**i-Tricat 신입부원 코딩 테스트**

**12210576 정재훈**

class Stack:

    def \_\_init\_\_(self):

        self.items = []                                 #스택을 초기화하는 메서드.

    def push(self, val):                                #스택에 원소를 추가하는 메서드.

        self.items.append(val)

    def pop(self):                                      #스택의 맨 위(가장 나중에 추가된) 원소를 제거하고 반환하는 메서드.

        try:

            return self.items.pop()

        except IndexError:

            print("Stack is empty")

    def top(self):                                      #스택의 맨 위(가장 나중에 추가된) 원소를 반환하는 메서드.

        try:

            return self.items[-1]

        except IndexError:

            print("Stack is empty")

    def \_\_len\_\_(self):                                  #스택의 길이를 반환하는 메서드.

        return len(self.items)

    def isEmpty(self):                                  #스택이 비어 있는지 여부를 반환하는 메서드.

        return self.\_\_len\_\_() == 0

# 입력 받은 수식의 오류를 찾아내는 함수

def search\_Error(expr):

    possible = list('0123456789.()+-\*/')            # 수식에 입력 가능한 문자들의 리스트

    integers = list('0123456789')                   # 0~9까지의 정수들의 리스트

    operators = list('+-\*/')                        # 수식에 사용될 연산자들의 리스트

    s = Stack()

    for i in range(len(expr)):

        if expr[i] not in possible:                 # 입력 가능한 문자가 아닐 경우 해당 위치 반환

            return i

        elif i == 0 and expr[i] in operators:       # 연산자가 수식의 가장 앞에 위치할 경우 해당 위치 반환 (ex. \*123+4...)

            return i

        elif expr[i] in operators:

            if i + 1 == len(expr):                  # 연산자 뒤에 피연산자가 오지 않고 수식이 끝날 경우 위치 반환 (ex. 1+2+)

                return i

            else:                                   # 연속해서 연산자가 위치할 경우 해당 위치 반환 (ex. 1+\*2...)

                if expr[i + 1] in operators:

                    return i + 1

        elif expr[i] == '(':                        # 수식에서 '('가 발견되었을 때

            if expr[i - 1] in integers and i != 0:  # 여는 괄호('(') 앞에 정수가 올 경우 해당 위치 반환

                return i

            else:

                if s.isEmpty():                     # 스택에 이미 값이 들어있으면 해당 위치 반환

                    s.push('(')                     # 스택이 비어있으면 push

                else:

                    return i

        elif expr[i] == ')':                        # 수식에서 ')'가 발견되었을 때

            if s.isEmpty():                         # 스택이 비어있으면

                return i                            # 해당 위치 반환

            elif expr[i + 1] in integers and i != len(expr) - 1:

                return i + 1                        # 닫는 괄호(')') 뒤에 정수가 올 경우 해당 위치 반환

            else:

                s.push(')')                         # 스택이 비어있지 않다면 해당 값을 스택에 push

    if not s.isEmpty():                             # 수식에 '('는 존재 하지만 ')'는 존재하지 않는 경우

        if s.pop() == '(':

            return len(expr)                        # 수식의 마지막 위치를 반환

# 사용자가 입력한 수식에서 피연산자를 인식하기 위한 함수

def recognize\_Operands(infix):

    numbers = list('0123456789.')                       # 실수 계산기 이므로, 모든 피연산자는 0~9인 정수와 .으로 이루어진다

    recognized = []                                      # 수식의 숫자들을 피연산자로 인식하여 다시 담아 줄 리스트

    i = 1

    while i < len(infix):

        j = 1

        if infix[i] in numbers:                         # 연산자가 아닐 경우, 즉 숫자 또는 .일 경우

            while i + j < len(infix):                   # 해당 요소의 다음 요소도 숫자 또는 .인지를 판별하고

                if infix[i + j] in numbers:

                    j += 1

                else:

                    break

            recognized.append(''.join(infix[i:i + j]))   # 이들을 하나로, 즉 하나의 숫자로 인식 할 수 있도록 묶어준다

            i += j

        else:                                           # 연산자일 경우엔 리스트에 바로 추가해준다

            recognized.append(infix[i])

            i += 1

    return recognized

priority = {'(': 1, ')': 1, '+': 2, '-': 2, '\*': 3, '/': 3}   # 연사자의 우선 순위(숫자가 클수록 더 높은 우선 순위)

