4주차 코드

```
# 스택이란
# 스택(stack) : 쌓아 놓은 더미
# 후입선출(LIFO) : 가장 최근에 들어온 데이터가 가장 먼저 나감

# 스택의 ADT(추상자료형)
# Stack() : 비어있는 새로운 스택을 만든다.
# isEmpty() : 스택이 비어있으면 T, 아니면 F를 반환한다.
# push() : 항목 e를 스택의 맨 위에 추가한다.
# pop() : 스택의 맨 위에 있는 항목을 꺼내 반환한다.
# peek() : 스택의 맨 위에 있는 항목을 삭제하지 않고 반환한다.
# size() : 스택 내의 모든 항목들의 개수를 반환한다.
# clear() : 스택을 공백 상태로 만든다.

# 스택의 용도
# 되돌리기
# 함수 호출
# 괄호검사
# 계산기 : 후위표시식 계산, 중위 표기식의 후위 표기식 변환
# 미로 탐색

# 스택의 구현
# 항목의 삽입/ 삭제 위치 : 파이썬 리스트의 후단을 사용하는 경우가 유리하다.

# 스택의 함수 구현
def isEmpty():
    return len(top) == 0  # 계산 결과가 True/False로 반환이 된다.

def push(item):
    top.append(item)  # 리스트 맨뒤에 item을 추가한다.

def pop():
    if not isEmpty(): # 공백상태가 아니면
        return top.pop(-1) # 리스트의 맨 뒤에서 항목을 하나 꺼내고 반환

def peek():
    if not isEmpty(): # 공백상태가 아니면
        return top[-1]  # 맨 뒷 항목을 반환하고 삭제하지 않는다.

def size():
    return len(top) # 스택의 크기

def clear():
    global top # top는 전역변수 임을 지정함
    top = [] # 스택의 초기화

# 본문
if __name__ == "__main__":
    top = []

    for i in range(1, 6):
        push(i)

    print(' push 5회: ', top) # [1, 2, 3, 4, 5]
    print(' pop() --> ', pop()) # 5
    print(' pop() --> ', pop()) # 4
```

```
    print(' pop 2회: ', top) # [1, 2, 3]

    push('홍길동')
    push('이순신')
    print(' pop() --> ', pop()) # 이순신
    print(' pop 1회: ', top) # [1, 2, 3, '홍길동']

'''
 push 5회:  [1, 2, 3, 4, 5]
 pop() -->  5
 pop() -->  4
 pop 2회:  [1, 2, 3]
 pop() -->  이순신
 pop 1회:  [1, 2, 3, '홍길동']
'''
```

```
# 스택의 구현
# 항목의 삽입/ 삭제 위치 : 파이썬 리스트의 후단을 사용하는 경우가 유리하다.

# 스택의 클래스 구현
class Stack:
    def __init__(self):
        self.top = []

    def isEmpty(self):
        return len(self.top) == 0    # 계산 결과가 True/False로 반환이 된다.

    def size(self):
        return len(self.top)  # 스택의 크기를 반환

    def clear(self):
        self.top = []

    def push(self, item):
        self.top.append(item)  # 리스트 맨뒤에 item을 추가한다.

    def pop(self):
        if not self.isEmpty(): # 공백상태가 아니면
            return self.top.pop(-1) # 리스트의 맨 뒤에서 항목을 하나 꺼내고 반환

    def peek(self):
        if not self.isEmpty():  # 공백상태가 아니면
            return self.top[-1]  # 맨 뒷 항목을 반환하고 삭제하지 않는다.

