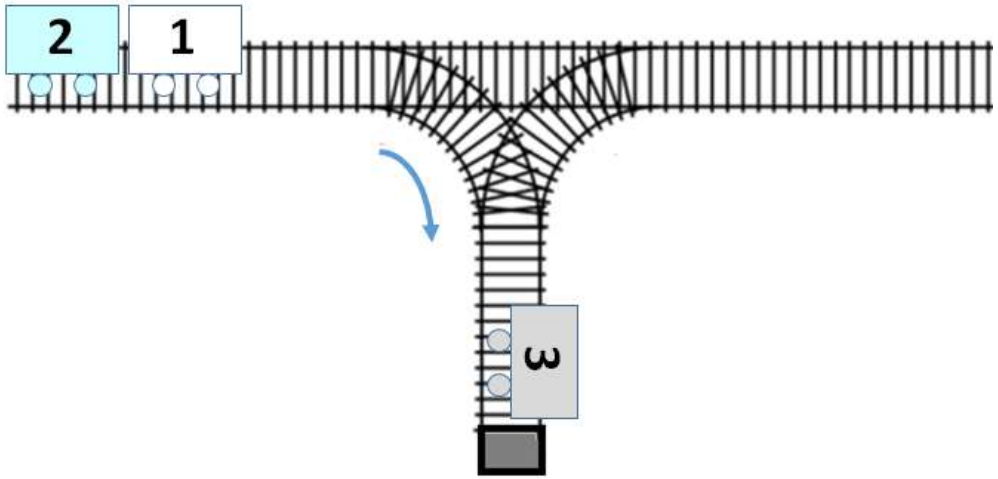
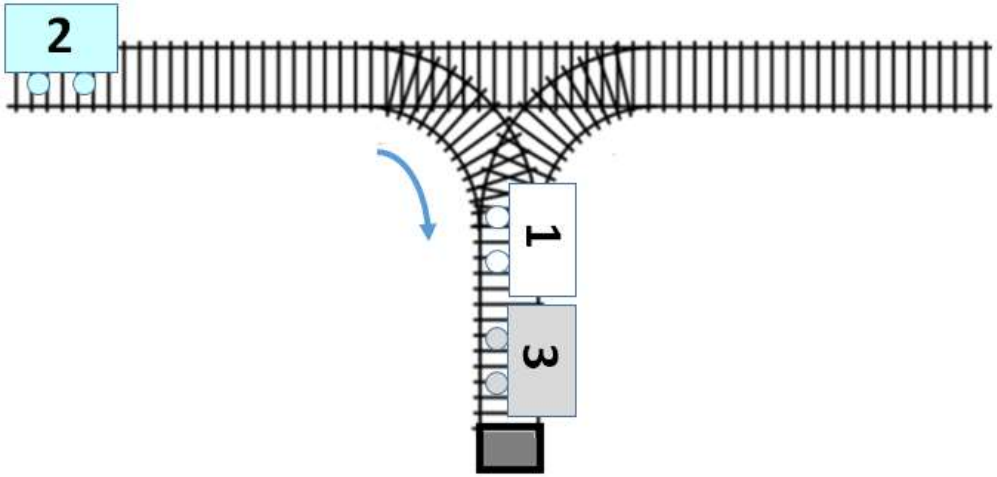
# 자료구조/알고리즘

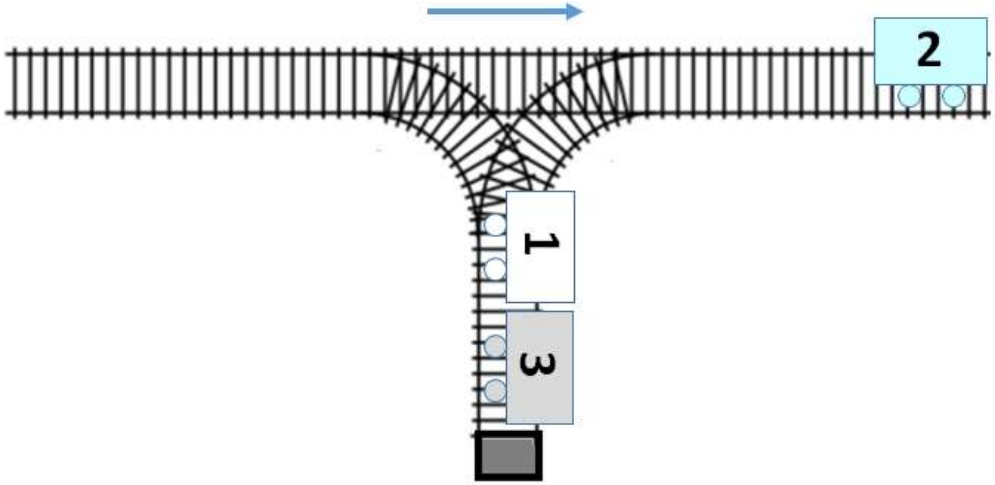
## 5주차 스택(Stack)

