파이썬 프로그래밍

16. 알고리즘(1)

00 Contents

❖ 수업 목표

- 스택 자료의 구조를 설명할 수 있다.
- 스택을 리스트와 클래스로 구현할 수 있다.
- 큐 자료의 구조를 설명할 수 있다.
- 큐를 리스트와 클래스로 구현할 수 있다.

❖ 세부 목표

- 16.1 스택(Stack)(1)
- 16.2 스택(Stack)(2)
- 16.3 큐(Queue)(1)
- 16.4 큐(Queue)(2)
- 16.5 덱(Deque)(1)
- 16.6 덱(Deque)(2)

1. 스택(Stack)(1)

❖ 알고리즘(Algorithm)

- 문제를 해결하기 위한 명확한 절차나 방법
- 알고리즘을 잘 이해하면 코드를 효율적으로 작성하고 속도를 최적화할 수 있음

❖ 자료 구조(Data Structure)

- 자료구조 소개
- 데이터를 저장하고 관리하는 방법
- 적절한 자료구조를 선택하면 속도와 효율성을 높일 수 있음
- 주요 자료구조 : 리스트, 튜플, 딕셔너리, 세트,

1. 스택(Stack)(1)

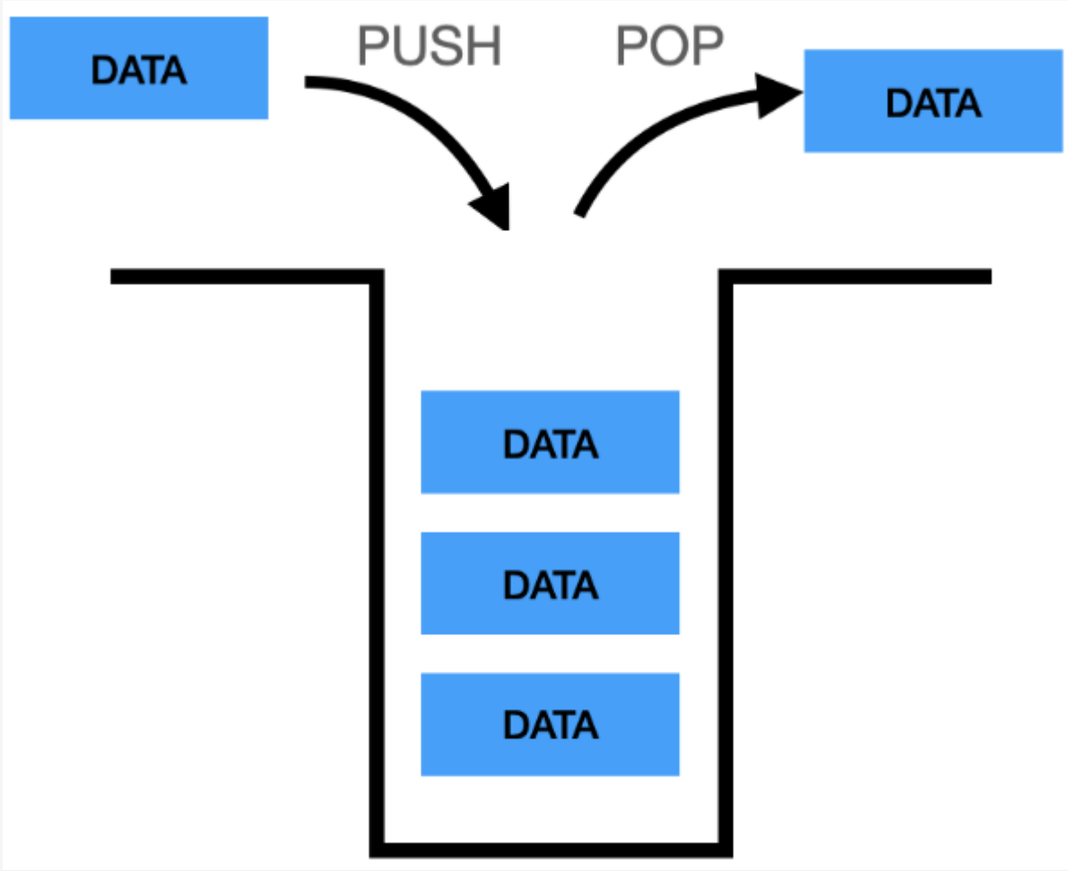
❖ 스택(Stack) 개요

- 데이터를 처리하는 기본 자료구조
- 데이터의 삽입과 삭제가 상단(top)에서만 발생
- 후입선출(Last In, First Out, LIFO) 형태로 데이터 처리
  - 가장 마지막에 입력된 데이터가 가장 먼저 제거되는 구조
  - 감자칩 통에 감자칩을 넣고 빼는 경우,
    먼저 넣은 감자칩을 나중에 넣은 감자칩보다 앞서 꺼낼 수 없듯이,
    데이터를 처리하는 경우에도
    먼저 넣은 데이터보다 나중에 넣은 데이터를 우선 처리하는 방법