    def __str__(self): # 프린트가 되었을때 형식 지정, 이게 없을때는 <>가 있는 부분이 출력됨.
        return str(self.top[::-1]) # 순서를 역순으로 하여 stack의 성질로 출력하게 함
```

```
# 본문
if __name__ == "__main__":

    odd = Stack()
    even = Stack()

    for i in range(10):
        if i%2 == 0:
            even.push(i)
        else:
            odd.push(i)

    print(' 스택 even push 5회: ', even) # [0, 2, 4, 6, 8]
    print(' 스택 odd push 5회: ', odd) # [1, 3, 5, 7, 9]
    print(' 스택 even      peek: ', even.peek()) # 8
    print(' 스택 odd      peek: ', odd.peek()) # 9

    for _ in range(2):
        even.pop()
    for _ in range(3):
        odd.pop()

    print(' 스택 even pop 2회: ', even) # [4, 2, 0]
    print(' 스택 odd pop 3회: ', odd) # [3, 1]

''' __str__설정을 하지 않고, even, odd로 출력하는 경우
 스택 even push 5회:  <__main__.Stack object at 0x000002AAC9811ED0> # 스택의 내용이 아니라 객체의 정보가 출력
 스택 odd push 5회:  <__main__.Stack object at 0x000002AAC9811D50> # 스택의 내용이 아니라 객체의 정보가 출력
 스택 even      peek:  8
 스택 odd      peek:  9
 스택 even pop 2회:  <__main__.Stack object at 0x000002AAC9811ED0> # 스택의 내용이 아니라 객체의 정보가 출력
 스택 odd pop 3회:  <__main__.Stack object at 0x000002AAC9811D50> # 스택의 내용이 아니라 객체의 정보가 출력
'''
''' __str__설정을 하지 않고, even.top, odd.top로 출력하는 경우
 스택 even push 5회:  [0, 2, 4, 6, 8]
 스택 odd push 5회:  [1, 3, 5, 7, 9]
 스택 even      peek:  8
 스택 odd      peek:  9
 스택 even pop 2회:  [0, 2, 4]
 스택 odd pop 3회:  [1, 3]
'''
''' __str__설정을 하고, even, odd로 출력하는 경우
 스택 even push 5회:  [8, 6, 4, 2, 0]
 스택 odd push 5회:  [9, 7, 5, 3, 1]
 스택 even      peek:  8
 스택 odd      peek:  9
 스택 even pop 2회:  [4, 2, 0]
 스택 odd pop 3회:  [3, 1]
'''
```

```
# 스택의 용도 : 괄호검사

# 괄호검사 알고리즘
# 조건 1 : 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야 한다.
# 조건 2 : 같은 타입의 괄호에서 외쪽 괄호가 오른쪽 괄호보다 먼저 나와야 한다.
# 조건 3 : 서로 다른 타입의 괄호 쌍이 서로를 교차하면 안된다.

from Stack_class import Stack

def checkBrackets(statement): # 한 문장에서 검사 함수
    stack = Stack()

    for ch in statement:
        if ch in ('{', '[', '('): # 왼쪽 괄호이면
            stack.push(ch) # 대입
        elif ch in ('}', ']', ')'): # 오른족 괄호이면
            if stack.isEmpty(): # 왼쪽 괄호가 이미 저장되어있나 확인.
                return False # 없으면 조건 2 위반
            else: # 값(왼쪽 괄호)이 있으면
                left = stack.pop() # 꺼내와서
                if (ch == "}" and left != "{") or (ch == "]" and left != "[")  or (ch == ")" and left != "("):
                    return False # 괄호의 짝이 맞지 않으면 조건 3 위반
    return stack.isEmpty() # False이면, 즉 오른쪽 부분이 없는데 왼쪽부분이 스택에 남아 있다면 조건 1 위반

def checkBracketsVer2(lines): # 여러 문장에서 검사 함수
    stack = Stack()
    for line in lines: # 전체 라인을 불러오고
        for ch in line: # 각 라인을 따로 불러서 확인한다.
            if ch in ('{', '[', '('): # 왼쪽 괄호이면
                stack.push(ch) # 대입
            elif ch in ('}', ']', ')'): # 오른족 괄호이면
                if stack.isEmpty(): # 왼쪽 괄호가 이미 저장되어있나 확인.
