6장. 스택

Contents

주요 내용

- 01 스택이란
- 02 리스트를 이용한 스택
- 03 연결 리스트를 이용한 스택
- 04 스택 응용

학습목표

- 스택의 의미를 이해한다.
- 스택의 추상적 구조를 이해한다.
- 리스트를 이용한 스택의 원리와 구현을 이해한다.
- 연결 리스트를 이용한 스택의 원리와 구현을 이해한다.
- 스택이 다른 클래스를 어떻게 재사용하는지 관찰한다.
- 간단한 스택의 파이썬 구현을 연습하고 이를 응용할 수 있도록 한다.

01 스택이란

스택 개념의 일상 예

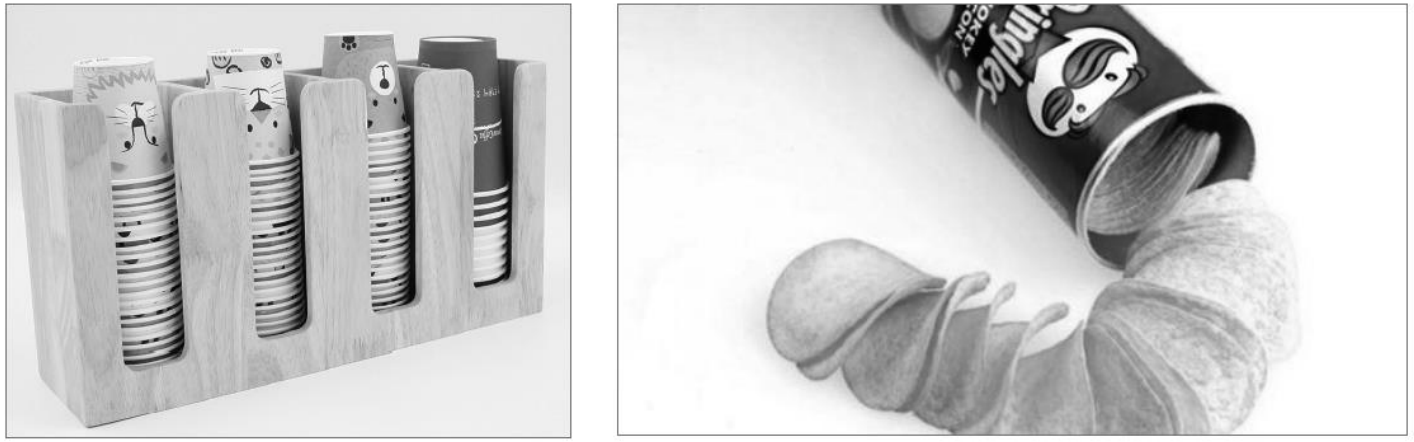


그림 6-1 스택 개념의 일상 예

식판의 스택

최근에 쌓은 식판을 꺼낸다

스택의 활용

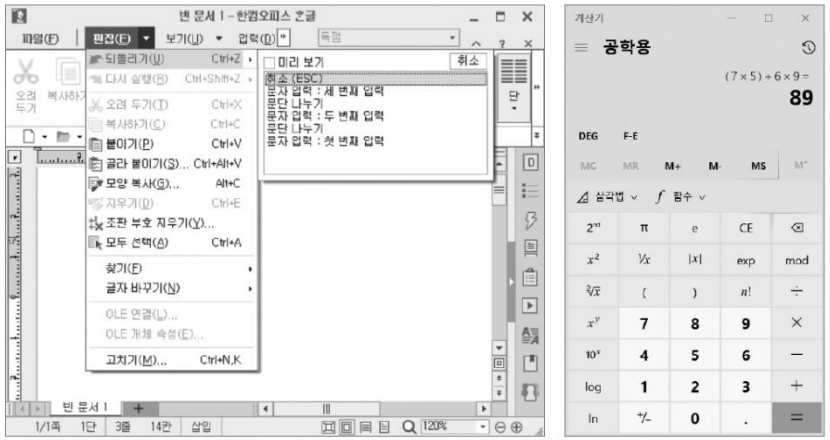


그림 6-2 스택의 활용(취소, 되돌리기)

- 키보드 입력을 하다가 백스페이스(←) 키를 누르면 최근에 입력한 글자를 지운다.
- 훈글, 워드, 파워포인트, 엑셀 등 모든 편집기는 최근에 한 작업순으로 취소하는 기능이 있다(보통 Ctrl+Z로 수행한다).
- 프로그램에서 함수 A가 함수 B를 호출하고 함수 B는 함수 C를 호출하는 호출 체인은 나중에 각 함수가 끝나면 돌아갈 때를 대비해서 호출 경로를 잘 관리해야 한다.

