A1-231277

COAL LAB

CS-360L

Submitted To: Mam Komal

Submitted by: Alishba Malik

Class & Section: BSCYS-F23-III-B

Reg ID: 231277

Date of Submission: 10/03/2024

BODMAS Calculator

```
import math

# Function to perform basic arithmetic operations
def apply_operation(op, a, b):
    if op == '+':
        return a + b
    elif op == '-':
        return a - b
    elif op == '*':
        return a * b
    elif op == '/':
        if b == 0:
            raise ValueError("Division by zero!")
        return a / b
    elif op == '%':
        if b == 0:
            raise ValueError("Modulo by zero!")
        return a % b
    elif op == '^':
        return math.pow(a, b)
    else:
        raise ValueError("Invalid operator!")
```

```
# Function to determine precedence of operators
def precedence(op):
    if op == '+' or op == '-':
        return 1
    if op == '*' or op == '/' or op == '%':
        return 2
    if op == '^':
        return 3
    return 0

# Function to process the expression using BODMAS rule with b
def evaluate(expression):
    # Function to handle precedence of operations within pare
    def evaluate_stack(ops, values):
        while ops and ops[-1] != '(':
            val2 = values.pop()
            val1 = values.pop()
            op = ops.pop()
            values.append(apply_operation(op, val1, val2))

    values = []  # Stack to store numbers
    ops = []     # Stack to store operators

    i = 0
    while i < len(expression):
        # If current character is a whitespace, skip it
        if expression[i] == ' ':
            i += 1
            continue

        # If current character is a number or a negative sign
        if expression[i] == '-' and (i == 0 or expression[i-1
            # Detect negative number
            sign = -1
            i += 1
        else:
            sign = 1
        if expression[i].isdigit() or expression[i] == '.':
            val = 0
            decimal_place = 0
            is_float = False

            # Parse the number, including fractional part
            while i < len(expression) and (expression[i].isdi
                if expression[i] == '.':
                    is_float = True
                    decimal_place = 1
                else:
                    if is_float:
                        val += (int(expression[i]) / (10 ** d
                        decimal_place += 1
                    else:
                        val = val * 10 + int(expression[i])
                i += 1
            values.append(sign * val)
            continue

        # If opening parenthesis, push it to ops stack
        elif expression[i] == '(':
            ops.append(expression[i])

        # If closing parenthesis, solve entire bracket
        elif expression[i] == ')':
            while ops and ops[-1] != '(':
                val2 = values.pop()
                val1 = values.pop()
                op = ops.pop()
                values.append(apply_operation(op, val1, val2)
            ops.pop()  # Remove '(' from the ops stack

        # Current character is an operator
        else:
            while ops and precedence(ops[-1]) >= precedence(e
                val2 = values.pop()
                val1 = values.pop()
                op = ops.pop()
                values.append(apply_operation(op, val1, val2)
            ops.append(expression[i])

        i += 1

    # After entire expression is parsed, apply remaining oper
    while ops:
        val2 = values.pop()
        val1 = values.pop()
        op = ops.pop()
        values.append(apply_operation(op, val1, val2))


    return values[-1]

if __name__ == "__main__":
    expression = "-15 + 12 * (4 * 6/3) - 2"
    try:
        result = evaluate(expression)
        print(f"Result: {result}")
    except Exception as e:
        print(f"Error: {e}")
```