1. 스택(Stack)(1)

❖ 스택 연산(데이터 처리) 방식

- push : 스택에 데이터를 삽입하는 동작
- pop : 스택에서 데이터를 제거하는 동작



1. 스택(Stack)(1)

❖ 스택 활용 사례

- 함수 호출 스택
- 웹 브라우저 방문 기록 (뒤로 가기)
- 괄호 검사와 같은 문자열 처리

2. 스택(Stack)(2)

❖ 스택 구현

- 리스트 활용

- 빈 스택 구현
  - stack = []

- push 연산 구현
  - stack = [1, 2, 3]
  - stack.append(4) # 4
  - print("stack = ", stack) # stack = [1, 2, 3, 4]

- pop 연산 구현
  - stack = [1, 2, 3]
  - stack.pop() # 3
  - print("stack = ", stack) # stack = [1, 2]

2. 스택(Stack)(2)

❖ 스택 구현

- 클래스 활용
  - 스택 리스트 생성
  - push 메서드 구현
  - pop 메서드 구현
    - 스택이 비어 있는 경우 고려
  - 스택 내 데이터 유무 확인 메서드 구현
  - 스택 최상위 데이터 확인 메서드 구현
    - 스택이 비어 있는 경우 고려
  - 스택 상태 반환 메서드

```python
class Stack:
    def __init__(self):
        self.stack = []

    def push(self, data):
        self.stack.append(data)

    def pop(self):
        if not self.is_empty():
            return self.stack.pop()
        return

    def is_empty(self):
        if len(self.stack) == 0:
            return True
        return False

    def peak(self):
        if not self.is_empty():
            return self.stack[-1]
        return

    def status_stack(self):
        return self.stack
```

2. 스택(Stack)(2)

❖ 스택 구현

- 스택 객체 생성 및 동작 확인

```
s1 = Stack()
print(s1.peak())
s1.pop()
s1.push(1)
s1.push(2)
s1.pop()
s1.push(3)
s1.push(4)
print(s1.peak())
print(s1.status_stack())
```

- 실행 결과
  - None
  - 4
  - [1, 3, 4]

3. 큐(Queue)(1)

