ADT Stack [ADT 7-1]

데이터 : 0개 이상의 원소를 가진 유한 순서 리스트

연산 :
S ∈ Stack; item ∈ Element;
createStack() ::= create an empty Stack;
// 공백 스택을 생성하는 연산
isEmpty(S) ::= **if** (S is empty) **then return true**
else return false;
// 스택 S가 공백인지 아닌지를 확인하는 연산
push(S, item) ::=insert item onto the top of S;
// 스택 S의 top에 item(원소)을 삽입하는 연산
pop(S) ::= **if** (isEmpty(S)) **then return** error
else { delete and **return** the top item of S };
// 스택 S의 top에 있는 item(원소)을 스택에서 삭제하고 반환하는 연산
delete(S) ::= **if** (isEmpty(S)) **then return** error
else delete the top item;a
// 스택 S의 top에 있는 item(원소)을 삭제하는 연산
peek(S) ::= **if** (isEmpty(S)) **then return** error
else return (the top item of the S);
// 스택 S의 top에 있는 item(원소)을 반환하는 연산

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

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

    def pop(self):
        if self.isEmpty() == False:
            return self.s.pop(-1)
        else:
            return None

    def peek(self):
        if self.isEmpty() == False:
            return self.s[-1]
        else:
            return None

    def isEmpty(self):
        if len(self.s) > 0:
            return False
        else:
            return True

    def size(self):
        return len(self.s)

    def print(self):
        print(self.s)

s = Stack()
s.push('a')
s.push('b')
s.push('c')
print(s.peek())
s.print()
s.isEmpty()
s.size()
```

빈 list를 활용해서 stack 자료구조를 만든다.

맨 끝에 위치한 원소의 인덱스.

맨 끝에 위치한 원소의 값을 반환함.

self.s 내 원소의 갯수가 0 이면, 해당 리스트는 비어있는 것임.

중요한 전제!!

```
# 수식 괄호 체크 : 수행해서 '(' 이면 push, ')'이면 pop 하여 Cancel
# 수식이 끝났을 때, 스택이 비어 있으면 괄호가 정확한 것임

def exBracketCheck(f):
    s = Stack()
    for i in range(len(f)):
        if f[i] == '(':
            s.push(f[i])
        elif f[i] == ')':
            if s.pop() == None: return False
    if s.isEmpty() == True:
        return True
    else:
        return False

f = '(1+2) + (34) - '
exBracketCheck(f)
```

```
True
```

Stack 자료구조에는 여는 괄호만 들어감.

elif f[i] == ")":
    s.pop()
    if s.pop() == None:
        return False

stack에 여는 괄호가 남아있으면 false를 반환함.

수식 후위표기 알고리즘

- 수식을 컴퓨터가 쉽게 계산할 수 있도록 바꾸는 알고리즘이다.

수식 후위표기 알고리즘 → 괄호가 필요없는 회법이다. 즉, 수식후위표기법으로 나타내진 식에는 괄호가 없다.

- 수식을 컴퓨터가 쉽게 계산할 수 있도록 바꾸는 알고리즘이다.
- 알고리즘을 수행하기 전에 빈 스택과 빈 리스트를 준비한다. → 숫자와 연산기호들이 들어가는 자료구조.

여는 괄호가 들어가는 자료구조

중위표기 수식의 원소를 순차적으로 읽으면서 아래의 작업을 수행

① "(": push
② ")" :스택에서 "("를 만나기 바로 직전까지 pop하여 문자열에 추가
"("를 만나면 pop
③ "+", "-": peek하여 연산자이면 pop하여 문자열에 추가를 반복
반복 중 연산자가 아니면 반복을 취소하고 원소를 push
④ "*","/": peek하여 "*","/"이면 문자열에 추가를 반복, 아니면 반복을 취소하고 원소를 push
⑤ 숫자: 문자열에 추가
⑥ 수식이 끝나면, 스택이 공백이 될 때까지 pop하여 문자열에 추가

```
# 연산부 여부 리턴
def isOper(item):

    if item == '+' or item == '-' or item == '*' or item == '/':
        return True
    else:
        return False

isOper('+')

# 주어진 문자가 숫자인지 여부 리턴
def isNum(s):
    try:
        float(s)
        return True
    except ValueError:
        return False
```

```
eq = "( 12.3 + 6 ) * 3 / 6"
eq_list = eq.split(" ")
print(eq_list)

s = Stack()
postEq = []

for item in eq_list:
    if item == '(':
        s.push(item)
    elif item == ')':
        while True:
            _tap = s.pop()
            if _tap != '(':
                postEq.append(_tap)
            else:
                break
    elif (item == '*') or (item == '-'):
        while isOper(s.peek()) == True:
            postEq.append(s.pop())
        s.push(item)
    elif ((item == '*') or (item == '/')):
        while (s.peek() == '*') or (s.peek() == '/'):
            postEq.append(s.pop())
        s.push(item)
    elif isNum(item) == True:
        postEq.append(item)

while s.isEmpty() != True:
    postEq.append(s.pop())

print(postEq)
```

```
['(', '12.3', '+', '6', ')', '*', '3', '/', '6']
['12.3', '6', '+', '3', '*', '6', '/']
```