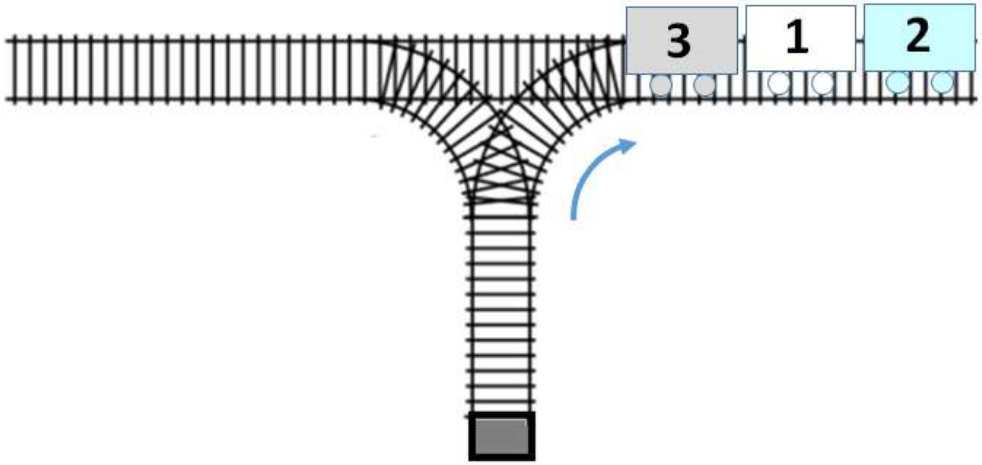
# 3.1 스택

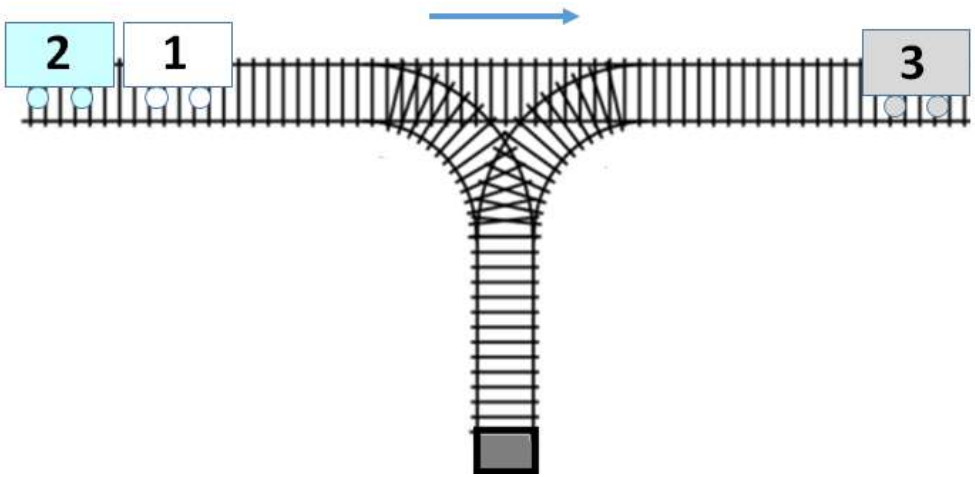
- 한 쪽 끝에서만 item(항목)을 삭제하거나 새로운 item을 저장하는 자료구조
- 새 item을 저장하는 연산: push
- Top item을 삭제하는 연산: pop
- 후입 선출(Last-In First-Out, LIFO) 원칙 하에 item의 삽입과 삭제 수행

