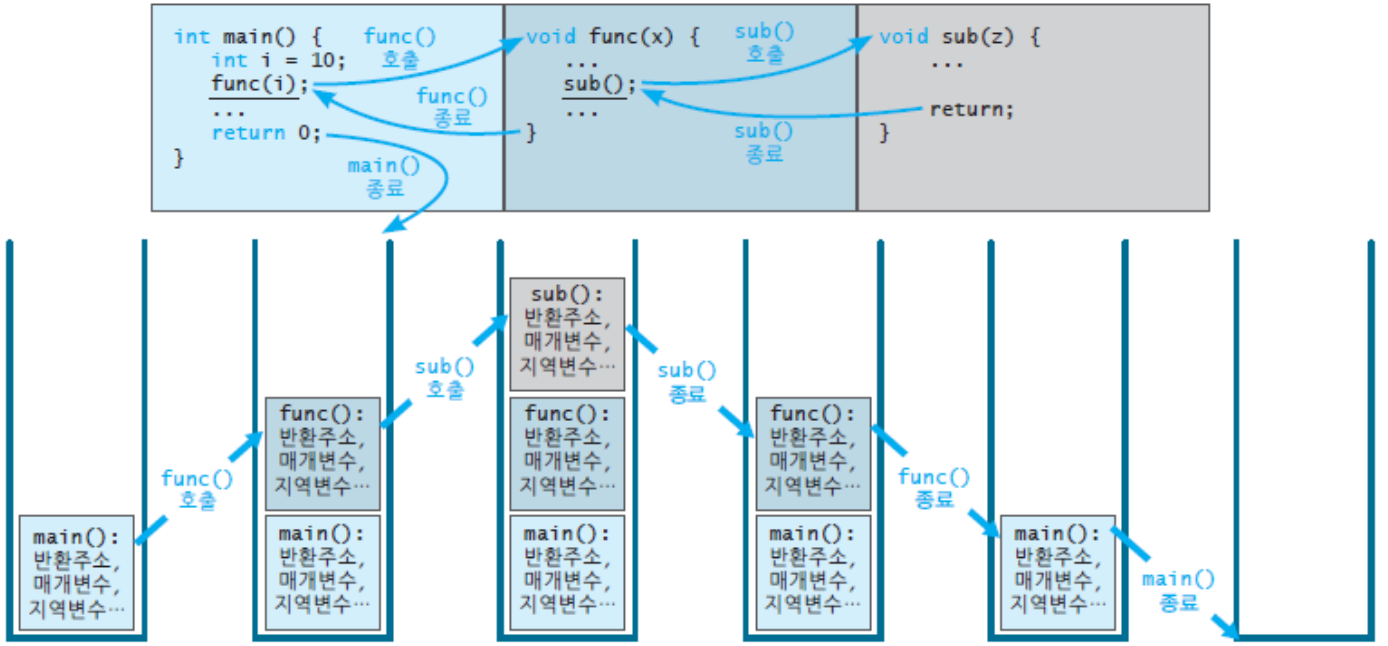
```
import queue
s = queue.LifoQueue(maxsize=100)
```

연산	ArrayStack	queue.LifoQueue
삽입/삭제	push(), pop()	put(), get()
공백/포화 상태 검사	isEmpty(), isFull()	empty(), full()
상단 들여다보기	peek()	제공하지 않음

시스템 스택과 순환 호출

- 시스템 스택
  - 운영체제가 관리하는 메모리에는 스택 영역이 있는데 함수의 호출과 반환을 위해 사용됨

    → 시스템 스택



[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

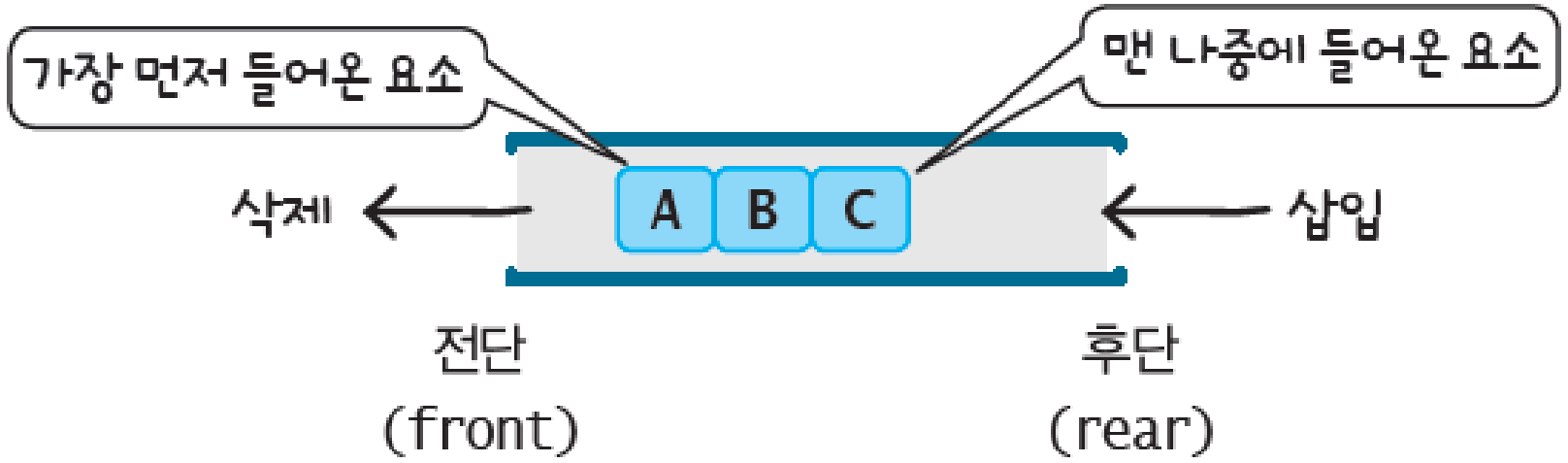
큐(Queue)

큐(Queue)

- 큐(Queue)란?
  - 선입선출(FIFO:First-In First-Out)의 자료구조
  - 가장 먼저 들어온 데이터가 가장 먼저 나감

큐의 구조

- 큐의 구조
  - 양쪽을 열어둔 구조
  - 큐의 한쪽 끝이 뒤(rear)가 되고 다른 한쪽이 앞(front)이 됨
  - 앞에서부터 제거하는 연산(dequeue), 뒤에서부터 추가하는 연산(enqueue)



