9월 22일 4주차 수업 리포트

1. 계산기 구현

* **중위 표현식을 후위 표현식으로 변환한 다음, 후위 표현식을 사용하여 주어진 수식을 계산하는 계산기를 구현했다.**
* 스택 객체를 생성해서 각 연산을 수행한 후 결과는 스택에 다시 저장한다.
* 항목이 피연산자이면 실수로 변경해서 스택에 저장한다.
* 피연산자를 만나면 그대로 출력한다.
* 연산자를 만나면 스택에 저장했다가 스택보다 우선 순위가 낮은 연산자가 나오면 그 때 출력한다.
* 왼쪽 괄호는 우선순위가 가장 낮은 연산자로 취급한다.
* 오른쪽 괄호가 나오면 스택에서 왼쪽 괄호 위에 쌓여있는 모든 연산자를 출력한다.
* 알고리즘은 스택을 사용했다.
* 계산기 소스 코드

class Stack:

def \_\_init\_\_(self):

self.items = []

def push(self, item):

self.items.append(item)

def pop(self):

if not self.isEmpty():

return self.items.pop()

raise IndexError("pop from empty stack")

def peek(self):

if not self.isEmpty():

return self.items[-1]

def isEmpty(self):

return len(self.items) == 0

def infixToPostfix(expr): # 주어진 중위 표현식을 후위 표현식으로 변환

precedence = {'+':1, '-':1, '\*':2, '/':2, '(':0}

output = []

s = Stack()

number = ""

for token in expr:

if token.isdigit():

number += token

else:

if number:

output.append(number)

number = ""

if token == '(':

s.push(token)

elif token == ')':

while s.peek() != '(':

output.append(s.pop())

s.pop() # pop the '('

else:

while not s.isEmpty() and precedence[s.peek()] >= precedence[token]:

output.append(s.pop())

s.push(token)

if number:

output.append(number)

while not s.isEmpty():

output.append(s.pop())

return output

def evalPostfix( expr ): # 주어진 후위 표현식을 계산하여 결과 값을 반환

s = Stack()

for token in expr:

if token in "+-\*/":

val2 = s.pop()

val1 = s.pop()

if (token == '+'): s.push(val1 + val2)

elif (token == '-'): s.push(val1 - val2)

elif (token == '\*'): s.push(val1 \* val2)

elif (token == '/'): s.push(val1 / val2)

else:

s.push( float(token) )

return s.pop()

def calculate(expr): # 주어진 중위 표현식을 계산하여 결과 값을 반환

postfix\_expr = infixToPostfix(expr.replace(" ", "")) # 공백 제거

return evalPostfix(postfix\_expr)

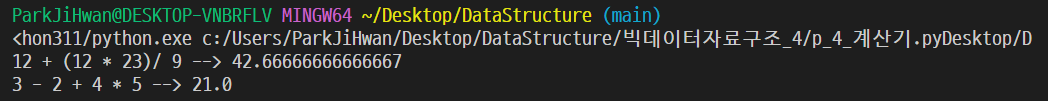
expr1 = "12 + (12 \* 23)/ 9"

expr2 = "3 - 2 + 4 \* 5"

print(expr1, '-->', calculate(expr1))

print(expr2, '-->', calculate(expr2))

* 수행 결과



1. 미로 탐색

* 주어진 2차원 미로에서 시작점 e부터 목표점 x까지의 경로를 찾는 것이다.
* 알고리즘은 BFS를 사용했다.
* BFS를 진행하기 위해 파이썬의 표준 라이브러리 collections 모듈에 포함된 자료형 deque를 사용했다.
* 미로 탐색 소스 코드

from collections import deque

def isValidPos(x, y):

if x < 0 or y < 0 or x >= MAZE\_SIZE or y >= MAZE\_SIZE: # 범위 바깥

return False

elif map[y][x] == '1' or map[y][x] == '.': # 벽이거나 이미 방문한 위치

return False

return True

def BFS():

que = deque()

que.append((0, 1))

print('BFS: ')

while que:

here = que.popleft()

print(here, end='->')

x, y = here

if (map[y][x] == 'x') : return True

else:

map[y][x] = '.'

if isValidPos(x, y - 1) : que.append((x, y - 1))

if isValidPos(x, y + 1) : que.append((x, y + 1))

if isValidPos(x - 1, y) : que.append((x - 1, y))

if isValidPos(x + 1, y) : que.append((x + 1, y))

return False

map = [ ['1', '1', '1', '1', '1', '1' ],

['e', '0', '1', '0', '0', '1' ],

['1', '0', '0', '0', '1', '1' ],

['1', '0', '1', '0', '1', '1' ],

['1', '0', '1', '0', '0', 'x' ],

['1', '1', '1', '1', '1', '1' ]

]

MAZE\_SIZE = 6

result = BFS()

if result : print('--> 미로탐색 성공')

else : print('--> 미로탐색 실패')

* 수행 결과