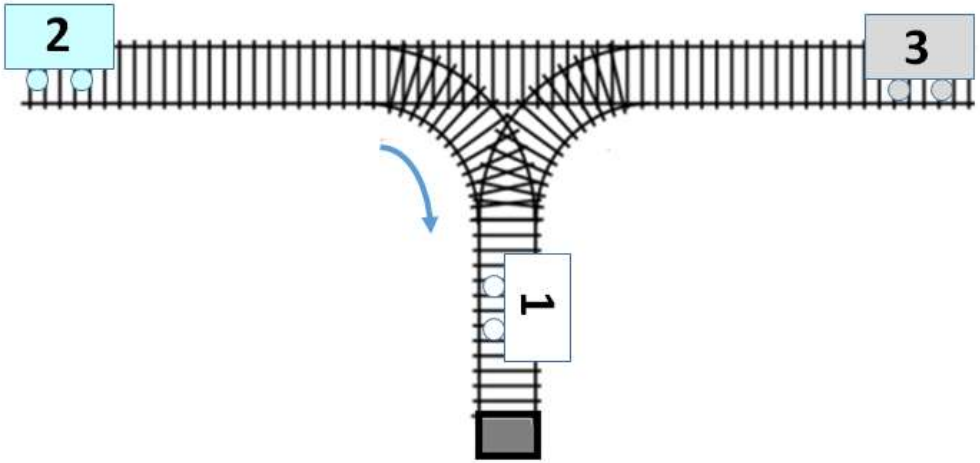

2
1
3

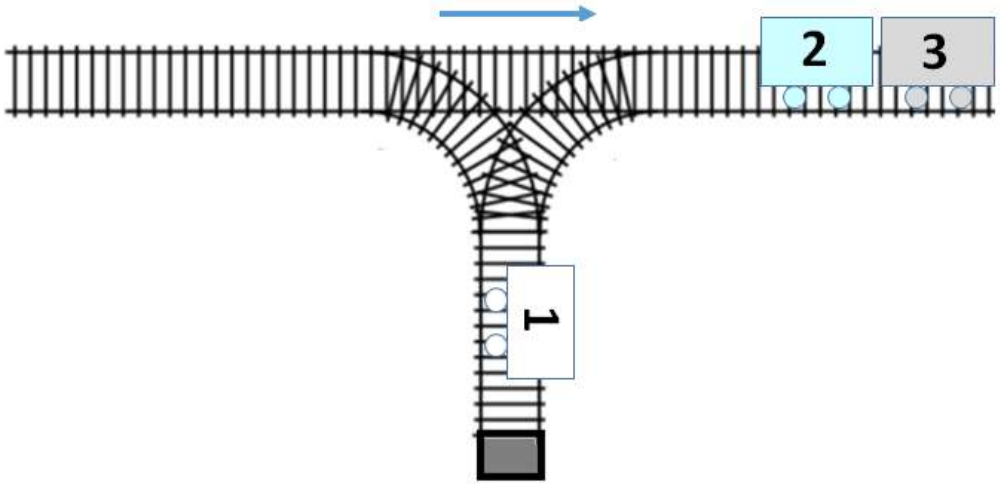

2
1
3

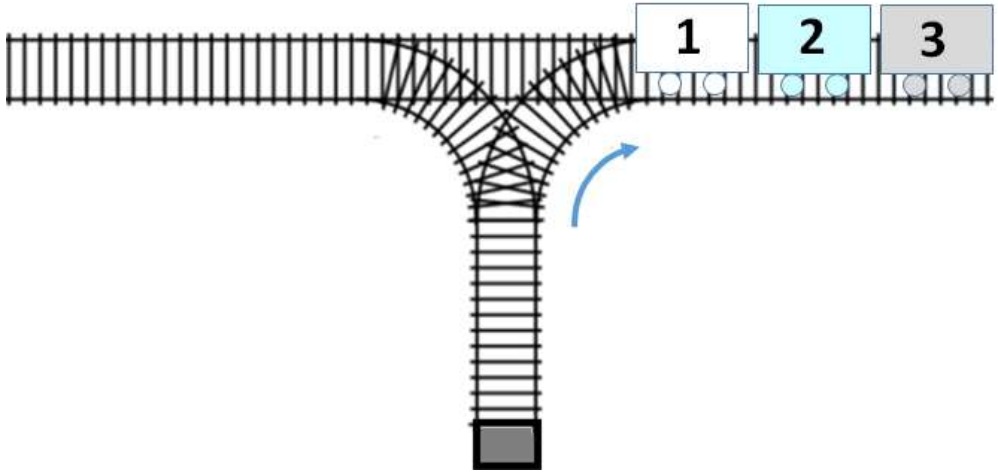

2
1
3

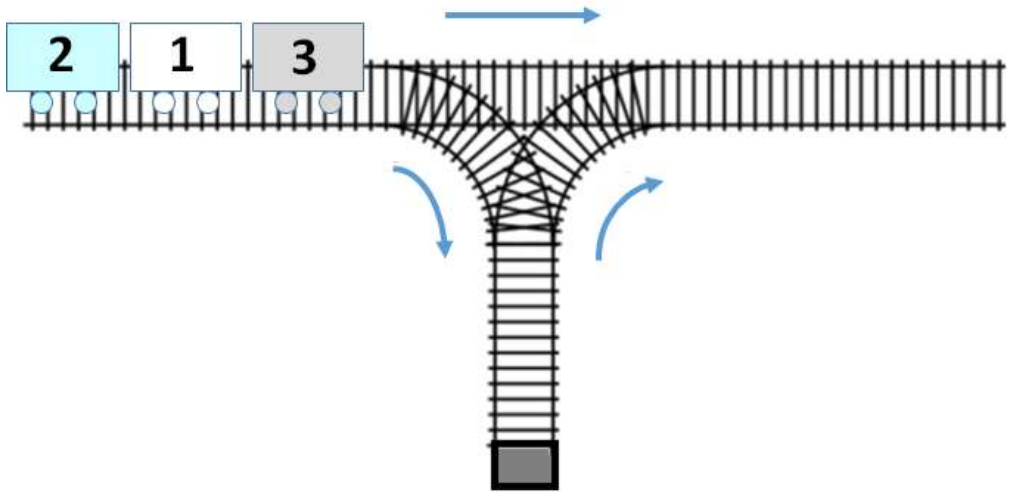

3
1
2

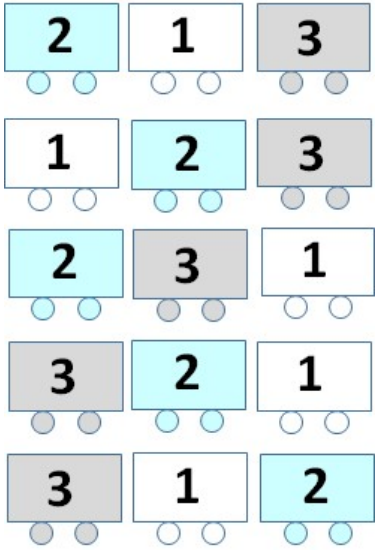

2
1
3

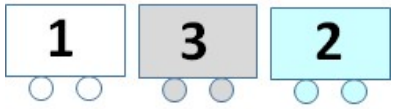

2
3
1

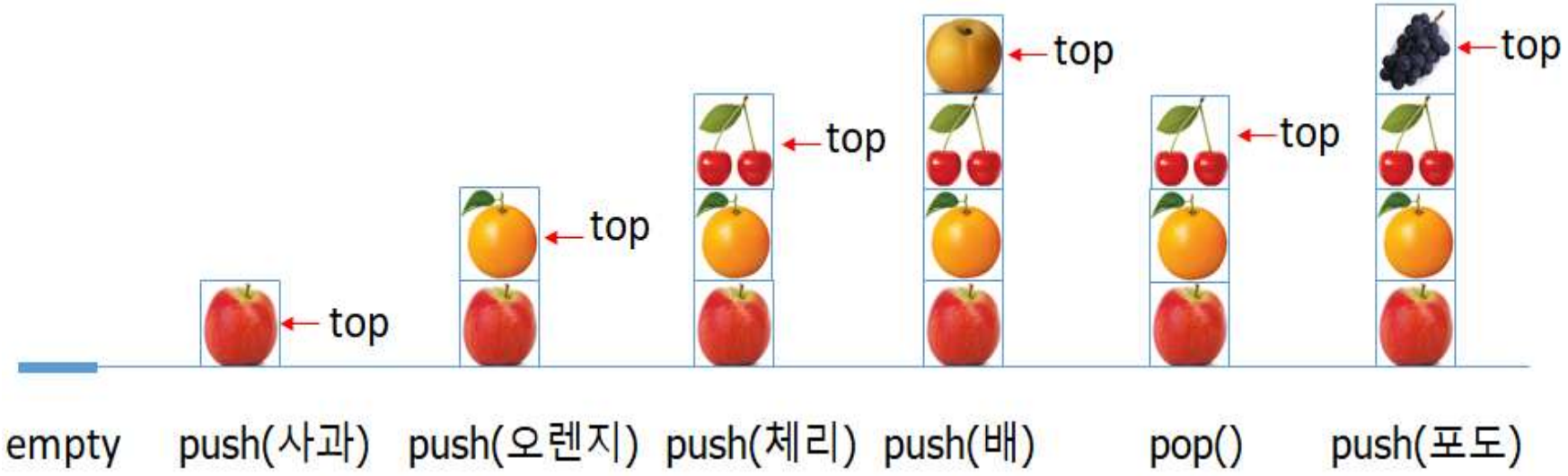

2
3
1


1
2
3

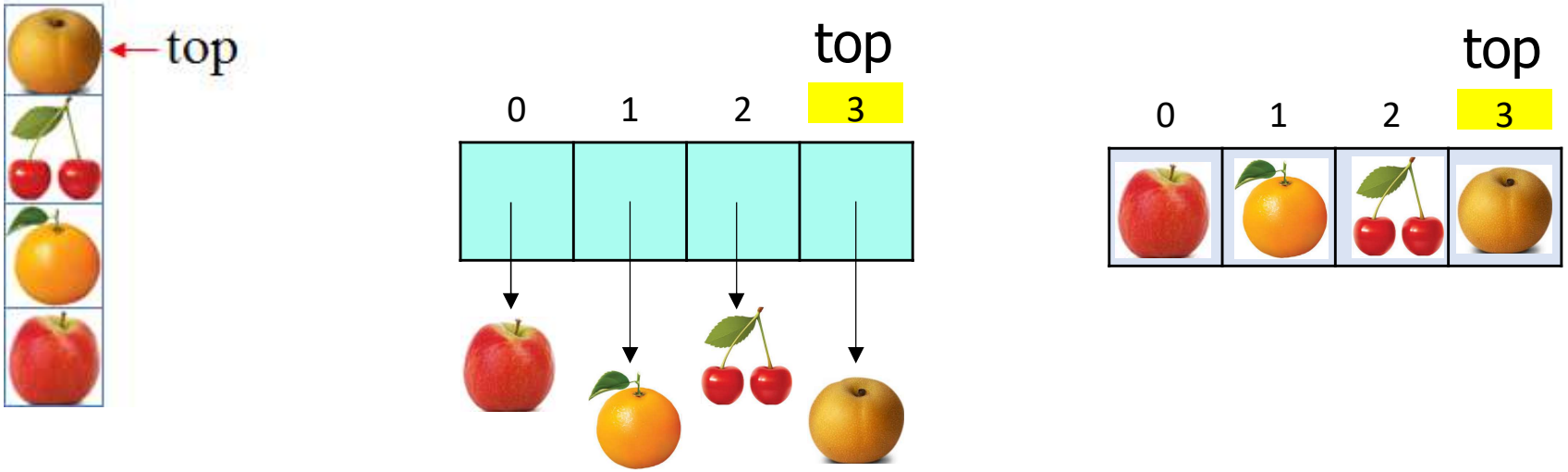

2
1
