대구대학교 DAEGU UNIVERSITY

일상에서 찾아보는

자료구조의 원리

03주차 쌓아 올리기 - 스택

3차시

파이썬 코딩: 수식의 괄호 검사하기, 미로 탈출하기

학습목표

- 스택을 이용하여 괄호가 여러 번 사용된 연산식에서 괄호 표기 오류를 검사할 수 있다.
- 스택을 활용한 깊이 우선 탐색으로 미로 탈출을 구현할 수 있다.

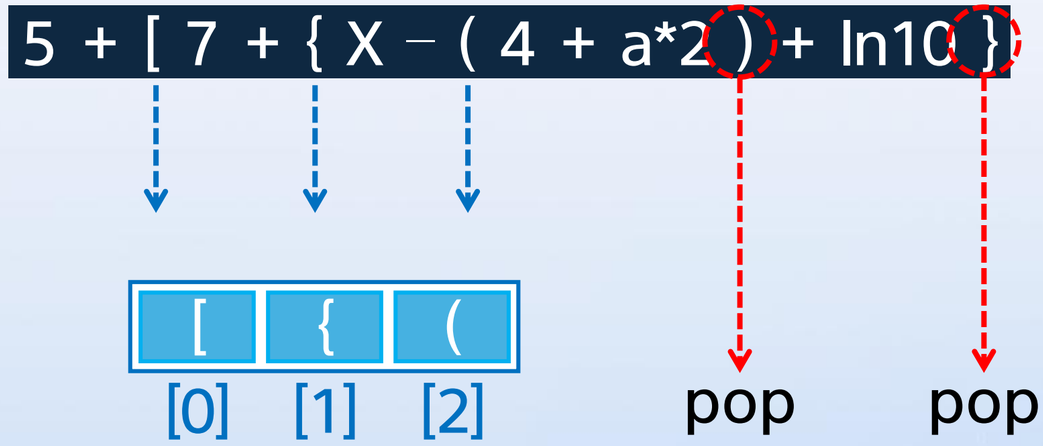
학습내용

- 스택 활용 실습: 수식의 괄호 검사
- 스택 활용: 깊이 우선 탐색
- 스택 활용 실습: 미로 탈출하기

스택 활용 실습 : 수식의 괄호 검사

- 왼쪽 괄호이면 push
- 오른쪽 괄호이면 pop

➡ 괄호의 쌍을 비교함



왼쪽 괄호를 만날 때마다 push

문자열을 모두 읽은 후에도
스택이 빈 상태가 아니므로
괄호가 잘못 쓰여진 경우임

스택 활용 실습: 수식의 괄호 검사

» 함수 정의

```
def checkBrackets(statement):
    stack = Stack()
    for ch in statement:   #수식 문자열에서 한 문자씩 읽는다.
        if ch in ('{', '[', '('):#문자가 왼쪽 괄호이면  push
            stack.push(ch)
        elif ch in ('}', ']', ')'):#문자가 오른쪽 괄호이면
            if stack.isEmpty() :    #스택이 비었는지 검사
                return "오른쪽 괄호가 많음"
            else :
                left = stack.pop()  #스택이 비어있지 않다면 pop
                if (ch == '}' and left != '{') or ₩
                   (ch == ']' and left != '[') or ₩
                   (ch == ')' and left != '(') :
                    return "괄호의 쌍이 맞지 않음"

    if stack.isEmpty() :
            return "오류 없음"
    else : return "왼쪽 괄호가 많음"
```

- 문자열에서 한 문자씩 읽음
  - 만약 왼쪽 괄호이면 push
  - 오른쪽 괄호이면 pop
    - ➡ 스택이 비었다면 오른쪽 괄호가 많음
    - ➡ pop과 괄호의 쌍을 비교
  - 문자열을 모두 읽은 후 스택이 빈 상태가 아니면 왼쪽 괄호가 많음

스택 활용 실습: 수식의 괄호 검사

» 테스트

```
#다음 세 가지의 문자열에서 괄호를 검사한다.
str = ( "{A[(i+1)] = 0;}", "if((i==0) && (j==0", "A[(i+1] ) = 0;" )
for s in str:
    result = checkBrackets(s)
    print('수식 {st:22} : {r}'.format(st=s, r = result))
```

```
수식 {A[(i+1)] = 0;} : 오류 없음
수식 if((i==0) && (j==0 : 왼쪽 괄호가 많음
수식 A[(i+1] ) = 0; : 괄호의 쌍이 맞지 않음
```