[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

큐의 응용

큐의 응용 예

- **버퍼(buffer)로 사용**

  컴퓨터에서 시간이나 속도 차이를 극복하기 위한 임시 기억 장치

- **운영 체제의 작업 스케줄링**

  운영 체제의 준비 상태 프로세스들은 실행을 위해 큐에 대기하며, CPU 스케줄러는 이 큐를 사용해 어떤 프로세스를 다음에 실행할지 결정함

- **인쇄 대기열 관리**

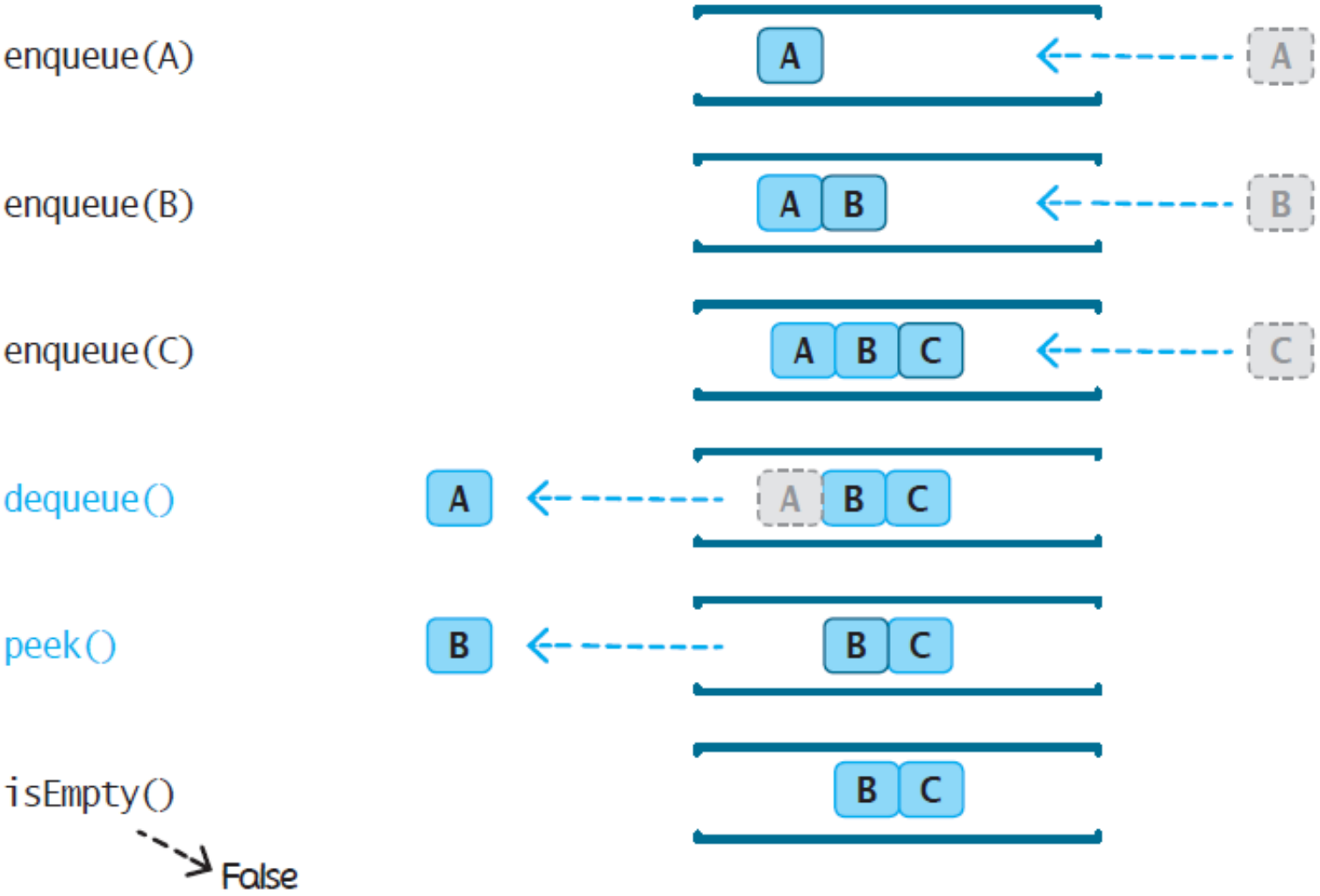
  네트워크 프린터 같은 인쇄 장치에서는 여러 사용자의 인쇄 요청을 순서대로 처리하기 위해 큐를 사용

- **웹 서버의 요청 처리**
- **실시간 시스템에서의 이벤트 관리**

큐의 연산(Operations)

- 큐의 연산
  - enqueue(e): 새로운 요소 e를 큐의 맨 뒤에 추가
  - dequeue(): 큐의 맨 앞에 있는 요소를 꺼내서 반환
  - peek(): 큐의 맨 위에 있는 요소를 삭제하지 않고 반환
  - isEmpty(): 큐가 비어 있는지 확인, True/False 반환
  - isFull(): 큐가 가득 차 있는지 확인, True/False 반환
  - size(): 큐에 들어 있는 전체 요소의 수 반환