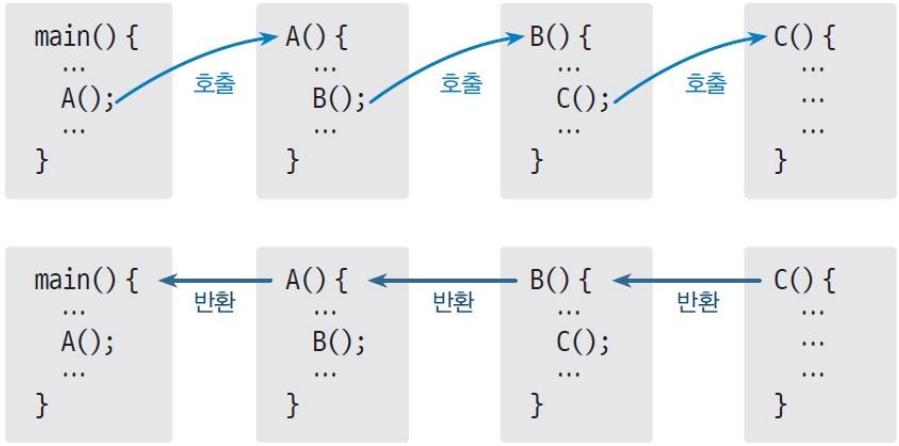
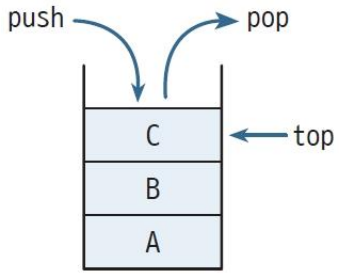


그림 6-7 함수 호출 시 스택 사용 예



'←' in keyboard input line

e.g., abcd←←efgh←←←ij←km←

결과: abeik

한 문자를 읽어 '←'이 아니면 저장하고

'←'이면 최근에 저장된 문자를 제거한다.