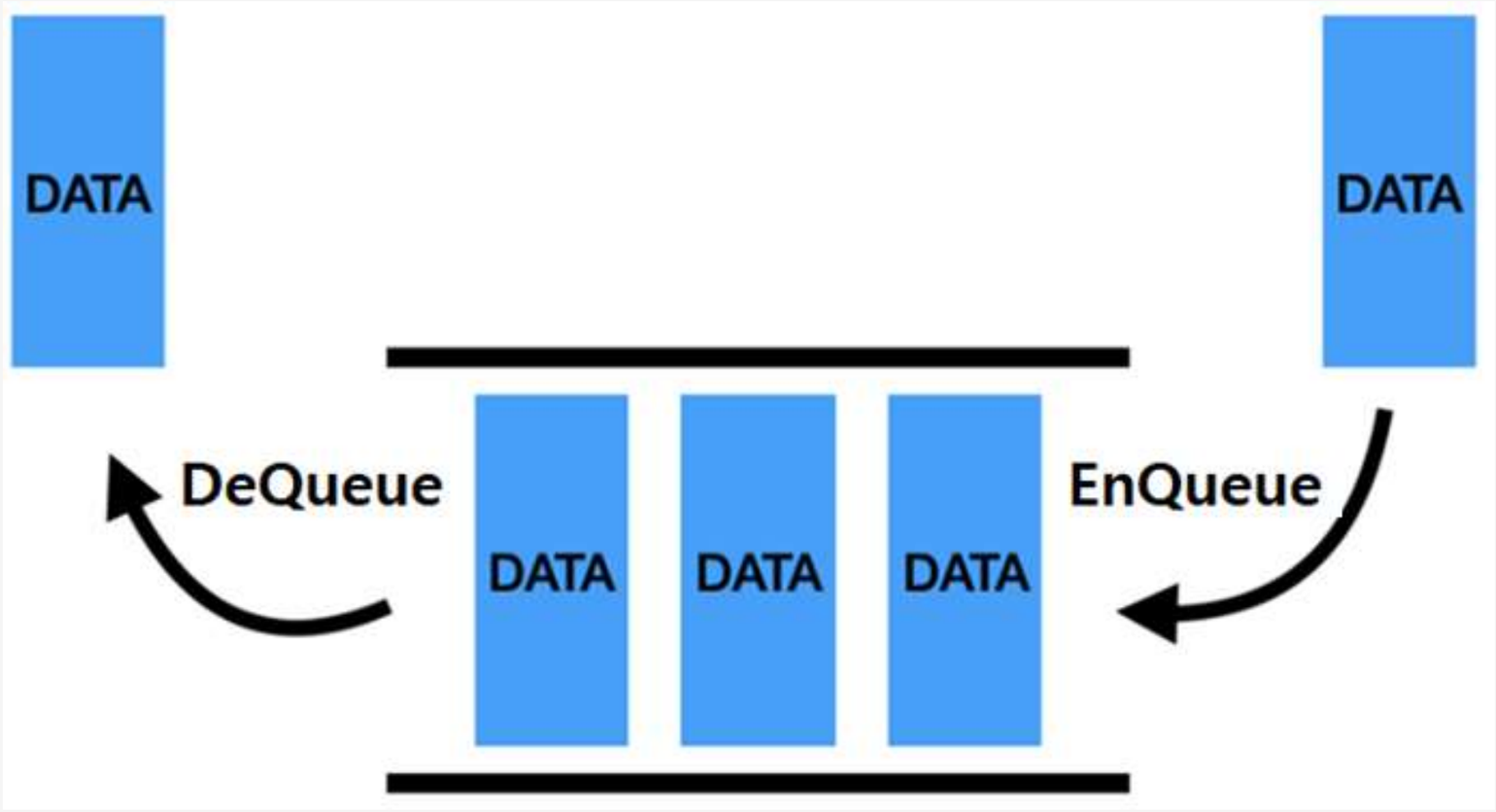
❖ 큐(Queue) 개요

- 데이터를 처리하는 기본 자료구조
- 선입선출(First In, First Out, FIFO) 형태로 데이터 처리
  - 마지막에 데이터를 입력하고 가장 먼저 입력된 데이터가 먼저 제거되는 구조
  - 일상 생활에서 줄 서는 모습과 동일, 먼저 온 사람이 우선 일을 처리하고, 마지막에 온 사람은 나중에 일을 처리하는 모습

3. 큐(Queue)(1)

❖ 큐 연산(데이터 처리) 방식

- Enqueue : 큐에 데이터를 삽입하는 동작
- Dequeue : 큐에서 데이터를 제거하는 동작



3. 큐(Queue)(1)

❖ 큐의 활용 사례

- 프로세스 관리 (예: 작업 대기열)
  - 운영체제의 프로세스를 관리 방법 중 하나로 줄을 세우고 순서에 따라 처리
    - 프로세스: 실행 중에 있는 프로그램
- BFS (너비 우선 탐색) 구현
  - 다음 차시 참고

4. 큐(Queue)(2)

❖ 큐 구현

- 리스트 활용

- 빈 큐 구현
  - queue = []

- enqueue 연산 구현
  - queue = [1, 2, 3]
  - queue.append(4) # 4
  - print(" queue = ", stack) # queue = [1, 2, 3, 4]

- dequeue 연산 구현
  - queue = [1, 2, 3]
  - queue.pop(0) # 1
  - print(“ queue = “, queue) # queue = [2, 3]

4. 큐(Queue)(2)

❖ 큐 구현

- 클래스 활용
  - 큐 리스트 생성
  - enqueue 메서드 구현
  - dequeue 메서드 구현
    - 큐가 비어 있는 경우 고려
  - 큐 내 데이터 유무 확인 메서드 구현
  - 큐 상태 반환 메서드

```
1   class Queue:
2       def __init__(self):
3           self.queue = []
4
5       def enqueue(self, data):
6           self.queue.append(data)
7
8       def dequeue(self):
9           if not self.is_empty():
10              return self.queue.pop(0)
11          return
12
13      def is_empty(self):
14          if len(self.queue) == 0:
15              return True
16          return False
17
18      def status_queue(self):
19          return self.queue
```