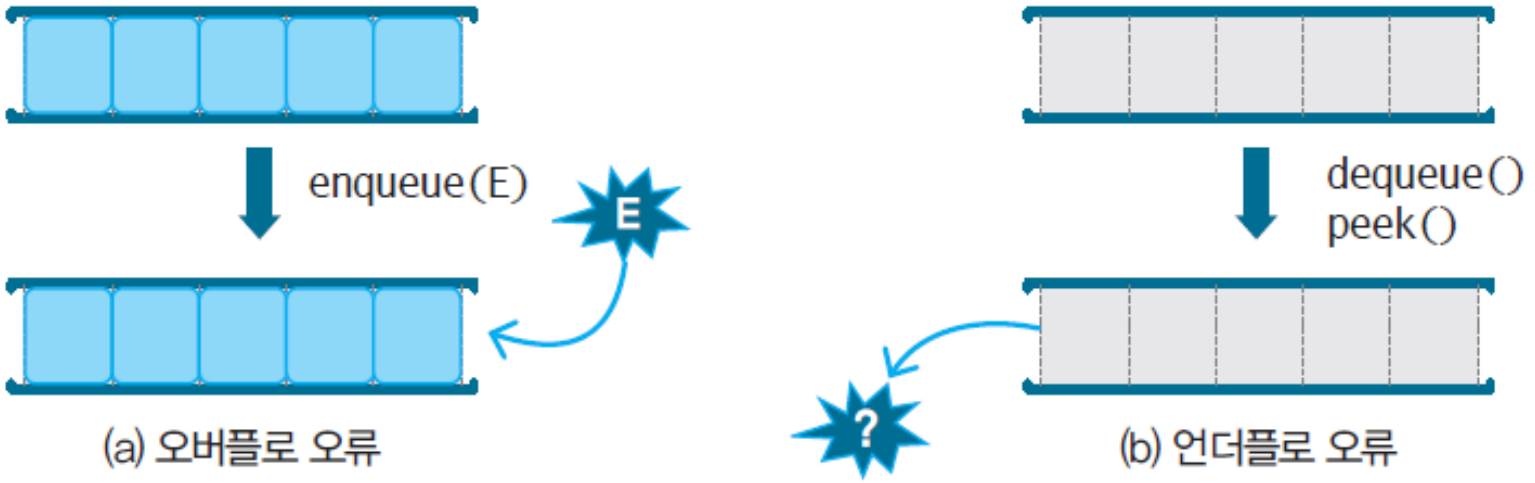
큐의 연산 동작



[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

큐의 오류 상황

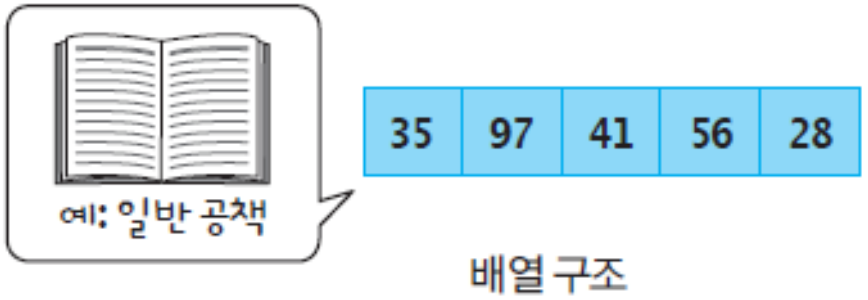
- overflow
  - 포화 상태인 큐에 enqueue( ) 연산을 실행하는 경우 발생
- underflow
  - 공백 상태의 큐에서 dequeue( )이나 peek( ) 연산을 실행하는 경우 발생



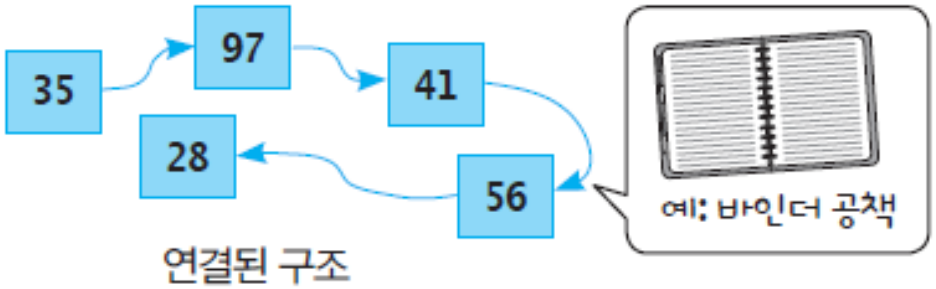
[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

큐의 구현 방법

- 배열을 이용한 구현
  - 자료를 배열에 모아 저장
  - (예) 일반 공책: 편리하지만 공책이 가득 차면 저장 불가 구현



- 연결 리스트를 이용한 구현
  - 요소들을 연결하여 하나로 관리
  - (예) 바인더 공책: 페이지의 위치를 바꾸거나 페이지를 쉽게 추가, 삭제 가능.