3


2 1 3
1 2 3
2 3 1
3 2 1
3 1 2


1 3 2




[그림 3-2] 스택의 push와 pop 연산




[그림 3-3] 리스트로 구현된 스택

# 스택 ADT

**정의 4.1** Stack ADT

데이터: 후입선출(LIFO)의 접근 방법을 유지하는 항목들의 모음

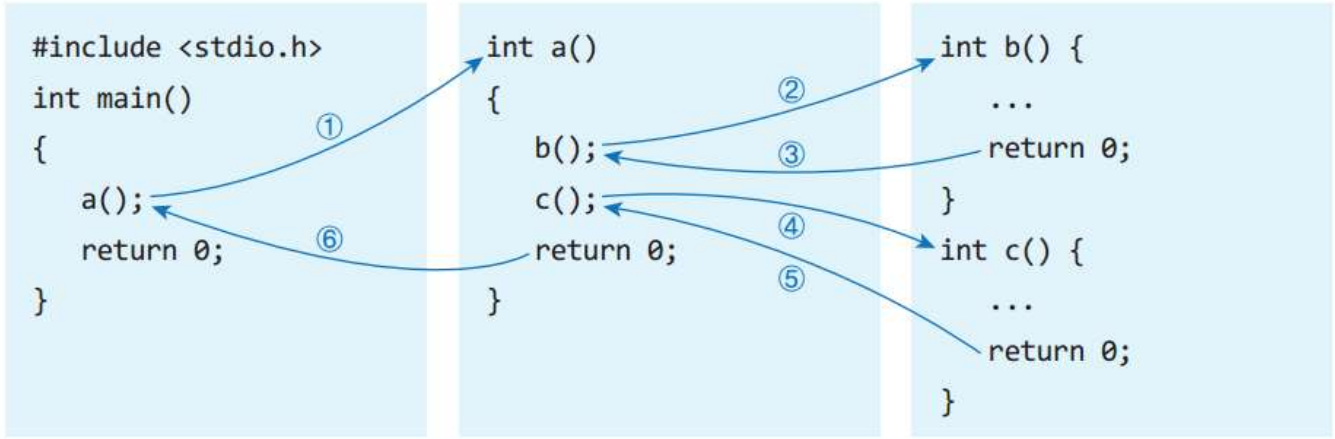
연산

- Stack(): 비어 있는 새로운 스택을 만든다.
- isEmpty(): 스택이 비어있으면 True를 아니면 False를 반환한다.
- push(e): 항목 e를 스택의 맨 위에 추가한다.
- pop(): 스택의 맨 위에 있는 항목을 꺼내 반환한다.
- peek(): 스택의 맨 위에 있는 항목을 삭제하지 않고 반환한다.