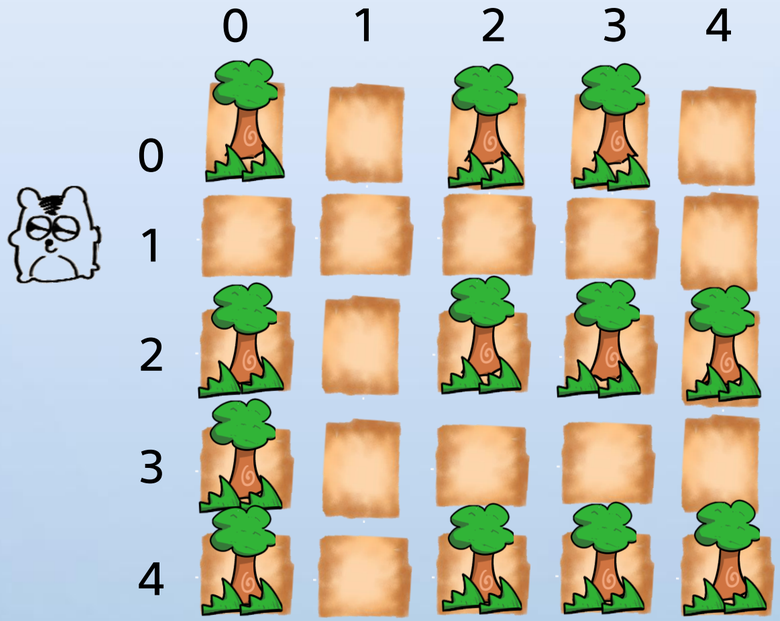
스택 활용 실습 : 미로 탈출

깊이 우선 탐색

가능한 한 깊게 탐색을 진행한 후, 더 이상 갈 수 없으면 다시 뒤로 돌아가 다른 경로를 찾는 방식

스택의 원리

막혀 있으면 다시 되돌아 옴



스택 활용 실습: 미로 탈출

```
def DFS(start, maze):
    stack = Stack()  # 스택 객체
    stack.push(start) # 출발 위치 삽입
    # 네 방향 (상, 하, 좌, 우)
    directions = [(0, -1), (0, 1), (-1, 0), (1, 0)]
    print("출발", start)

    while not stack.isEmpty():  # 스택이 빌 때까지 반복
        position = stack.pop() # pop 현재 위치 확인
        print("현재 위치", position, end=' -> ')
        x, y = position

        if maze[y][x] == 'x':  # 현재 위치가 도착점이면
            return "탈출 성공"

        maze[y][x] = '*'  # 방문 여부를 표시
        for direction in directions:  # 네 방향으로 이동 가능 지점 확인
            next_x, next_y = x + direction[0], y + direction[1]
            if (0 <= next_x < len(maze[0])) and
                    (0 <= next_y < len(maze)) and
                    maze[next_y][next_x] in ['0', 'x']:
                stack.push((next_x, next_y))
                        # 이동할 수 있는 지점이면 push

    # 스택 출력해 보기
    print("스택", stack.s_list)
    # 미로 출력해 보기
    for row in maze:
        for cell in row:
            print(cell, end=' ')
        print()
return "탈출 실패"
```

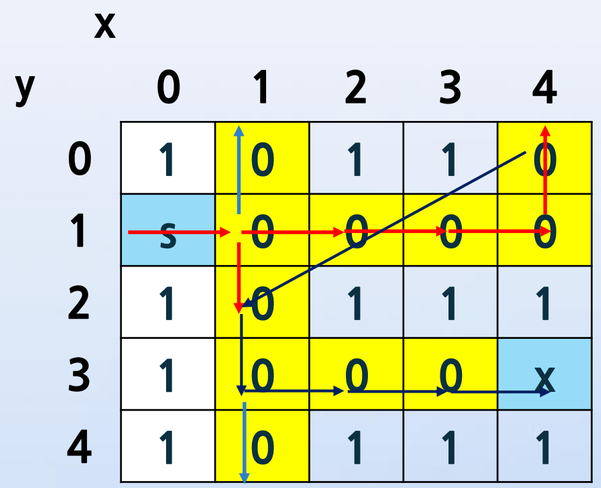
스택 활용 실습 :미로 탈출

```
# 미로 설정 (0: 경로, 1: 벽, x: 도착점)
maze = [
    ['1', '0', '1', '1', '0'],
    ['0', '0', '0', '0', '0'],
    ['1', '0', '1', '1','1'],
    ['1', '0', '0', '0', 'x'],
    ['1', '0', '1', '1', '1']
]

start = (0, 1)   # 출발 지점

# 미로 탐색 실행
result = DFS(start, maze)

print(result)
```



미로 출발 지점: 's'

미로 도착 지점: 'x'

1 0 1 1 *
* * * * *
1 * 1 1 1
1 * * * x
1 0 1 1 1

현재 위치 (4, 3)
→ 탈출 성공

학습정리

스택의 활용

- 입력의 반대 순으로 출력하여 사용하는 경우에 용이
  - 수식의 괄호가 올바른지를 검사하기 위해서 가장 가까운 괄호 쌍을 확인
  - 미로 탈출을 위해서 깊이 우선 탐색으로 길을 찾는 방법