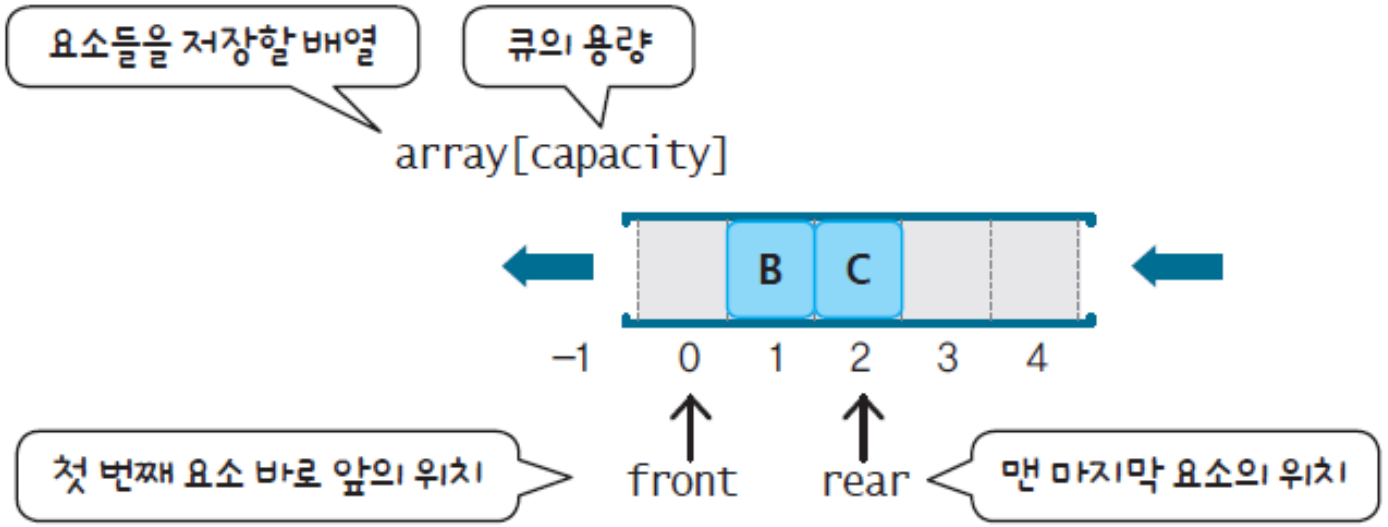


쉽게 추가/삭제 가능한 만큼 관리하기 복잡

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

배열로 큐 구현하기

- 전역변수 데이터
  - 배열 구조 → 용량이 고정된 큐
  - 변수 사용 → 전역 변수

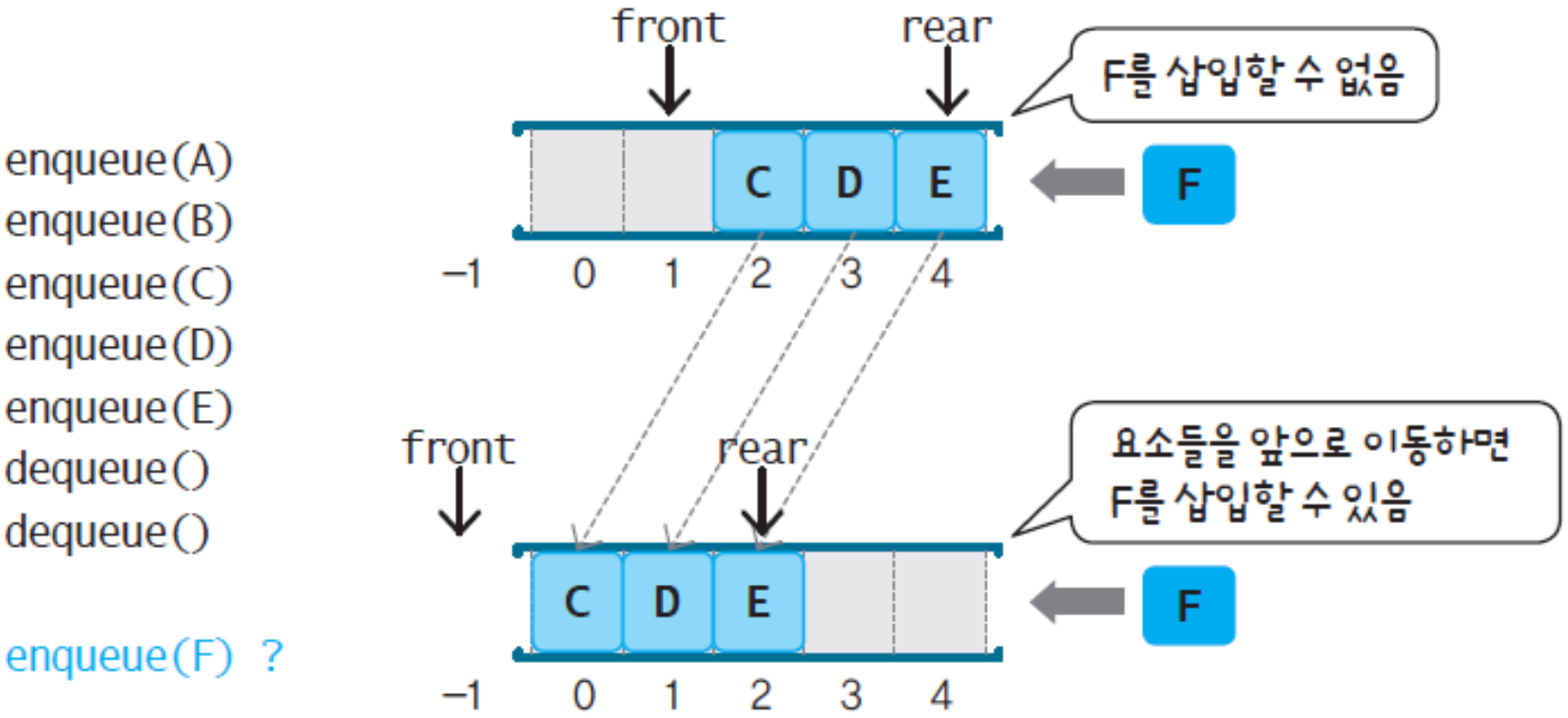


- array[] : 큐 요소들을 저장할 배열
- capacity : 큐에 저장할 수 있는 요소의 최대 개수
- rear : 맨 마지막(후단) 요소의 위치(인덱스)
- front : 첫 번째(전단) 요소 바로 이전의 위치(인덱스)

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

선형 큐의 문제점

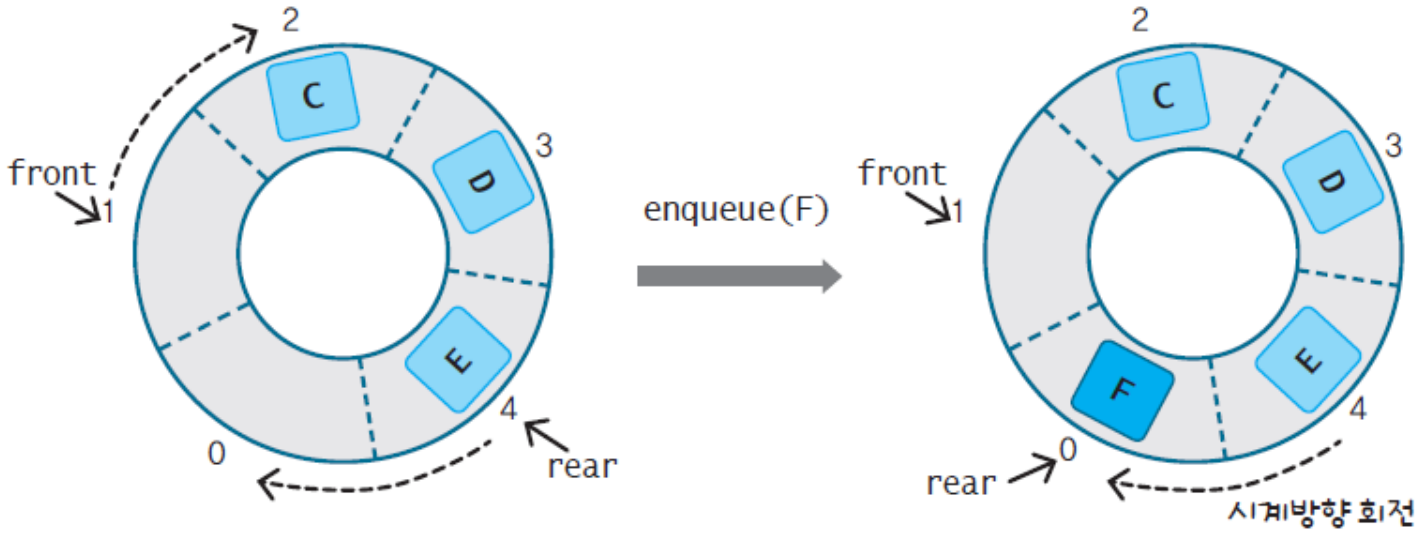
- 요소들의 많은 이동이 필요함



[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

원형 큐의 원리

- 배열을 선형이 아니라 원형으로 생각하는 것
  - 인덱스 front와 rear를 원형으로 회전시키는 개념



- 원형 큐는 인덱스를 시계 방향으로 회전시킴

- 전단 회전 : front ← (front+1) % capacity
- 후단 회전 : rear ← (rear+1) % capacity

- front와 rear가 증가하다가 용량(capacity)과 같아지면 다시 0으로 만들어준다.