- size(): 스택내의 모든 항목들의 개수를 반환한다.
- clear(): 스택을 공백상태로 만든다.

# 스택의 용도

- 되돌리기

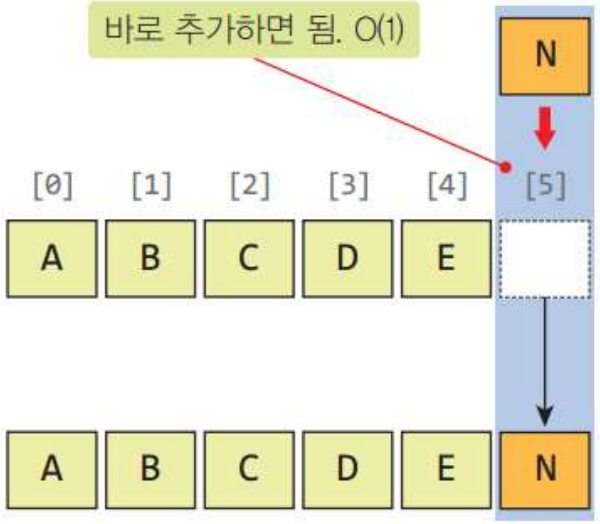

- 함수호출



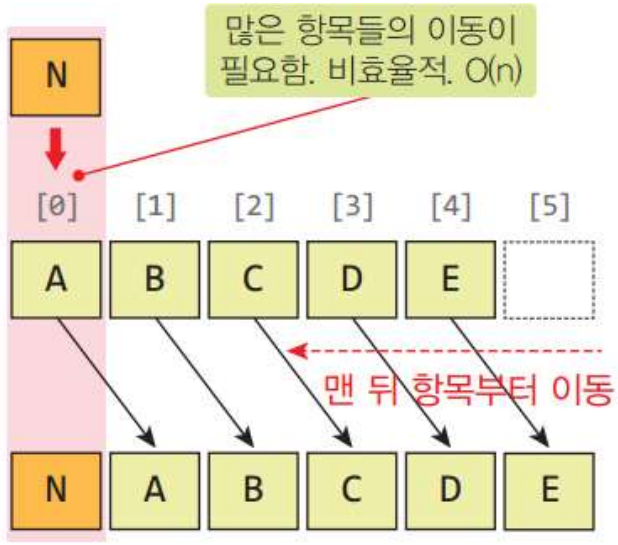

- 괄호 검사
- 계산기: 후위 표기식 계산, 중위 표기식의 후위 표기식 변환
- 미로 탐색 등

# 스택의 구현(리스트(배열) 구조)

- 데이터
  - top: 스택 항목을 저장하는 파이썬 리스트
  - 항목의 개수는 len(top)으로 구할 수 있음
- 연산: isEmpty(), push(), pop(), peek(), display()
- 항목 삽입/삭제 위치: 리스트의 맨 뒤가 유리함. Why?