# 중위 표기법 수식을 후위 표기법으로 바꿔주는 함수

def in2post(infix):

    s = Stack()                                             # 스택 생성

    postfix = []                                            # 후위 표기법으로 바뀐 수식을 담아 줄 리스트

    for i in infix:

        if i == '(':                                        # 여는 괄호('(')일 경우 스택에 바로 push

            s.push(i)

        elif i == ')':                                      # 닫는 괄호(')')일 경우

            while s.top() != '(':                           # '('와 ')'사이의 연산자들을 모두 postfix 리스트에 추가

                postfix.append(s.pop())

            s.pop()                                         # 그리고 stack에 들어있던 '('는 삭제. (후위 표기법은 괄호를 표시하지 않으므로)

        elif i in priority:                                 # '+ - \* /'일 경우

            while not s.isEmpty():                          # 스택이 비어 있지 않을 때

                if priority[s.top()] >= priority[i]:        # 스택의 top에 해당하는 연산자의 우선순위가 비교할 연산자의 우선순위보다 크거나 같을 경우

                    postfix.append(s.pop())                 # 해당 연산자(top)를 postfix 리스트에 추가한다

                else:

                    break

            s.push(i)                                       # 스택이 비어있을 경우엔 해당 연산자를 스택에 바로 push

        else:                                               # 피연산자(숫자)의 경우 리스트에 바로 추가

           postfix.append(i)

    while not s.isEmpty():                                  # '('보다 밑 스택에 남아있던 연산자들을

        postfix.append(s.pop())                             # 모두 리스트에 추가해준다

    return postfix                                          # 변환된 후위 표기식을 반환한다

# 후위 표기법으로 변환된 수식을 계산해주는 함수

def post\_Cal(postfix):

    s = Stack()

    for i in postfix:

        if i in priority:                                   # 연산자일 경우

            num1 = s.pop()                                  # 스택에 쌓여 있던 두 피연산자(숫자)를 꺼내

            num2 = s.pop()

            if i == '+':                                    # 덧셈(+)

                s.push(num2 + num1)

            elif i == '-':                                  # 뺄셈(-)

                s.push(num2 - num1)

            elif i == '/':                                  # 나눗셈(/)

                s.push(num2 / num1)

            elif i == '\*':                                  # 곱셈(\*)

                s.push(num2 \* num1)                         # 두 피연산자에 대해 각 연산자에 해당하는 연산을 수행한 값을 스택에 push해준다

        else:                                               # 숫자일 경우

            s.push(float(i))                                # 해당 숫자를 float 자료형으로 스택에 push(실수 계산기이므로)

    return s.pop()                                          # 계산된 값을 반환한다

while True:

    expr = list(input("수식을 입력하세요 : "))                  # 수식을 입력 받음

    if search\_Error(expr) != None:                          # 수식의 오류를 찾는다

        print(' ' \* 15, ' ' \* search\_Error(expr), end='')

        print('^이 위치에 오류가 발견되었습니다.')

        continue

    infix = recognize\_Operands(expr)                        # 오류가 없을 경우 수식의 연산자를 인식하여 정리한다

    postfix = in2post(infix)                                # 수식을 후위 표기법으로 바꾸어준다

    print('=', post\_Cal(postfix))

#EngineerCalculator 클래스의 생성자 메서드입니다.