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

배열로 큐 구현하기

- 원형 큐 클래스
  - 용량이 고정된 원형 큐 클래스

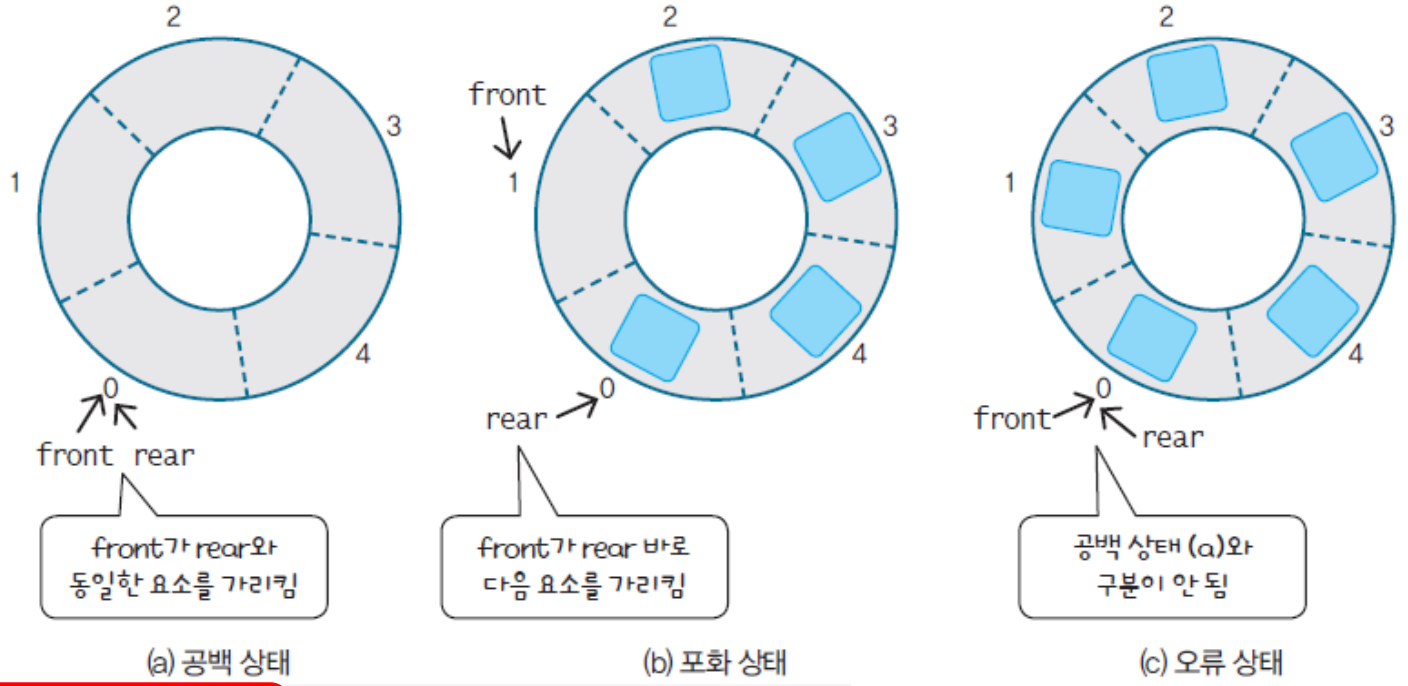
```
class ArrayQueue:
    def __init__(self, capacity=10):
        self.capacity = capacity
        self.array = [None] * capacity
        self.front = 0
        self.rear = 0
```

- array[] : 큐 요소들을 저장할 배열
- capacity : 큐에 저장할 수 있는 요소의 최대 개수
- rear : 맨 마지막(후단) 요소의 위치(인덱스)
- front : 첫 번째(전단) 요소 바로 이전의 위치(인덱스)

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

배열로 큐 구현하기

- 큐 연산
  - 공백 상태 : isEmpty()
  - 포화 상태 : isFull()




```
# 공백 상태
def isEmpty(self):
    return self.front == self.rear

# 포화 상태
def isFull(self):
    return self.front == (self.rear+1)%self.capacity
```

- 원형 큐에서는 보통 하나의 자리를 비워두는 전략을 사용하며 이 규칙을 포화 상태로 정의하기로 함

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

배열로 큐 구현하기

- 큐 연산
  - 추가 상태 : enqueue(e)
  - 삭제 상태 : dequeue()

```
def enqueue(self, item):
    if not self.isFull():
        self.rear = (self.rear+1)%self.capacity
        self.array[self.rear] = item
    else:
        print('stack overflow!')
        pass
```

• 후단 회전

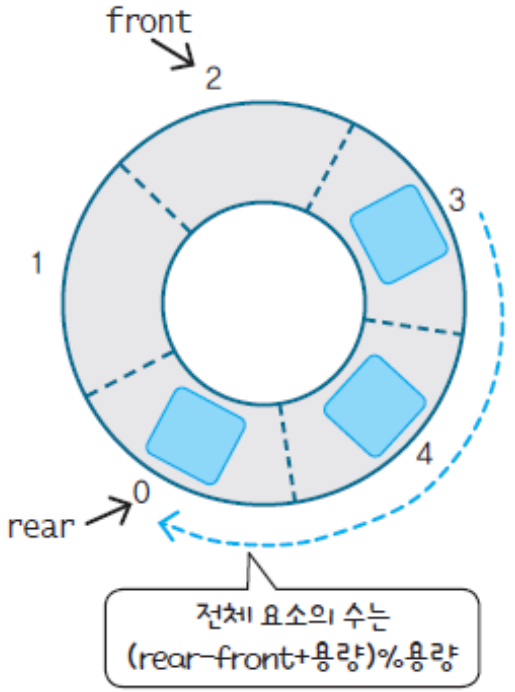
```
def dequeue(self):
    if not self.isEmpty():
        self.front = (self.front+1)%self.capacity
        return self.array[self.front]
```