파이썬 리스트의 후단을 사용하는 경우




파이썬 리스트의 전단을 사용하는 경우

# 리스트로 구현한 스택

```
01 def push(item): # 삽입 연산
02     stack.append(item)
03
04 def peek(): # top 항목 접근
05     if len(stack) != 0:
06         return stack[-1]
07
08 def pop(): # 삭제 연산
09     if len(stack) != 0:
10         item = stack.pop(-1)
11         return item
12 stack = []
```

push() = append()
리스트의 맨 뒤에 item 추가

top 항목
= 리스트의 맨 뒤 항목 리턴

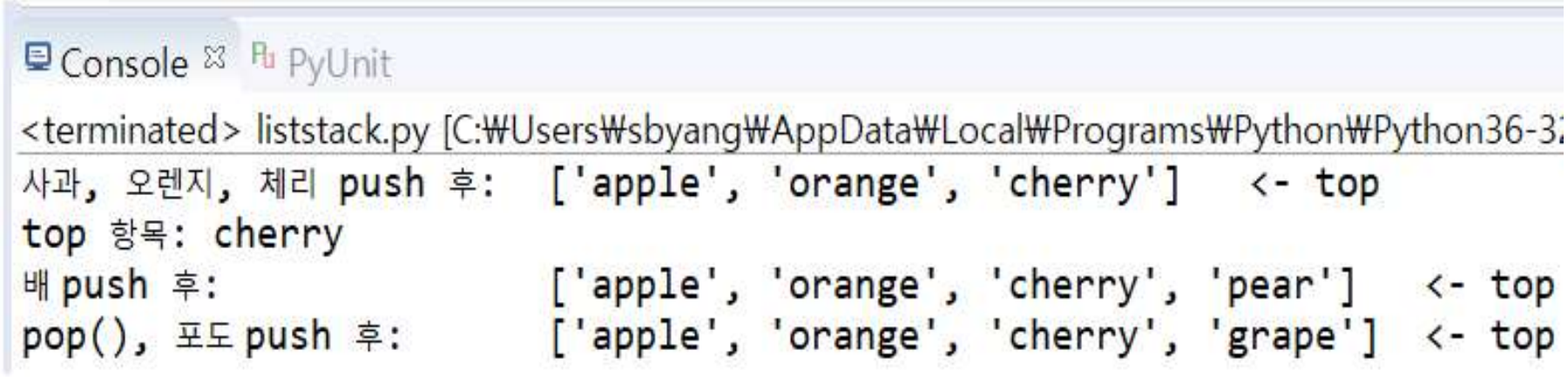
pop()
리스트의 맨 뒤에 있는 항목 제거

리스트 선언

```
13 push('apple')
14 push('orange')
15 push('cherry')
16 print('사과, 오렌지, 체리 push 후:\t', end='')
17 print(stack, '\t<- top')
18 print('top 항목: ', end='')
19 print(peek())
20 push('pear')
21 print('배 push 후:\t\t', end='')
22 print(stack, '\t<- top')
23 pop()
24 push('grape')
25 print('pop(), 포도 push 후:\t', end='')
26 print(stack, '\t<- top')
```

일련의 스택 연산과 출력

[프로그램 3-1]



# 단순연결리스트로 구현한 스택

```
01 class Node: # Node 클래스
02     def __init__(self, item, link):
03         self.item = item
04         self.next = link
05
06 def push(item): # push 연산
07     global top
08     global size
09     top = Node(item, top)
10     size += 1
11
12 def peek(): # peek 연산
13     if size != 0:
14         return top.item
15
```

노드 생성자
항목과 다음 노드 레퍼런스

전역 변수

새 노드 객체를 생성하여
연결리스트의 첫 노드로 삽입

top 항목만 리턴

```
16 def pop(): # pop 연산
17     global top
18     global size
19     if size != 0:
20         top_item = top.item
21         top = top.next
22         size -= 1
23         return top_item

24 def print_stack(): # 스택 출력
25     print('top ->\t', end='')
26     p = top
27     while p:
28         if p.next != None:
29             print(p.item, '-> ', end='')
30         else:
31             print(p.item, end='')
32         p = p.next
33     print()
```

전역 변수

연결리스트에서 top이 참조하던 노드 분리시킴

제거된 top 항목 리턴

```
34 top = None
35 size = 0
36 push('apple')
37 push('orange')
38 push('cherry')
39 print('사과, 오렌지, 체리 push 후:\t', end='')
40 print_stack()
41 print('top 항목: ', end='')
42 print(peek())
43 push('pear')
44 print('배 push 후:\t\t', end='')
45 print_stack()
46 pop()
47 push('grape')
48 print('pop(), 포도 push 후:\t', end='')
49 print_stack()
```

초기화

프로그램 3-1과 동일함

[프로그램 3-2]

```
Console  PyUnit
<terminated> linkedstack.py [C:₩Users₩sbyang₩AppData₩Local₩Programs₩Python₩Pyt
사과, 오렌지, 체리 push 후:   top ->  cherry -> orange -> apple
top 항목: cherry
배 push 후:                   top ->  pear -> cherry -> orange -> apple
pop(), 포도 push 후:          top ->  grape -> cherry -> orange -> apple
```

# 수행시간

- 파이썬의 리스트로 구현한 스택의 push와 pop 연산은 각각 O(1) 시간이 소요
- 파이썬의 리스트는 크기가 동적으로 확대 또는 축소되며, 이러한 크기 조절은 사용자도 모르게 수행된다. 이러한 동적 크기 조절은 스택(리스트)의 모든 항목들을 새 리스트로 복사해야 하기 때문에 O(N) 시간이 소요
- 단순연결리스트로 구현한 스택의 push와 pop 연산은 각각 O(1) 시간
  - 연결리스트의 맨 앞 부분에서 노드를 삽입하거나 삭제하기 때문