스택의 간단 개념

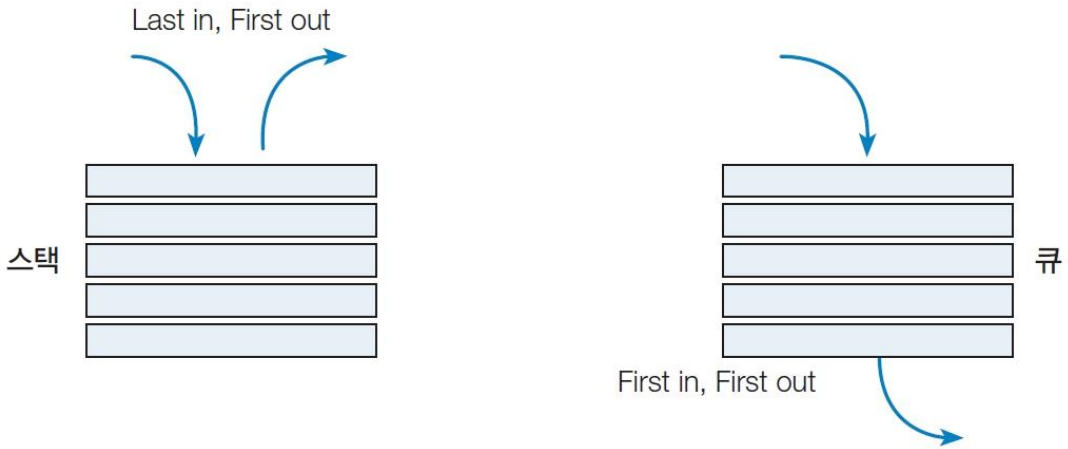


그림 6-3 스택과 큐 비교

그림 6-4 스택의 개념과 원리

스택의 삽입, 삭제 예

넣을 때는 위로 쌓아올리고,
뺄 때는 위에서부터 뺀다
맨 윗 원소의 내용만 알아볼 수 있다

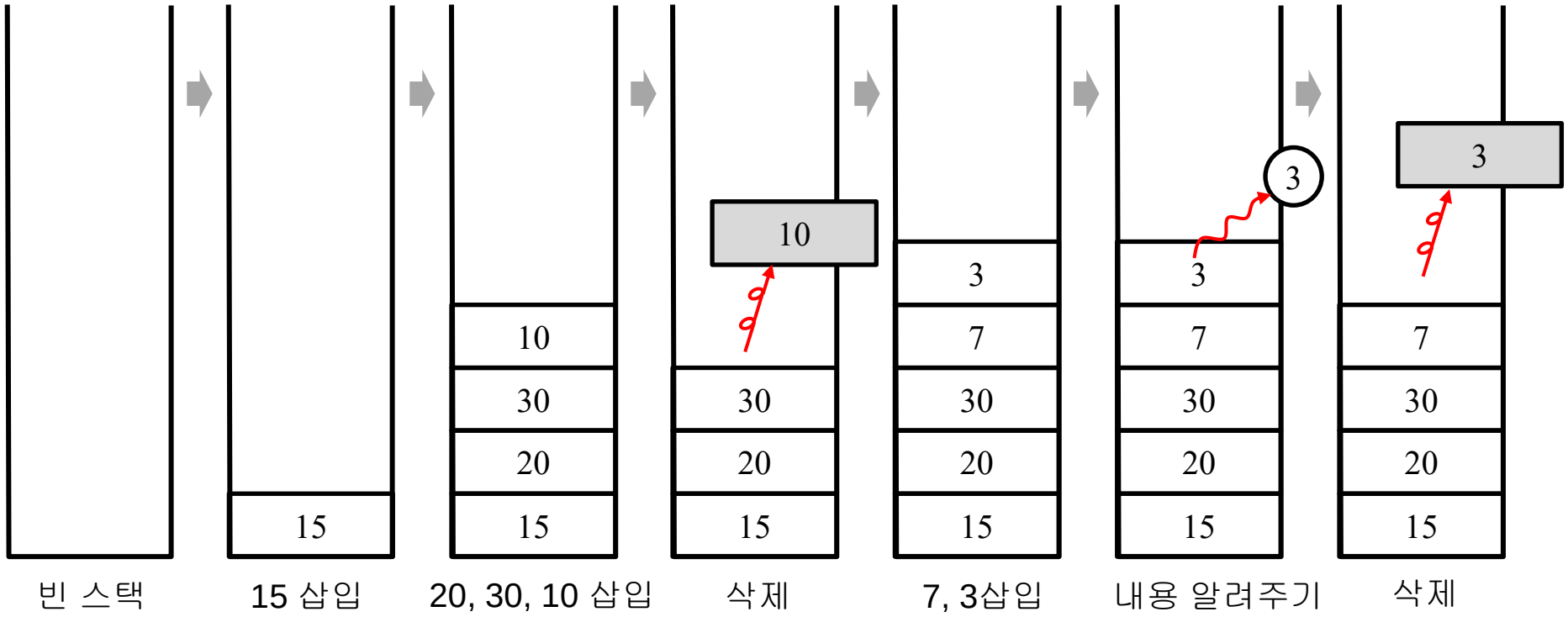
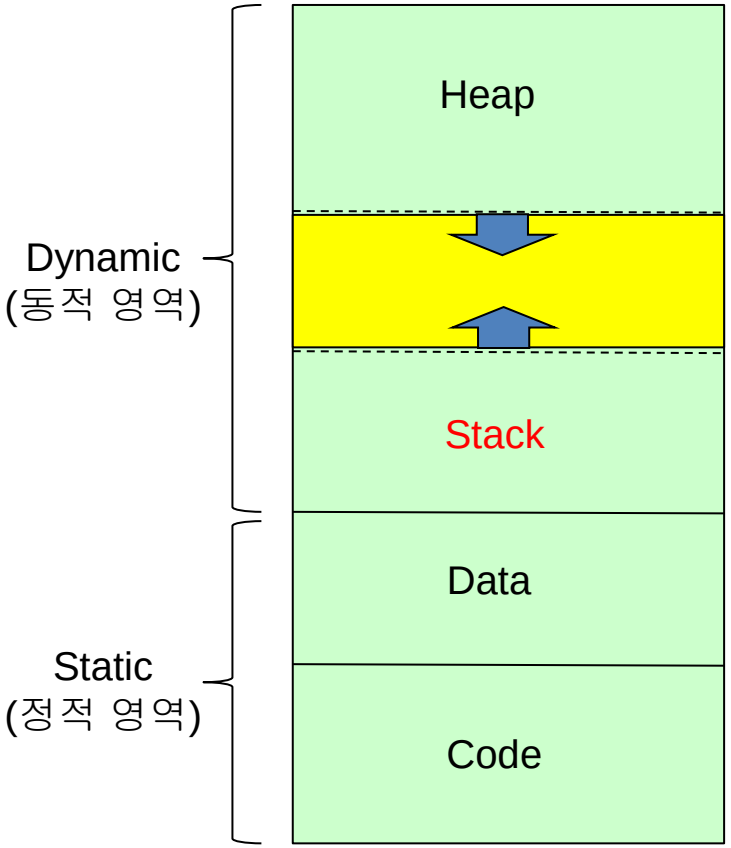


그림 6-5 스택에서 행하는 작업의 예

프로그램 수행과 스택 활용



수행중에 사용자가 할당받는 메모리
예: 자바의 new, C의 malloc(), 파이썬의 객체 생성

active 상태의 함수 관련 파라미터와 변수들

프로그램이 끝날 때까지 없어지지 않는 데이터.
전역 변수, 정적 변수

프로그램 수행 코드

그림 6-6 일반적인 가상 메모리 구조

ADT 스택

맨 윗부분에 원소를 추가한다
맨 윗부분에 있는 원소를 알려준다
맨 윗부분에 있는 원소를 삭제하면서 알려준다
스택이 비어 있는지 확인한다
스택을 깨끗이 비운다

그림 6-8 ADT 스택

유일하게 접근 가능한 원소는
최근에 삽입한 원소

LIFO(Last-In-First-Out)란 별칭을 갖고 있다

02 리스트를 이용한 스택

리스트 스택 객체 구조

필드:

__stack[] ◀ 스택 원소들이 저장되는 리스트

작업:

push() ◀ 스택의 맨 위에 원소 를 삽입한다
pop() ◀ 스택의 맨 위에 있는 원소를 알려주고 삭제한다
top() ◀ 스택의 맨 위에 있는 원소를 알려준다
isEmpty() ◀ 스택이 빈 스택인지를 알려준다
popAll() ◀ 스택을 깨끗이 청소한다