4. 큐(Queue)(2)

❖ 큐 구현

- 큐 객체 생성 및 동작 확인

```
q1 = Queue()
q1.dequeue()
q1.enqueue(1)
q1.enqueue(2)
q1.dequeue()
print(q1.status_queue())
q1.enqueue(3)
q1.enqueue(4)
q1.dequeue()
print(q1.status_queue())
```

- 실행 결과
  - [2]
  - [3, 4]

5. 덱(Deque)(1)

❖ 덱(Deque) 개요

- 양쪽 끝에서 삽입과 삭제가 가능한 자료구조
- 스택과 큐를 병합한 형태의 자료구조
- 파이썬 내장 모듈 collections 활용
- 덱(Deque)은 Double Ended QUEue의 약자

5. 덱(Deque)(1)

❖ 덱의 활용 사례

- 회전 큐 (예: 슬라이딩 윈도우)
  - 고정 사이즈의 윈도우가 이동하며 윈도우 내부 데이터를 이용해 문제를 풀이
- 문자열 회문 검사
  - 회문: 순서를 거꾸로 읽어도 제대로 읽은 것과 같은 단어와 문장
    - 예) "level", "SOS", "rotator"

5. 덱(Deque)(1)

❖ 덱 연산(데이터 처리) 방식

- append : 덱의 마지막에 데이터를 삽입하는 동작
- appendleft : 덱의 처음에 데이터를 삽입하는 동작
- pop : 덱의 마지막 데이터를 제거하는 동작
- popleft : 덱의 처음 데이터를 제거하는 동작

6. 덱(Deque)(2)

❖ 덱 구현

- collections 모듈 활용
- deque 클래스 활용

- 덱 객체 생성
- append 메서드 구현
- appendleft 메서드 구현
- pop 메서드 구현
- popleft 메서드 구현

- 실행 결과
  - deque([1])
  - deque([2, 1])
  - deque([2])
  - deque([])

```
1   from collections import deque
2
3   dq = deque()          # 덱 생성
4
5   dq.append(1)          # dq에 뒤로 데이터 넣기
6   print(dq)
7
8   dq.appendleft(2)      # dq에 앞으로 데이터 넣기
9   print(dq)
10
11  dq.pop()              # 마지막 데이터 꺼내기
12  print(dq)
13
14  dq.popleft()          # 처음 데이터 꺼내기
15  print(dq)
```

7. 연습 문제

❖ 문제 1: 괄호 짝 검사

- 문자열을 인수로 받아 포함된 괄호의 짝이 올바르게 사용되었는지 확인하는 프로그램을 작성하시오
  - 스택 활용
  - 괄호의 짝이 맞으면 True 반환
  - 괄호의 짝이 맞지 않으면 False 반환

7. 연습 문제

❖ 문제 1: 괄호 짝 검사

7. 연습 문제

❖ 문제 2: 회전 큐 구현

- 데이터를 양방향으로 삽입하거나 제거할 수 있는 회전 큐를 구현하시오.
  - 덱 사용
  - 리스트로 구현
  - 왼쪽으로 2만큼 회전

7. 연습 문제

❖ 문제 2: 회전 큐 구현

8. 세트

❖ set(세트) 개요

- 중복을 허용하지 않는 자료구조

- 예제 코드)

```
# 세트 생성 (중복 제거)
numbers = {1, 2, 3, 3, 4}
print(numbers)  # {1, 2, 3, 4}

# 요소 추가 및 제거
numbers.add(5)
numbers.remove(3)

print(numbers)  # {1, 2, 4, 5}
```

8. 세트

❖ set(세트) 개요

- 중복을 허용하지 않는 자료구조

- 집합 연산
  - 예제 코드)
    - set1 = {1, 2, 3}
    - set2 = {3, 4, 5}

    - # 교집합
    - print(set1 & set2) # {3}

    - # 합집합
    - print(set1 | set2) # {1, 2, 3, 4, 5}

    - # 차집합
    - print(set1 - set2) # {1, 2}

❖ 과제

- 1. 스택 리스트로 구현하기
- 2. 스택 클래스로 구현하기
- 3. 큐 리스트로 구현하기
- 4. 큐 클래스로 구현하기

❖ 다음 수업 내용

- 알고리즘 2
  - 그래프(Graph)
  - DFS(Depth-First-Search)
  - BFS(Breadth First Search)