• 전단 회전

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

배열로 큐 구현하기

- 큐 연산
  - 상단 요소 참조 : peek()
  - 현재 스택 크기 : size()




```
def peek(self):
    if not self.isEmpty():
        return self.array[(self.front+1)%self.capacity]
    else: pass
```

```
def size(self):
    return (self.rear - self.front + self.capacity) % self.capacity
```

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

배열로 큐 구현하기

- 큐 연산
  - 전체 요소 출력 : display()

```
def display(self, msg='Queue: '):
    print(msg, end='=[')
    count = self.size()
    for i in range(count):
        print(self.array[(self.front+1+i)%self.capacity], end=' ')
    print("]")
```

실습 : 배열로 원형 큐 클래스 구현하기

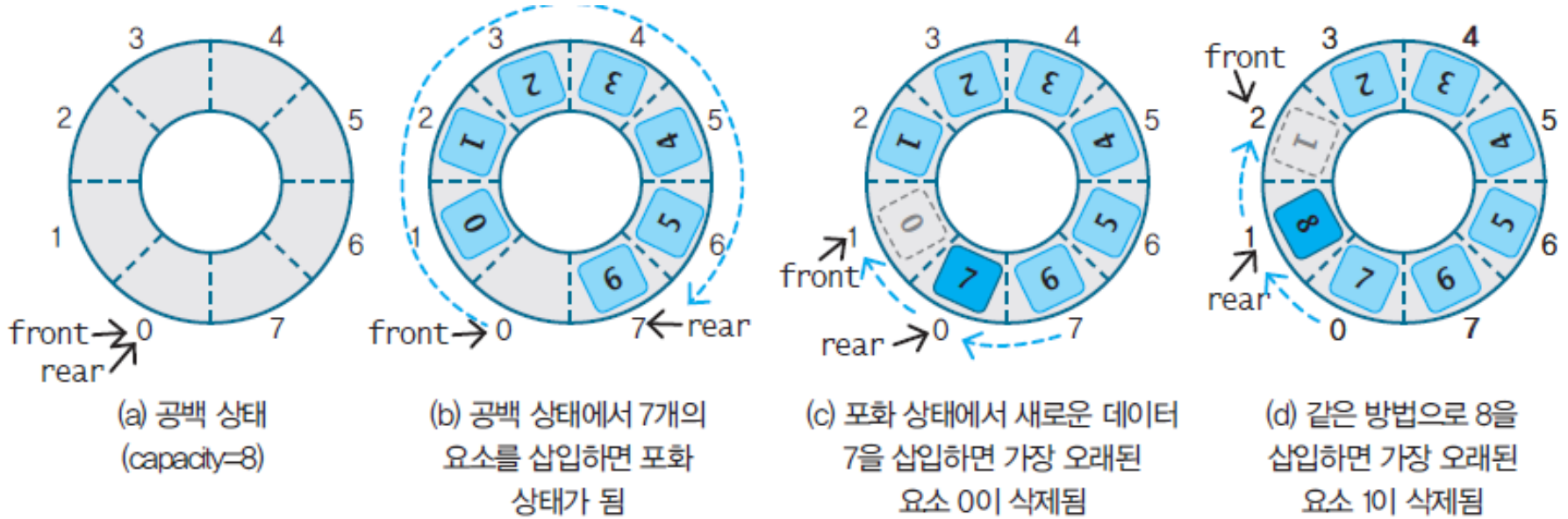
- 원형 큐 클래스를 만들고, 아래 테스트 프로그램으로 실행시켜 보기
  1) 큐 클래스(ArrayQueue) 생성하기
  2) 큐 테스트 프로그램 실행하기

```
import random

q = ArrayQueue(8)

q.display("초기 상태")
while not q.isFull() :
    q.enqueue(random.randint(0,100))
q.display("포화 상태")

print("삭제 순서: ", end='')
while not q.isEmpty() :
    print(q.dequeue(), end=' ')
print()
```

큐 응용

- 큐 응용 : 원형 큐를 링 버퍼(ring buffer)로 사용하기
  - 소스코드나 오래된 자료를 버리고 항상 최근의 자료를 유지하는 용도로 사용

    → 링 버퍼(ring buffer)



(a) 공백 상태 (capacity=8)

(b) 공백 상태에서 7개의 요소를 삽입하면 포화 상태가 됨

(c) 포화 상태에서 새로운 데이터 7을 삽입하면 가장 오래된 요소 0이 삭제됨

(d) 같은 방법으로 8을 삽입하면 가장 오래된 요소 1이 삭제됨

[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

실습 : 원형 큐를 링 버퍼로 구현하기

- 앞에서 만든 원형 큐 클래스에 링 버퍼용 메서드를 추가하고 테스트 하기

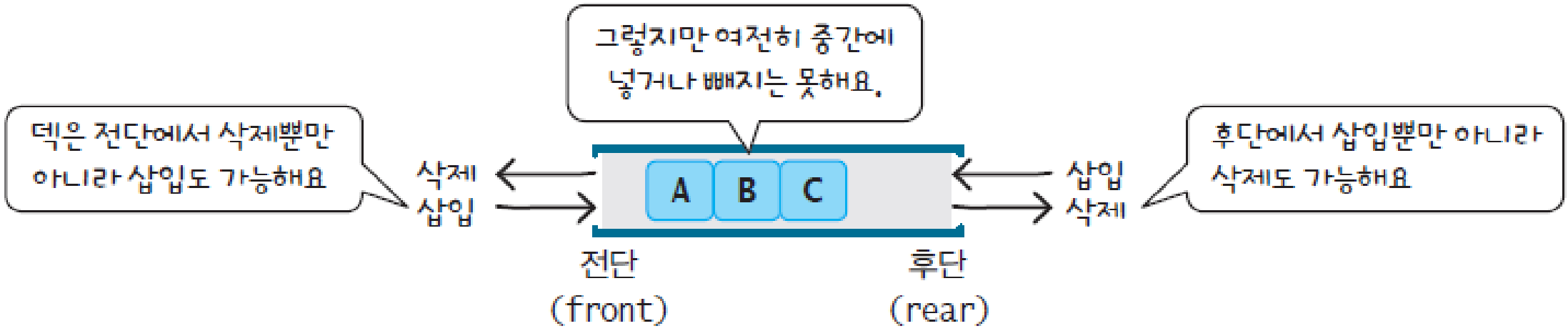
```
def enqueue2(self, item):                    # 링 버퍼 삽입 연산
    self.rear = (self.rear + 1) % self.capacity # 무조건 삽입
    self.array[self.rear] = item
    if self.isEmpty():                       # front == rear
        self.front = (self.front + 1) % self.capacity
```