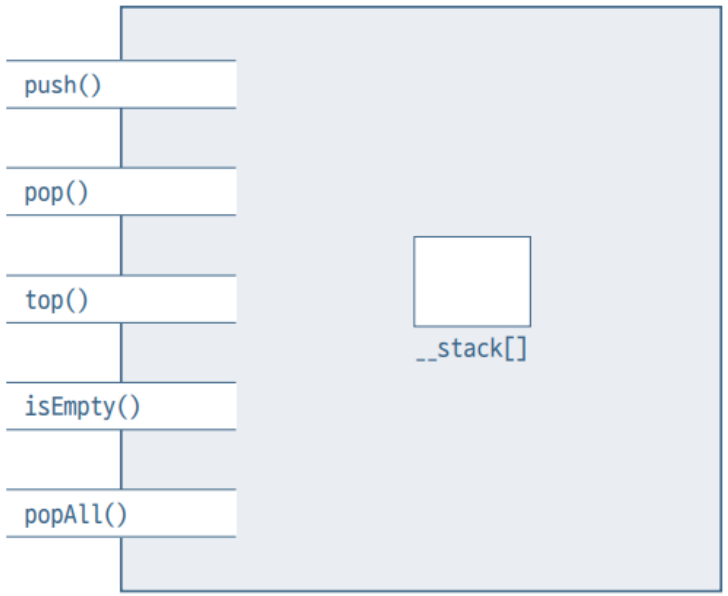


그림 6-9 리스트를 이용한 스택 객체 구조

삽입Insertion

코드 6-1 스택에 원소 삽입하기

```
def push(self, x):
    self.__stack.append(x)
```

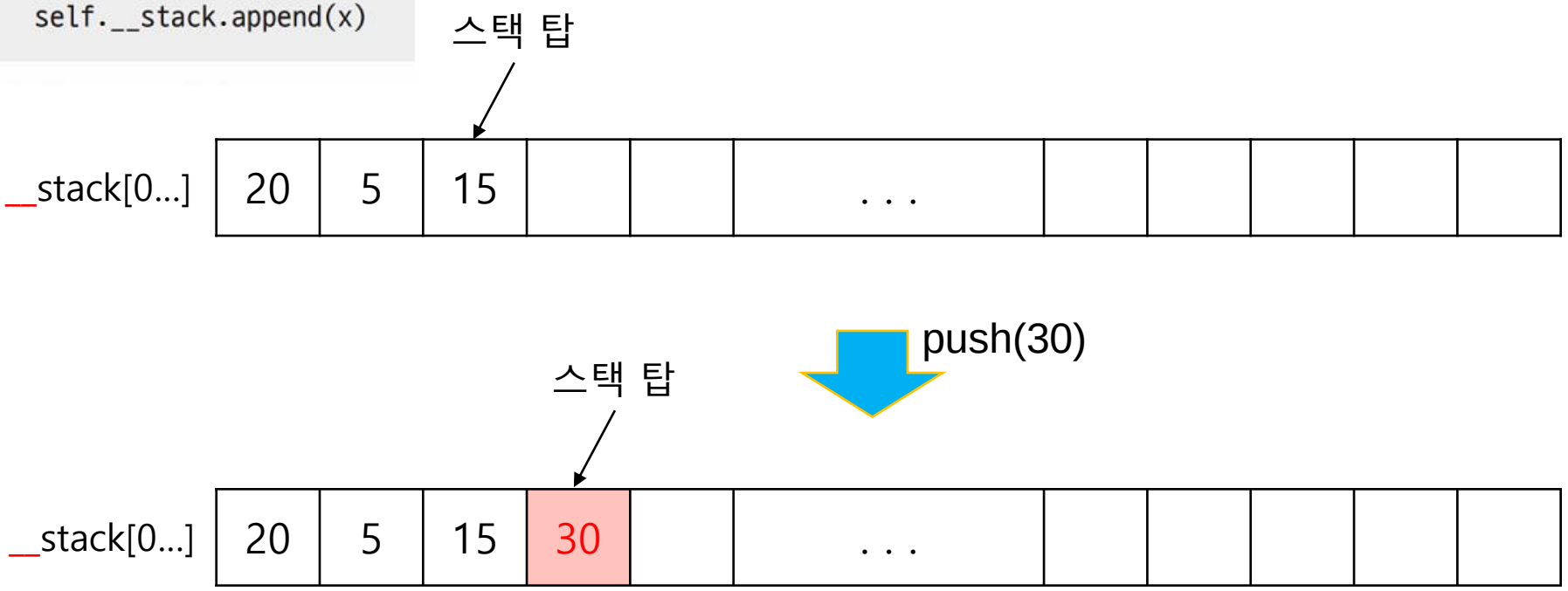


그림 6-11 리스트 스택에 원소를 삽입하는 예

삭제Deletion

코드 6-2 스택에서 원소 삭제하기

```
def pop(self):
    return self.__stack.pop()
```

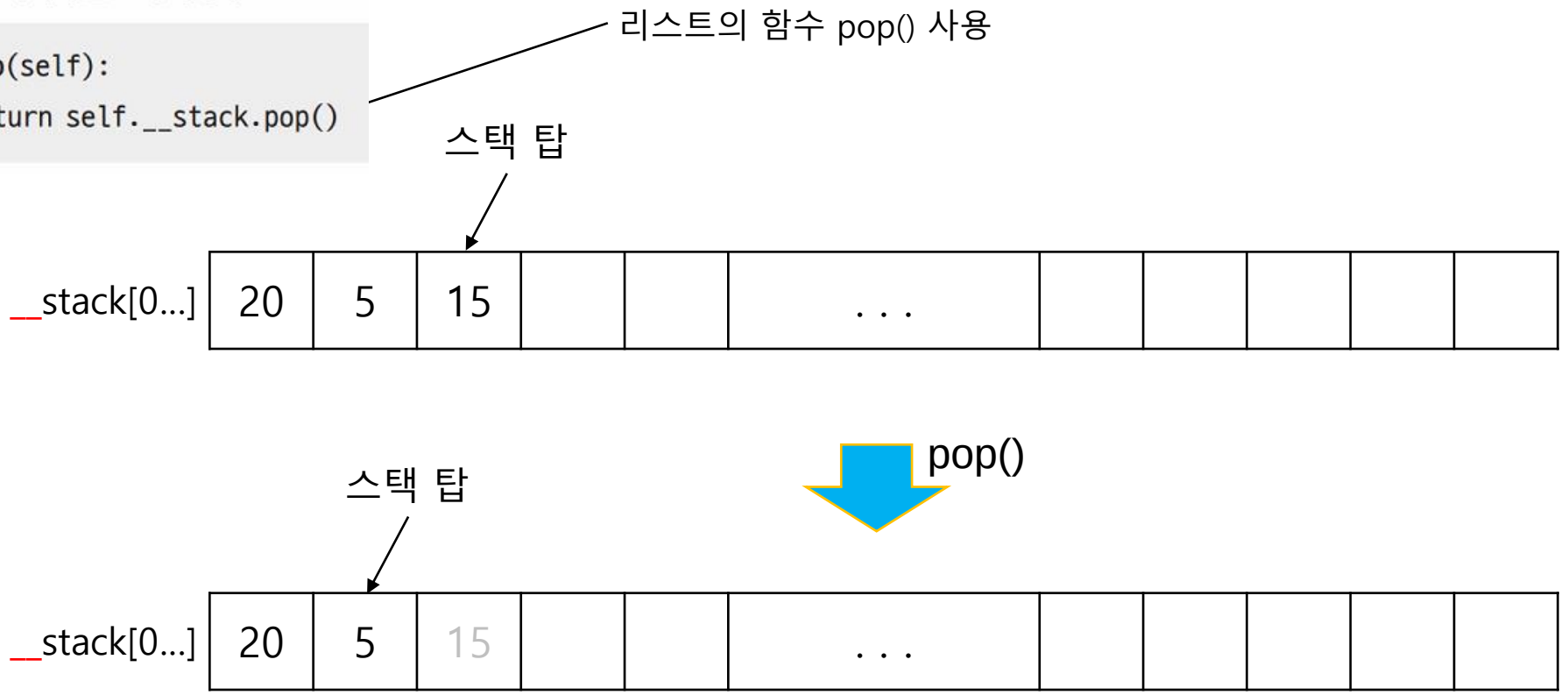


그림 6-12 배열 스택에서 원소를 삭제하는 예

기타 작업

top() `return stack[끝 인덱스]`

isEmpty() `return len(self.__stack)==0`

또는, `return not bool(self.__stack)`

```
def isEmpty(self) -> bool:
    return not bool(self.__stack)
```

popAll() `self.__stack.clear()`

또는, `self.__stack = []`

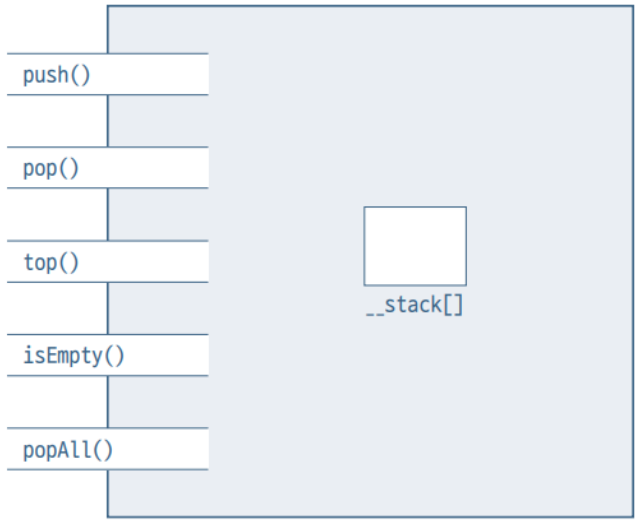
```
def popAll(self):
    self.__stack = []
```

리스트 스택 클래스 구조

```
class ListStack:
    def __init__(self):
        self.__stack = []
    def push(self, x):
    ...
    def pop(self):
    ...
    def top(self):
    ...
    def isEmpty(self):
    ...
    def popAll(self):
...
}
```

그림 6-10 리스트 스택 클래스 구조(listStack.py)

파이썬 구현



리스트 스택 객체 구조

```
class ListStack:
    def __init__(self):
        self.__stack = []

    def push(self, x):
        self.__stack.append(x)

    def pop(self):
        return self.__stack.pop()

    def top(self):
        if self.isEmpty():
            return None
        else:
            return self.__stack[-1]

    def isEmpty(self) -> bool:
        return not bool(self.__stack)
        # 또는 return len(self.__stack) == 0

    def popAll(self):
        self.__stack.clear()

    def printStack(self):
        print("Stack from top:", end=' ')
        for i in range(len(self.__stack) -1, -1, -1):
            print(self.__stack[i], end=' ')
        print()
```

클래스 ListStack 사용 예

리스트 스택 사용 예

```
st1 = ListStack()
print(st1.top()) # No effect
st1.push(100)
st1.push(200)
print("Top is", st1.top())
st1.pop()
st1.push('Monday')
st1.printStack()
print('isEmpty?', st1.isEmpty())
```