# 3.2 스택의 응용

- 컴파일러의 괄호 짝 맞추기
- 회문(Palindrome) 검사하기
- 후위표기법 수식 계산

# 스택 응용 : 괄호 검사

[핵심 아이디어] 왼쪽 괄호는 스택에 push, 오른쪽 괄호를 읽으면 pop 수행

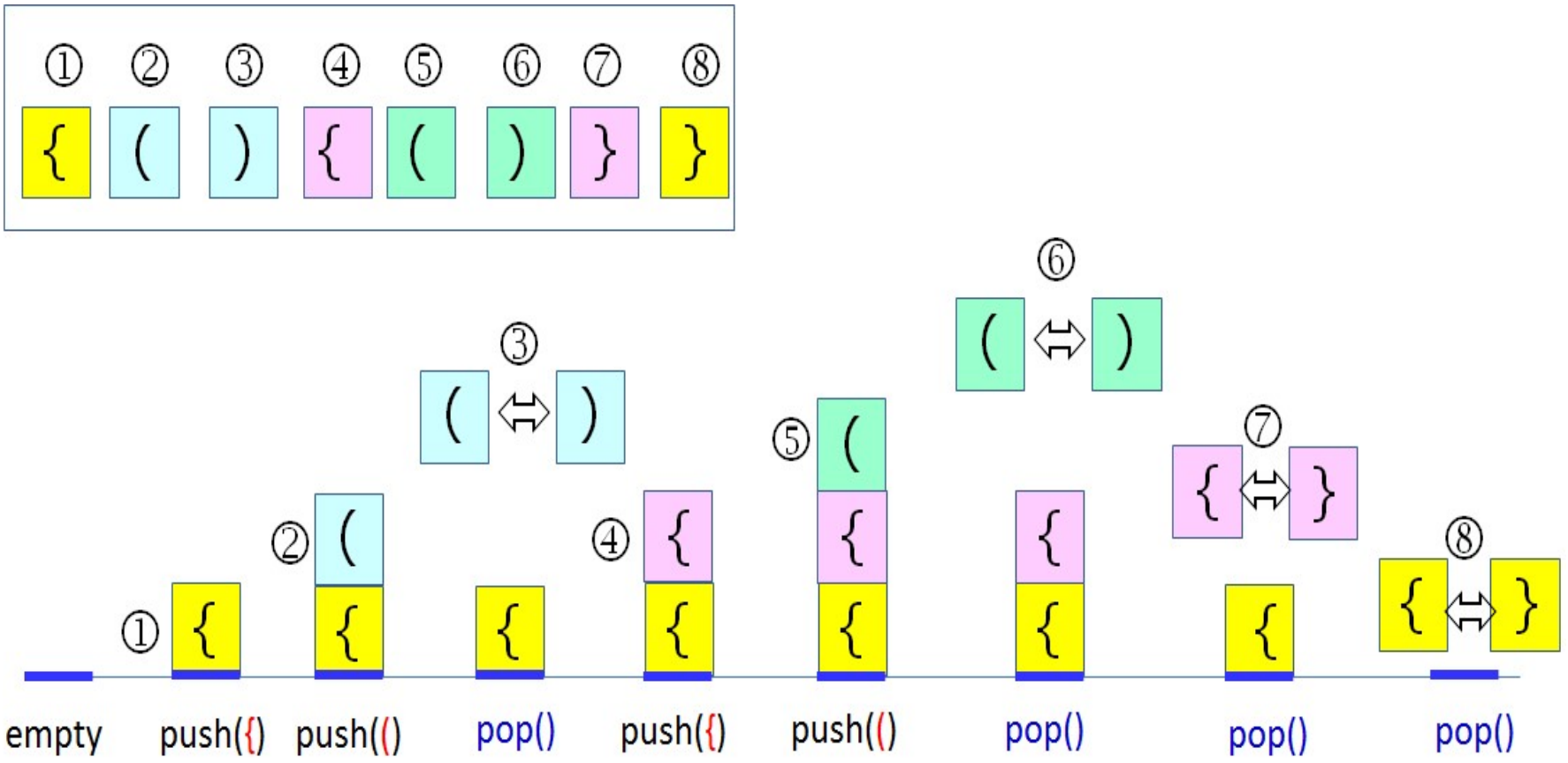
- pop된 왼쪽 괄호와 바로 읽었던 오른쪽 괄호가 다른 종류이면 에러 처리, 같은 종류이면 다음 괄호를 읽음
- 모든 괄호를 읽은 뒤 에러가 없고 스택이 empty이면, 괄호들이 정상적으로 사용된 것
- 만일 모든 괄호를 처리한 후 스택이 empty가 아니면 짝이 맞지 않는 괄호가 스택에 남은 것이므로 에러 처리

# 스택 응용 : 괄호 검사

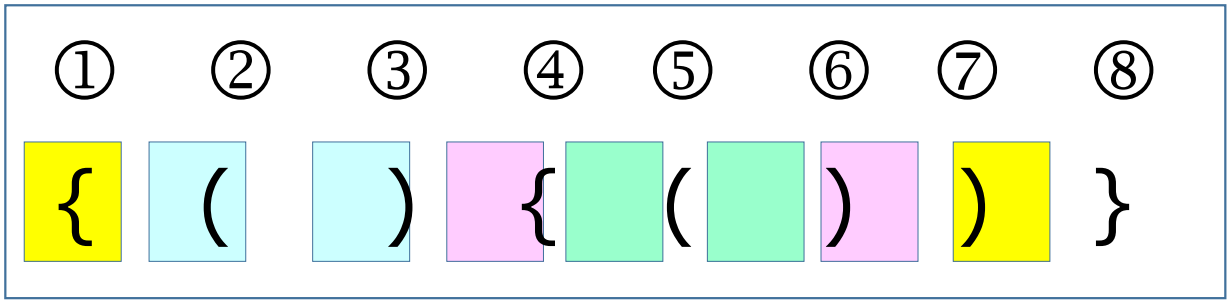
- 조건
  - 조건1 : 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야 한다.
  - 조건2 : 왼쪽 괄호는 오른쪽 괄호보다 먼저 나와야 한다.
  - 조건3 : 괄호 사이에는 포함 관계만 존재한다.