실습 : 원형 큐를 링 버퍼로 구현하기

```
import random
q = ArrayQueue(8)                    # 큐 객체를 생성(capacity=8)

q.display("초기 상태")
for i in range(6) :                  # enqueue2(): 0, 1, 2, 3, 4, 5
    q.enqueue2(i)
q.display("삽입 0-5")

q.enqueue2(6); q.enqueue2(7)         # enqueue2(): 6, 7
q.display("삽입 6,7")

q.enqueue2(8); q.enqueue2(9)         # enqueue2(): 8, 9
q.display("삽입 8,9")

q.dequeue(); q.dequeue()             # dequeue() 2회
q.display("삭제  x2")
```

덱(Deque)이란?

- 덱(deque) : double-ended queue
  - 전단과 후단에서 모두 삽입과 삭제가 가능한 큐
- 덱의 구조



[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

덱의 연산

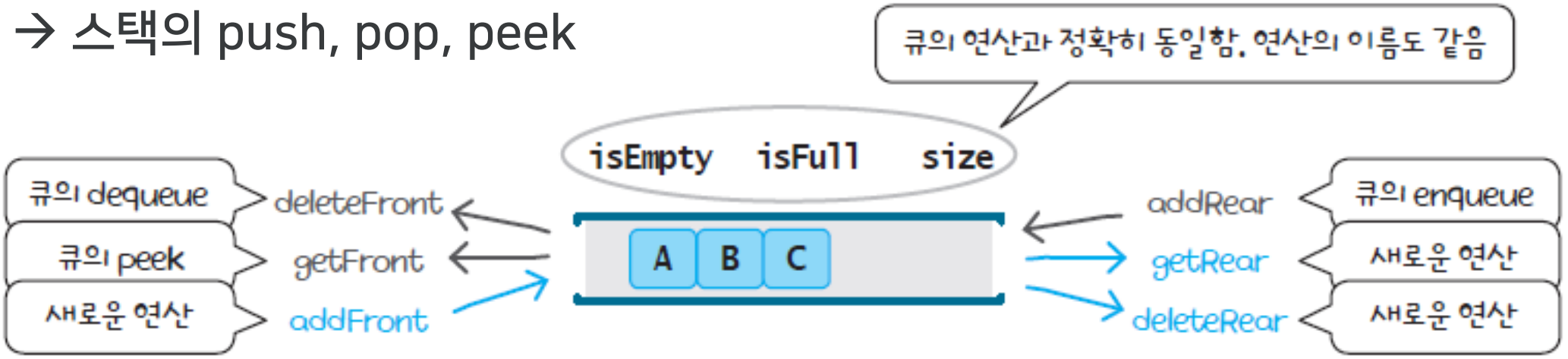
- 덱의 연산
  - addFront(e): 새로운 요소 e를 전단에 추가
  - addRear(e): 새로운 요소 e를 후단에 추가
  - deleteFront(): 덱의 전단 요소를 꺼내서 반환
  - deleteRear(): 덱의 후단 요소를 꺼내서 반환
  - getFront(): 덱의 전단 요소를 삭제하지 않고 반환
  - getRear(): 덱의 후단 요소를 삭제하지 않고 반환
  - isEmpty(): 덱이 비어 있는지 확인, True/False 반환
  - isFull(): 덱이 가득 차 있는지 확인, True/False 반환
  - size(): 덱에 들어 있는 전체 요소의 수 반환

덱의 연산

- 덱은 스택과 큐의 연산을 모두 가짐
  - addRear(e) , deleteFront(), getFront()

    → 큐의 enqueue, dequeue, peek
  - addFront(e), deleteRear(), getRear()