수행 결과

```
Top is 200
Stack from top: Monday 100
isEmpty? False
```

03 연결 리스트를 이용한 스택

연결 리스트 스택 객체 구조

필드:
__list ◀ 스택 원소들이 저장되는 연결 리스트
작업:
push() ◀ 스택의 맨 위에 원소 를 삽입한다
pop() ◀ 스택의 맨 위에 있는 원소를 알려주고 삭제한다
top() ◀ 스택의 맨 위에 있는 원소를 알려준다
isEmpty() ◀ 스택이 빈 스택인지를 알려준다
popAll() ◀ 스택을 깨끗이 청소한다

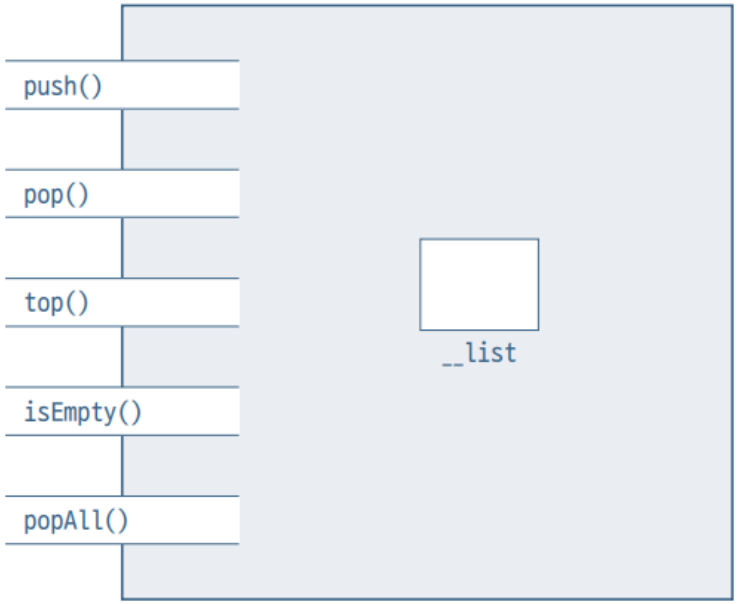


그림 6-14 연결 리스트 스택의 객체 구조

연결 리스트 스택 클래스 구조

```
class LinkedStack:
    def __init__(self):
        self.__list = LinkedListBasic()
    def push(self, x):
        ...
    def pop(self):
        ...
    def top(self):
        ...
    def isEmpty(self):
        ...
    def popAll(self):
        ...
    ...
```

그림 6-15 연결 리스트 스택 클래스 구조(linkedStack.py)