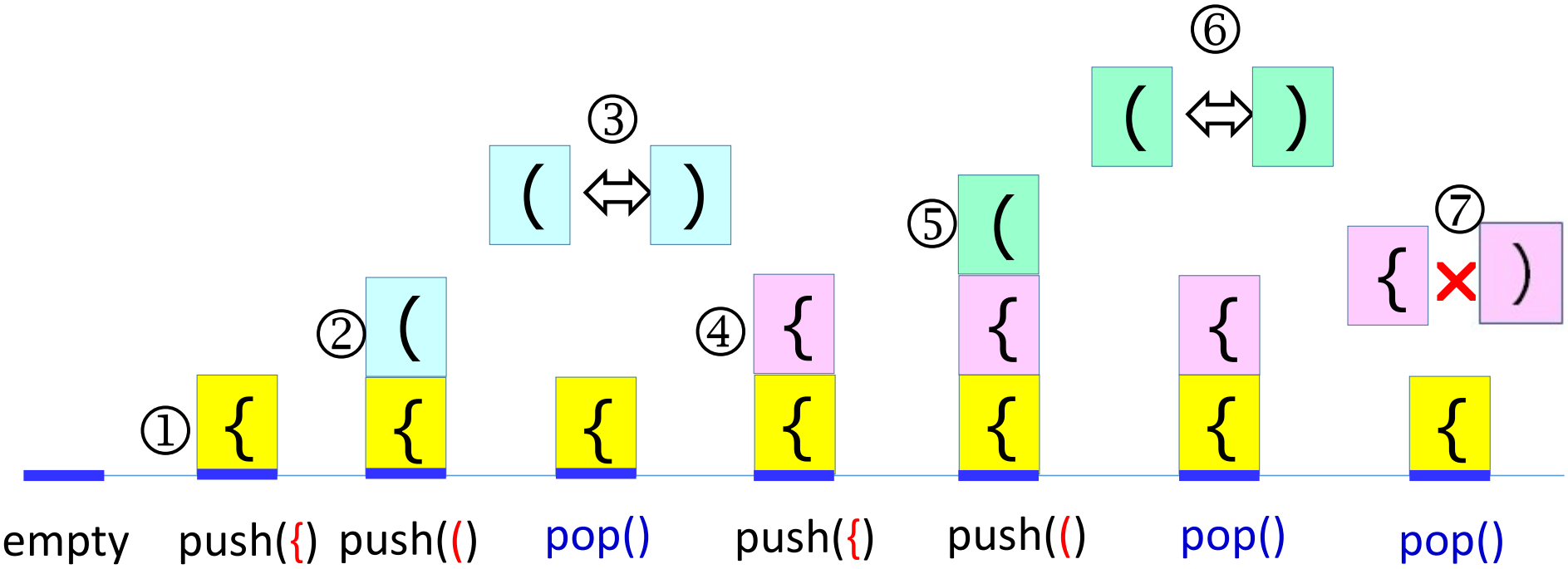
- 잘못된 괄호 사용의 예
  - (a(b)
  - a(b)c)
  - a{b(c[d]e}f)

# [예제 1]

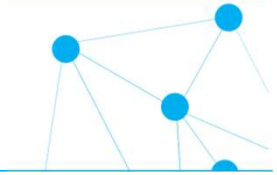


[예제 2]



# 스택의 구현(클래스 버전)



```
class Stack :
    def __init__( self ):                  # 생성자
        self.top = []                      # top이 클래스의 멤버 변수가 됨

    def isEmpty( self ): return len(self.top) == 0
    def size( self ): return len(self.top)
    def clear( self ): self.top = []

    def push( self, item ):
        self.top.append(item)

    def pop( self ):
        if not self.isEmpty():
            return self.top.pop(-1)

    def peek( self ):
        if not self.isEmpty():
            return self.top[-1]
```

# 괄호 검사 알고리즘

```
def checkBrackets(statement):
    stack = Stack()
    for ch in statement:                # 문자열의 각 문자에 대해
        if ch in ('{', '[', '('):       # in '{[('도 동일하게 동작함
            stack.push(ch)
        elif ch in ('}', ']', ')'):     # in '}])'도 동일하게 동작함
            if stack.isEmpty() :
                return False            # 조건 2 위반
            else :
                left = stack.pop()
                if (ch == "}" and left != "{") or \
                    (ch == "]" and left != "[") or \
                    (ch == ")" and left != "(") :
                    return False        # 조건 3 위반

    return stack.isEmpty()              # False이면 조건 1 위반
```

# 테스트 프로그램

```
str = ( "{ A[(i+1)] = 0; }", "if( (i==0) && (j==0 )", "A[ (i+1] ) = 0;" )
for s in str:
    m = checkBrackets(s)
    print(s," ---> ", m)
```

수고하셨습니다