                    return False # 없으면 조건 2 위반
                else: # 값(왼쪽 괄호)이 있으면
                    left = stack.pop() # 꺼내와서
                    if (ch == "}" and left != "{") or (ch == "]" and left != "[")  or (ch == ")" and left != "("):
                            return False # 괄호의 짝이 맞지 않으면 조건 3 위반
    return stack.isEmpty() # False이면, 즉 오른쪽 부분이 없는데 왼쪽부분이 스택에 남아 있다면 조건 1 위반

# 본문
if __name__ == "__main__":

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

    # filename = "Stack_class.py"
    filename = "test.txt" # 괄호가 부족함. // test.txt -> [{( )]
    infile = open(filename, "r", encoding='UTF8')
    lines = infile.readlines();
    infile.close()

    result = checkBracketsVer2(lines)
    print("\n", filename, " ---> ", result) # test.txt  --->  False
```

```
'''
{ A[(i+1)] = 0; } --->  True
if( (i==0) && (j==0) --->  False
A[ (i+1] ) = 0; --->  False

 test.txt  --->  False
'''
```

```
# 스택의 응용 : 수식의 계산
# 스택을 이용한 후위표기 수식의 계산
# 스택을 이용한 중위표기 수식의 후위표기 변환

# 수식의 표기 방법 3가지
# 전위 : 연산자 피연산자1 피연산자2
# 중위 : 피연산자 연산자 피연산자2
# 후위 : 피연산자1 피연산자2 연산자

# 후위표기 수식의 알고리즘
from Stack_class import Stack

def evalPostfix( expr ):
    s = Stack() # s.top = [] 객체 생성

    for token in expr: # 리스트의 모든 항목에 대해
        if token in "+-*/" : # in 연산자는 문자열이 들어오면 원소로 나누어서 인식, 항목이 연산자 이면
            val2 = s.pop()
            val1 = s.pop()
            if (token == '+'): # 해당하는 연산이면
                s.push(val1 + val2) # 해당하는 연산을 진행 한 후 s.top = []에 추가
            elif (token == '-'):
                s.push(val1 - val2)
            elif (token == '*'):
                s.push(val1 * val2)
            elif (token == '/'):
                s.push(val1 / val2)
        else: # 항목이 피연산자이면
            s.push( float(token) ) # 실수로 변경에서 스택에 저장
    return s.pop() # 최종결과를 반환

# 본문
if __name__ == "__main__":
    expr1 = [ '8', '2', '/', '3', '-', '3', '2', '*', '+']
    expr2 = [ '1', '2', '/', '4', '*', '1', '4', '/', '*']
    print(expr1, ' -->', evalPostfix(expr1))
    print(expr2, ' -->', evalPostfix(expr2))

'''
['8', '2', '/', '3', '-', '3', '2', '*', '+']  --> 7.0
['1', '2', '/', '4', '*', '1', '4', '/', '*']  --> 0.5
'''
```

```
# 중위 표기 수식의 후위 표기 변환

# 중위와 루위 표기법의 공통점 : 피연산자의 순서가 동일
# 중위와 루위 표기법의 차이점 : 연산자의 순서만 다름(우선 순위 순서)

# 알고리즘
# 피연산자를 만나면 그래도 출력
# 연산자를 ㅁ나나면 스택에 저장했다가 스택보다 우선 순위가 낮은 연산자가 나오면 그때 출력
# 왼족 괄호는 우선수위가 가장낮은 연산자로 취금
# 오른족 괄호가 나오면 스택에서 왼쪽 괄호위에 쌓여있는 모든 연산자를 출력

from Stack_class import Stack
from evalPostfix import evalPostfix

def precedence (op): # 우선수위 판별 함수
    if op == '(' or op == ')':
        return 0
    elif op == '+' or op == '-':
        return 1
    elif op == '*' or op == '/':
        return 2
    else :
        return -1