삽입Insertion

__list 대신 list를 문맥에 따라 혼용

핵심 작업

코드 6-8 스택에 원소 x 삽입하기

```
def push(self, x):
    self.__list.insert(0, x)
```

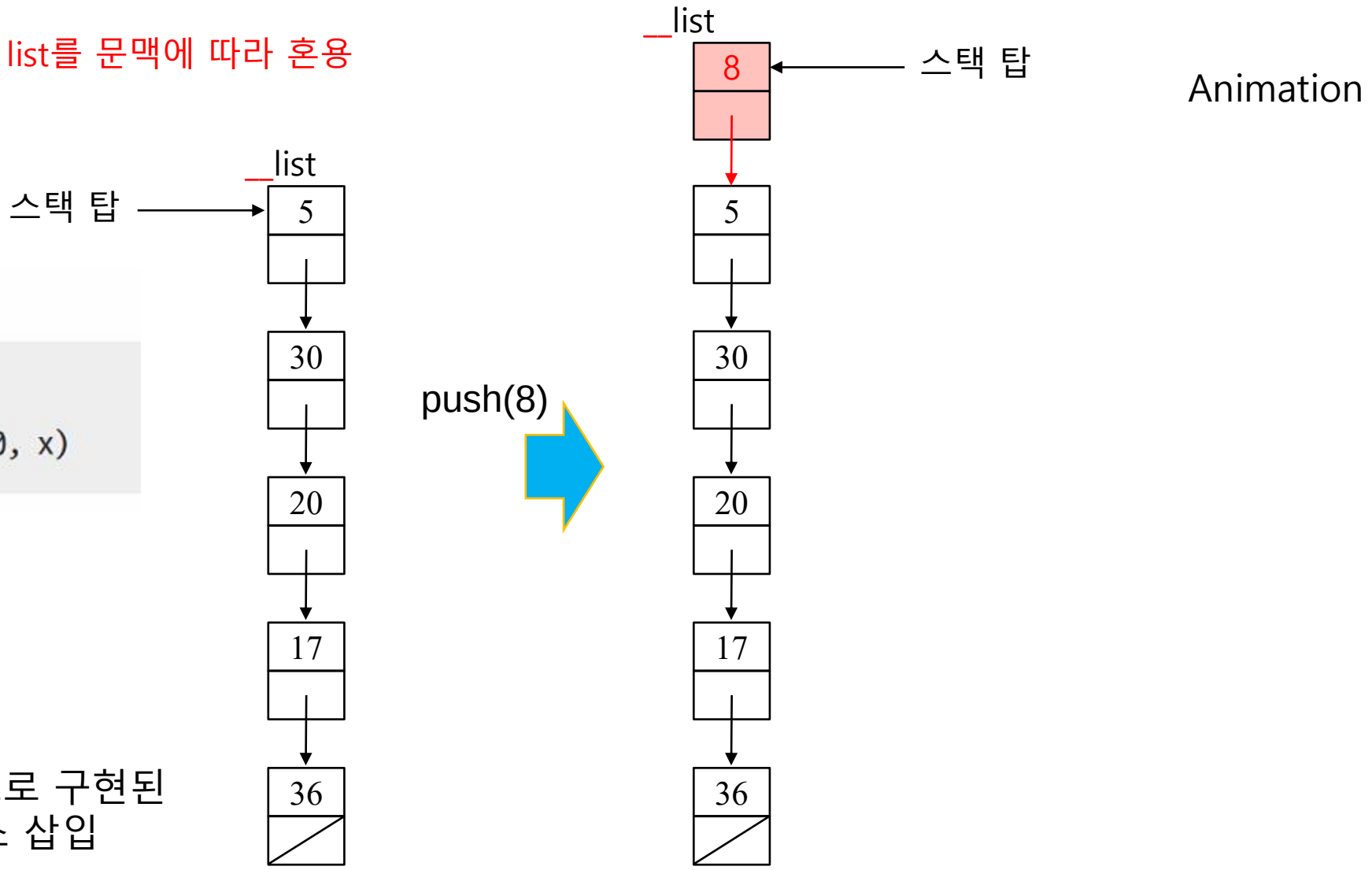


그림 6-16 연결 리스트로 구현된 스택의 원소 삽입

Animation

삭제Deletion

핵심 작업

코드 6-9 스택에서 원소 삭제하기

```
def pop(self):
    return self.__list.pop(0)
```

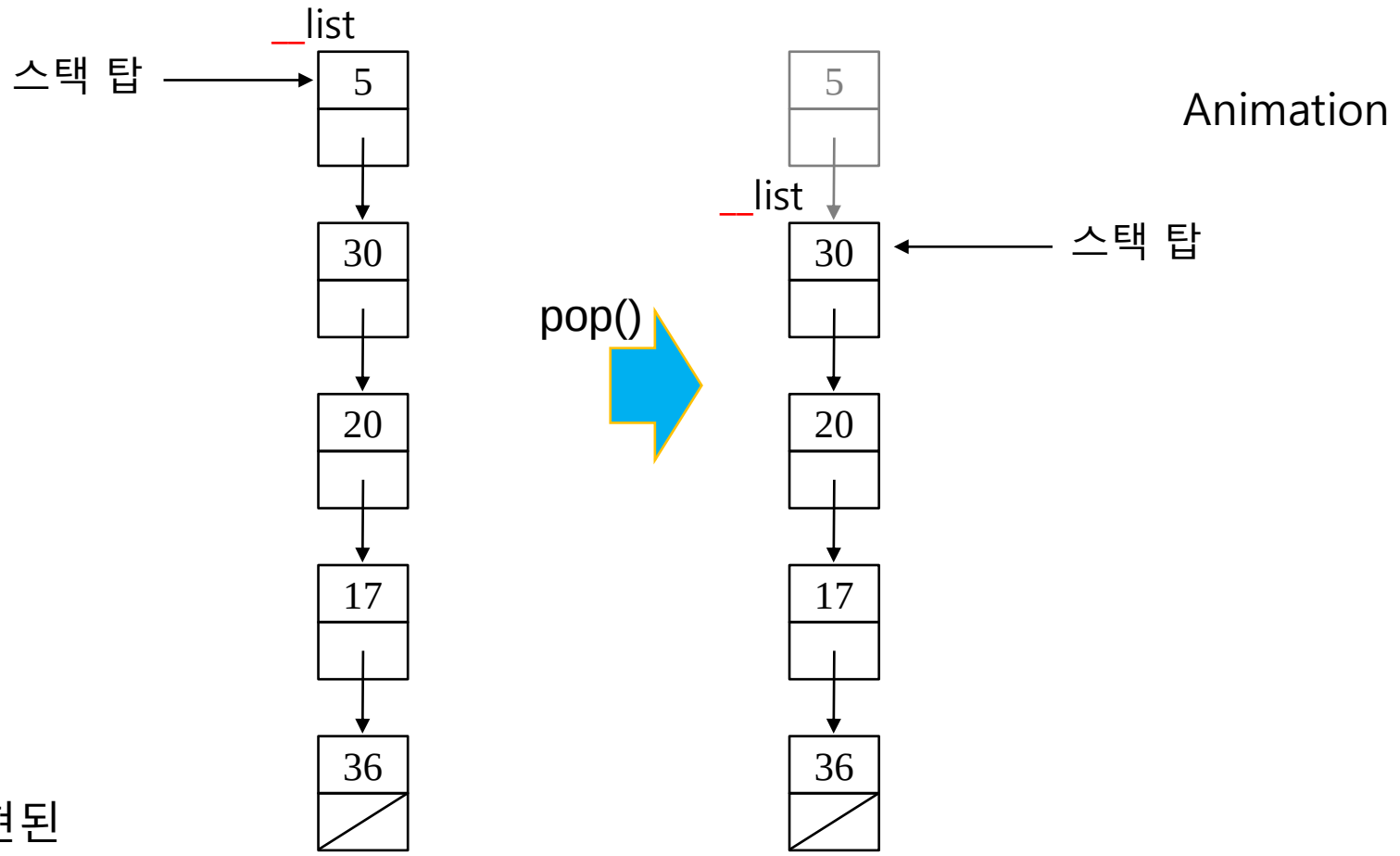


그림 6-17 연결 리스트로 구현된 스택의 원소 삭제

기타 작업

top() `return __list.get(0)`

코드 6-10 스택 탑 원소 알려주기

```
def top(self):
    if self.isEmpty():
        return None
    else:
        return self.__list.get(0)
```