class EngineerCalculator(Calculator):

    def \_\_init\_\_(self, calculator\_id=None):

        super().\_\_init\_\_(calculator\_id)

    def factorial(self, n):                                #팩토리알 함수

        if n == 0 or n == 1:

            return 1

        else:

            return n \* self.factorial(n - 1)

    def calculate\_trigonometric(self, function, angle\_degrees):   # sin,cos,tan 함수

        angle\_radians = math.radians(angle\_degrees)

        if function == 'sin':

            return math.sin(angle\_radians)

        elif function == 'cos':

            return math.cos(angle\_radians)

        elif function == 'tan':

            return math.tan(angle\_radians)

    def calculate\_engineering(self, expression):                    # 공학적 게산기의 함수

        try:

            for char in expression:

                if char.isdigit():

                    self.push(int(char))

                elif char.isalpha():

                    function = char.lower()

                    if function in {'s', 'c', 't'}:

                        if len(self.stack) < 1:

                            raise ValueError("삼각 함수에 필요한 피연산자가 충분하지 않습니다")

                        angle = self.pop()

                        result = self.calculate\_trigonometric(function, angle)

                        self.push(result)

                    elif function == 'f':

                        if len(self.stack) < 1:

                            raise ValueError("계승 함수에 필요한 피연산자가 충분하지 않습니다")

                        operand = self.pop()

                        result = self.factorial(operand)

                        self.push(result)

                elif char in {'+', '-', '\*', '/'}:

                    self.calculate\_step()

                    self.push(char)

            while not self.is\_empty():

                self.calculate\_step()

            self.result = self.pop()

        except ValueError as e:

            print(f"계산 오류: {e}")

            self.result = None

    def main():

    calculators = {}                                        # 계산기 객체를 저장할 딕셔너리

    while True:

        print("옵션:")

        print("1. 계산기 생성")

        print("2. 계산 수행")

        print("3. 종료")

        choice = input("원하는 옵션을 입력하세요: ")

        if choice == '1':                                  #계산기 id를 입력

            calculator\_id = input("계산기 ID를 입력하세요: ")

            calculators[calculator\_id] = EngineerCalculator(calculator\_id=calculator\_id)

            print(f"계산기 {calculator\_id}이(가) 생성되었습니다.")

        elif choice == '2':                                #계산기 수식을 입력

            calculator\_id = input("계산기 ID를 입력하세요: ")

            if calculator\_id in calculators:

                expression = input("수식을 입력하세요: ")

                calculators[calculator\_id].calculate\_engineering(expression)

                print(f"{calculator\_id} 계산기 결과: {calculators[calculator\_id].result}")

            else:

                print(f"{calculator\_id} 계산기가 존재하지 않습니다.")

        elif choice == '3':                               #계산기 프로그램을 종료

            print("프로그램을 종료합니다.")

            break

        else:

            print("잘못된 선택입니다. 1, 2, 또는 3을 입력하세요.")

    # 각 계산기의 최종 결과 출력

    for calculator\_id, calculator in calculators.items():

        print(f"{calculator\_id} 계산기 최종 결과: {calculator.result}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**최대한 코드를 짜보았습니다. 주석에 설명은 최대한 달아 놓았습니다. 코드 실행하면 자꾸 none 이라고 뜨는데 진짜 몇번을 해보아도 계산값이 안 나옵니다. 계산기 객체를 활용한 딕셔너리를 안넣으면 실수의 사칙연산은 계산이 되는데 계산기 객체를 활용하면 사직연산도 되지않습니다. Sin, cos, tan 함수는 몇백번은 해보았는데 어디서 잘못되었는지 안나옵니다. Sin()도 해보고 띄어쓰기도 해보고 안되서 그냥 냈습니다. 그리고 실수 계산할때에 6.0+6.0을 하면 첫번째 숫자인 6은 사라지고 .뒤에 숫자인 0만 계산이되어서 0.0+6.0 이 됩니다. 예) 4.12+4.54=12+4.54=16.54**

**앞에 숫자가 사라집니다. 최대한 코드 짜보았는데 내보겠습니다.**