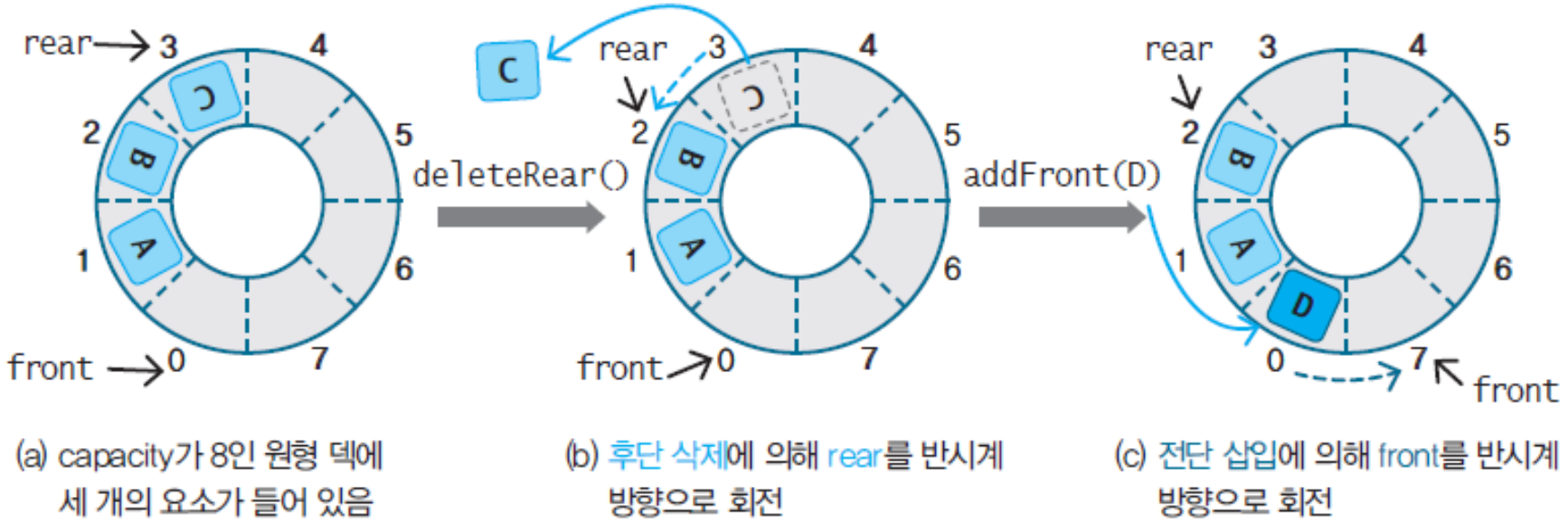
    → 스택의 push, pop, peek



[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

덱의 동작 원리

- 덱의 구조가 큐와 매우 유사 → 원형 덱
  - deleteRear( ), addFront( ) 연산의 예



(a) capacity가 8인 원형 덱에 세 개의 요소가 들어 있음

(b) 후단 삭제에 의해 rear를 반시계 방향으로 회전

(c) 전단 삽입에 의해 front를 반시계 방향으로 회전

- 덱은 인덱스를 반시계 방향의 회전도 필요함

- 전단 회전(반시계 방향) : front ← (front-1+capacity) % capacity
- 후단 회전(반시계 방향) : rear ← (rear-1+capacity) % capacity

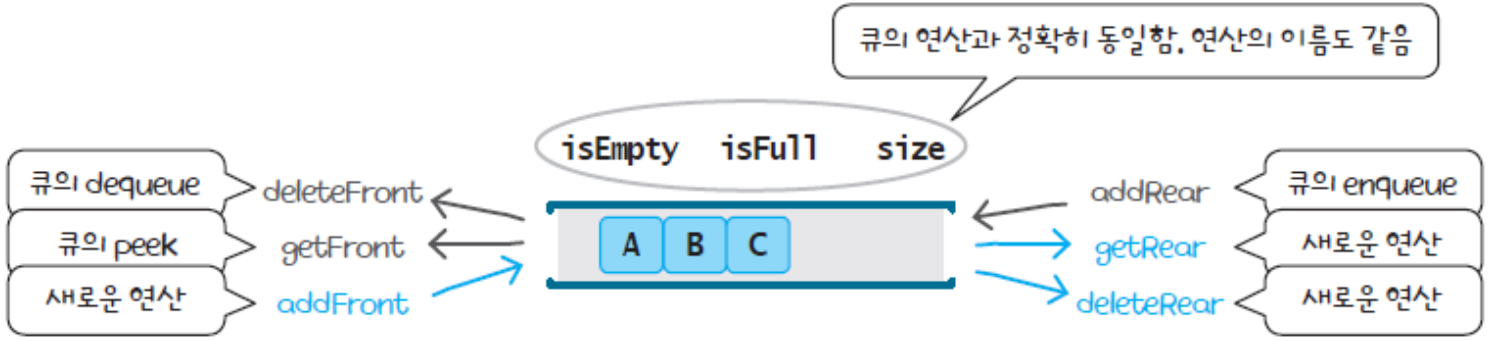
[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

상속을 이용하여 덱 구현하기

- 원형 큐를 상속한 원형 덱
  - 데이터는 추가로 정의할 필요가 없음
  - isEmpty, isFull도 추가로 정의할 필요 없음
  - addRear, deleteFront, getFront:

    → 단순히 원형 큐의 enqueue, dequeue, peek 연산을 호출하면 됨
  - addFront, deleteRear, getRear는 원형 큐에 없는 연산이므로 새로 구현 필요

• 코드에서 처리할 부분



[그림 출처] : 자료구조와 알고리즘 with 파이썬 by 생능북스

실습 : 원형 큐 상속을 이용한 덱 구현하기

- 앞에서 만든 원형 큐 클래스를 상속받아 덱 구현하기

```
# 원형 큐 클래스 상속받아 원형 덱 클래스 생성
class CircularDeque(ArrayQueue) :
    def __init__( self, capacity=10 ) :
        super().__init__(capacity)
```

- 부모(super()) 생성자 직접 호출하여 초기화

```
    # 원형 덱: 동작이 동일한 연산들
    def addRear( self, item ): self.enqueue(item )
    def deleteFront( self ): return self.dequeue()
    def getFront( self ): return self.peek()
```

실습 : 원형 큐 상속을 이용한 덱 구현하기

```
# 원형 덱: 추가된 연산들
def addFront( self, item ):
    if not self.isFull():
        self.array[self.front] = item
        self.front = (self.front - 1 + self.capacity) % self.capacity
    else: pass

def deleteRear( self ):
    if not self.isEmpty():
        item = self.array[self.rear];
        self.rear = (self.rear - 1 + self.capacity) % self.capacity
        return item
    else: pass

def getRear( self ):
    if not self.isEmpty():
        return self.array[self.rear]
    else: pass
```

- 전단 회전(반시계방향)
- 후단 회전(반시계방향)

파이썬에서 큐와 덱 사용하기

- 파이썬에서 큐와 덱 사용하는 방법
  - 방법 1) queue 모듈의 Queue 사용하기
  - 방법 2) collections모듈의 deque 클래스 사용하기

파이썬에서 큐와 덱 사용하기

- 방법 1) queue 모듈의 Queue 사용하기

```
import queue
q = queue.Queue(8)
```

연산	ArrayQueue	queue.Queue
삽입/삭제	enqueue(), dequeue()	put(), get()
공백/포화 상태 검사	isEmpty(), isFull()	empty(), full()
전단 들여다보기	peek()	제공하지 않음

파이썬에서 큐와 덱 사용하기

- 방법 2) collections모듈의 deque 클래스 사용하기

```
import collections
dq = collections.deque()
```

연산	CircularDeque	collections.deque
전단 삽입/삭제	addFront(), deleteFront()	appendleft(), popleft()
후단 삽입/삭제	addRear(), deleteRear()	append(), pop()
공백 상태 검사	isEmpty()	dq (예: if dq : …)
포화 상태 검사	isFull()	의미 없음
들여다보기	getFront(), getRear()	제공하지 않음

리스트(List)

Q & A

Next Topic

- 점화식과 재귀 알고리즘(Recursive Algorithm)

다음 시간에 만나요!