isEmpty() `return __list.isEmpty()`

코드 6-11 스택이 비었는지 확인하기

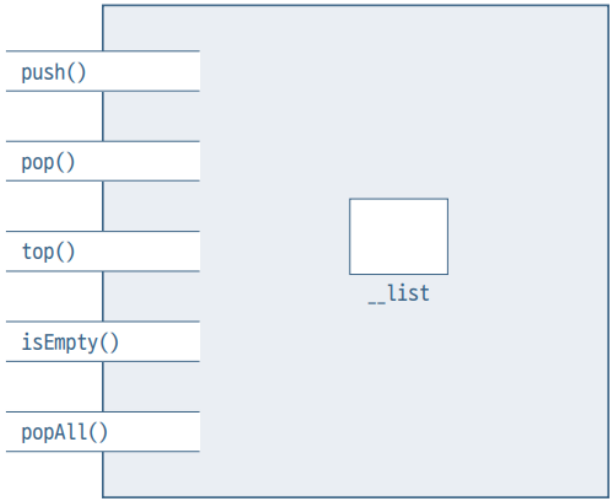
```
def isEmpty(self) -> bool:
    return self.__list.isEmpty()
```

popAll() `__list.clear()`

코드 6-12 스택 비우기

```
def popAll(self):
    self.__list.clear()
```

파이썬 구현



연결 리스트 스택 객체 구조

```
class LinkedStack:
    def __init__(self):
        self.__list = LinkedListBasic()

    def push(self, newItem):
        self.__list.insert(0, newItem)

    def pop(self):
        return self.__list.pop(0)

    def top(self):
        return self.__list.get(0)

    def isEmpty(self) -> bool:
        return self.__list.isEmpty()

    def popAll(self):
        self.__list.clear()

    def printStack(self):
        print("Stack from top:", end=' ')
        for i in range(self.__list.size()):
            print(self.__list.get(i), end=' ')
        print()
```

연결 리스트 스택 사용 예

```
st1 = linkedStack()
st1.push(100)
st1.push(200)
print("Top is", st1.top())
st1.pop()
st1.push('Monday')
st1.printStack()
print('isEmpty?', st1.isEmpty())
```

수행 결과

```
Top is 200
Stack from top: Monday 100
isEmpty? False
```

04 스택 응용

문자열 뒤집기

```
def reverse(str):
        st = ListStack()
        for i in range(len(str)):
                st.push(str[i])
        out = ""
        while not st.isEmpty():
                out += st.pop()
        return out

def main():
        input = "Test Seq 12345"  # 테스트 입력 문자열
        answer = reverse(input)
        print("Input string: ", input)
        print("Reversed string: ", answer)

if __name__ == "__main__":
        main()
```

✓main()과 __name__ 사용법은 p.83~85 참조

출력

```
Input string: Test Seq 12345
Answer: 54321 qeS tseT
```

Postfix 계산

1. 문자가 숫자이면 push()를 통해 스택에 삽입한다. 단, 바로 앞의 문자가 숫자이면 연속된 숫자로서 수 하나에 속하므로 스택 탑에 저장된 수를 뽑아서(pop()) 합쳐 계산한 다음 스택에 다시 넣어준다(push()). 예를 들어, 스택 탑에 25가 저장되어 있고 현재 숫자가 3이면 앞부분의 25는 253의 25이므로 25*10+3으로 계산해줘야 한다. 이런 식으로 수가 끝나는 공백을 만날 때까지 스택 탑에 있는 수를 뽑아서(pop()) 계산한 다음 결과를 스택에 넣어준다(push()).
2. 문자가 연산자이면, 스택의 맨 위에 있는 수 2개를 뽑아서(pop() 2번) 계산한 다음 결과를 스택에 삽입한다(push()).
3. 문자가 공백이면 그냥 무시하고 넘어간다.

```
def evaluate(p):
        s = ListStack()
        digitPreviously = False
        for i in range(len(p)):
                ch = p[i]  # i번 문자. 번호는 0번부터.
                if ch.isdigit():    # ch가 숫자
                        if digitPreviously:
                                tmp = s.pop()
                                tmp = 10 * tmp + (ord(ch) - ord('0'))
                                s.push(tmp)
                        else:
                                s.push(ord(ch) - ord('0'))
                                digitPreviously = True
                elif isOperator(ch): # ch가 연산자
                        s.push(operation(s.pop(), s.pop(), ch))
                        digitPreviously = False
                else:       # ch가 공백
                        digitPreviously = False
        return s.pop()

def isOperator(ch) -> bool:  # 연산자인가?
        return (ch == '+' or ch == '-' or ch == '*' or ch == '/')

def operation(opr2:int, opr1:int, ch) -> int:  # 연산하기
        return {'+': opr1 + opr2, '-': opr1 - opr2, '*': opr1 * opr2, '/': opr1 // opr2}[ch]

def main():
        postfix = "700 3 47 + 6 * - 4 /"  # 테스트 샘플 입력(후위 표현식)
        print("Input string: ", postfix);
        answer = evaluate(postfix)
        print("Answer: ", answer)
        print(ord('0'), ord('9'))

if __name__ == "__main__":
        main()
```

출력

```
Input string: 700 3 47 + 6 * - 4 /
Answer: 100
```