def Infix2Postfix( expr ): # expr : 입력 리스트(중위표기식)
    s = Stack() # s.top= [] 생성
    output = [] # output : 출력 리스트(후위 표기식)

    for term in expr :
        if term in '(': # 왼쪽 괄호이면
            s.push('(') # 스택에 삽입
        elif term in ')': # 오른쪽 괄호이면
            while not s.isEmpty(): # s.top= []가 채워져 있으면 반복
                op = s.pop() # 맨 위에 값 반환
                if op == '(': # 왼쪽 괄호가 나올때까지
                    break;
                else:
                    output.append(op) # 왼쪽 괄호를 만나면
        elif term in "+-*/": # 연산자이면
            while not s.isEmpty(): # s.top = []의 값들이 들어있으면 반복
                op = s.peek() # 스택에서 최근 값을 꺼내서
                if ( precedence(term) <= precedence(op)): # 현재 값과 우선순위 비교
                    output.append(op) # 현재 값의 우선순위가 최근 값의 우선순위보다 작거나 같으면
                    s.pop() # 후위표기 리스트에 추가
                else:
                    break
            s.push(term) # 현재 연산자 삽입
        else: # 피연산자 이면
            output.append(term) # 후위 표기 리스트에 추가

    while not s.isEmpty(): # s.top = []의 값이 있으면
        output.append(s.pop()) # 최근 값부터 차례대로 후위표기 리스트에 추가

    return output # 결과(후위 표기식 리스트)를 반환
```

```
# 본문
if __name__ == "__main__":
    infix1 = ['8', '/', '2', '-', '3', '+', '(', '3', '*', '2', ')'] # 평범식1 (중위표기)
    infix2 = ['1', '/', '2', '*', '4', '*', '(', '1', '/', '4', ')'] # 평범식2 (중위표기)
    postfix1 = Infix2Postfix(infix1) # 평범식1 -> 후위표기식1
    postfix2 = Infix2Postfix(infix2) # 평범식2 -> 후위표기식2
    result1 = evalPostfix(postfix1) # 후위표기식1의 값
    result2 = evalPostfix(postfix2) # 후위표기식2의 값

    # 출력
    print('중위표기: ', infix1)
    print('후위표기: ', postfix1)
    print('계산결과 : ', result1, end='\n\n')
    print('중위표기: ', infix2)
    print('후위표기: ', postfix2)
    print('계산결과 : ', result2)

'''
중위표기:  ['8', '/', '2', '-', '3', '+', '(', '3', '*', '2', ')']
후위표기:  ['8', '2', '/', '3', '-', '3', '2', '*', '+']
계산결과 :  7.0

중위표기:  ['1', '/', '2', '*', '4', '*', '(', '1', '/', '4', ')']
후위표기:  ['1', '2', '/', '4', '*', '1', '4', '/', '*']
계산결과 :  0.5
'''
```

```
# 스택의 응용 : 미로탐색

# 미로 구현 코드
from Stack_class import Stack

def DFS(): # 깊이우선탐색 함수(Depth Frish Search)
    stack = Stack() # Stack.top = [] 생성
    stack.push( (0,1) ) # 시작위치 삽입, (0,1)은 튜플
    print('DFS:')
    while not stack.isEmpty(): # 공백이 아닐동안
        here = stack.pop() # 최근의 좌표(튜플)을 pop을 통해서 반환
        print(here, end = '->')
        (x, y) = here # 스택에 저장된 튜플을 (x,y) 순서로 표기
        # 컴퓨터는 입력한 값이 열 중심으로 입력되기 때문에 y=-x 대칭으로 인식하여
        # map[x][y]가 아닌 map[y][x]로 해야 우리가 보는 모습과 같게 위치시킬수 있다.
        if (map[y][x] == 'x'): # 출구면 탐색 성공, T반환
            return True
        else:
            map[y][x] = '.' # 현재 위치를 지나왔다고 '.'표시, 그래야 isValidPos에서 범위 밖으로 판단하여 역주행을 안한다.
            # 4방향의 이웃을 검사해 갈 수 있으면 스택에 상하좌우를 우선순위로 삽입
            if isValidPos(x, y-1):
                stack.push((x, y-1)) # 상
            if isValidPos(x, y+1):
                stack.push((x, y+1)) # 하
            if isValidPos(x-1, y):
                stack.push((x-1, y)) # 좌
            if isValidPos(x+1, y):
                stack.push((x+1, y)) # 우
        print(' 현재스택: ', stack) # 현재 스택 내용 출력
    return False # 탐색 실패, F반환
```

```
def isValidPos(x, y): # 지금의 좌표가 유의한지 검사하는 함수
    if x < 0 or y < 0 or x >= MAZE_SIZE or y >= MAZE_SIZE : # 범위 밖이면 False
        return False
    else :
        return map[y][x] == '0' or map[y][x] == 'x' # 0 or x 이면 T, 1이면 F

# 본문
if __name__ == "__main__":
    map = [ [ '1', '1', '1', '1', '1', '1'],
            [ 'e', '0', '0', '0', '0', '1'],
            [ '1', '0', '1', '0', '1', '1'],
            [ '1', '1', '1', '0', '0', 'x'],
            [ '1', '1', '1', '0', '1', '1'],
            [ '1', '1', '1', '1', '1', '1']]
    MAZE_SIZE = 6
    result = DFS()

    if result:
        print('--> 미로 탐색 성공')
    else:
        print('--> 미로 탐색 실패')

'''
DFS:
(0, 1)-> 현재스택:  [(1, 1)]
(1, 1)-> 현재스택:  [(2, 1), (1, 2)]
(2, 1)-> 현재스택:  [(3, 1), (1, 2)]
(3, 1)-> 현재스택:  [(4, 1), (3, 2), (1, 2)]
(4, 1)-> 현재스택:  [(3, 2), (1, 2)]
(3, 2)-> 현재스택:  [(3, 3), (1, 2)]
(3, 3)-> 현재스택:  [(4, 3), (3, 4), (1, 2)]
(4, 3)-> 현재스택:  [(5, 3), (3, 4), (1, 2)]
(5, 3)->--> 미로 탐색 성공
'''
```

```
# 역순 출력

from Stack_class import Stack

stack = Stack() # 스택을 위한 stack.top =[] 리스트 생성
strs = input("역순으로 출력할 문자열을 입력하세요 :  ")

for s in strs:
    stack.push(s) # 역순으로 입력

# 스택(리스트)으로 출력
print(stack) # 스틱을 리스트 형식으로 만들었지 때문에 리스트 형식으로 출력된다.
# 스택(리스트)을 문자열로 변환하여 출력
print(''.join(stack.pop() for _ in range(len(strs))))

'''
역순으로 출력할 문자열을 입력하세요 :  list
['t', 's', 'i', 'l']
tsil
'''
```

```
# 회문인지 아닌지 확인하는 코드

# 함수 구현
from Stack_class import Stack

def palindrome(list):
    list = list.lower() # 대문자로 입력을 할 것을 대비하여 소문자로
    list_re = [] # 콤마, 마침표, 괄호, 특수기호, 숫자 등의 문자를 제외한 문자열을 만들기 위해 공간 설정
    for s in list: # 콤마, 마침표, 괄호, 특수기호, 숫자 등이 입력 되는 것을 막기 위해 아스키코드 값을 사용
        if s >="a" and s <="z":
            strs.push(s) # 스택에 저장
            list_re.append(s) # 리스트에 저장

    num = 1 # 참일 경우 1을 유지
    for i in range(strs.size()): # 자리수 만큼 비교를 한다.
        # 두 값이 리스트 형식이지만 리스트와 스택이므로 순서대로 비교하면 회문 비교가 가능
        if list_re[i] != strs.pop():
            num = 0 # 거짓일 경우 0으로 변경
    return num # 참:1 / 거짓:0 을 반환

def check_palindrome (num):
    if num == 1:
        print("회문입니다.")
    else:
        print("회문이 아닙니다.")

# 본문

if __name__ == "__main__":

    strs = Stack() # 스택 생성
    list = input("회문인지 확인할 출력할 문자열을 입력하세요 :  ")
    num = palindrome(list) # 사용자 지정 함수를 이용하여 회문 검사
    check_palindrome(num) # 결과 출력

'''
회문인지 확인할 출력할 문자열을 입력하세요 :  madam, i'm Adam
회문입니다.
'''
'''
회문인지 확인할 출력할 문자열을 입력하세요 :  madan, i'm Adam
회문이 아닙니다.
'''
```