**Problem: Different Ways to Add Parentheses**

**Scenario**

You are given a string expression that contains numbers and operators. The task is to return all possible results from computing the different ways to group the numbers and operators using parentheses. The output can be returned in any order.

**Problem Statement**

Given a string expression of numbers and operators, print all possible results from computing all the different possible ways to group numbers and operators.

**Input Format**

* A single line containing the string expression.

**Constraints**

* 1 <= expression.length <= 20
* The expression consists of digits and the operators '+', '-', and '\*'.
* All the integer values in the input expression are in the range [0, 99].

**Output Format**

* A list of integers representing all possible results from computing the different ways to group the numbers and operators.

**Example**

**Example 1:**

Input:

"2-1-1"

Output:

[0, 2]

Explanation:

((2-1)-1) = 0

(2-(1-1)) = 2

**Example 2:**

Input:

"2\*3-4\*5"

Output:

[-34, -14, -10, 10]

Explanation:

(2\*(3-(4\*5))) = -34

((2\*3)-(4\*5)) = -14

((2\*(3-4))\*5) = -10

(2\*((3-4)\*5)) = -10

(((2\*3)-4)\*5) = 10

**Other Test Cases**

**Test Case1**

**Input**

"2\*10-4\*10"

**Output**

[-60, -20, 120, 160]

**Test Case2**

**Input**

"2\*10-4+10"

**Output**

[-8, 6, 22, 26, 32]

**Test Case3**

**Input**

"2+10-4+10"

**Output**

[-2, 18]

**Test Case4**

**Input**

"2+10+4+10"

**Output**

[26]

**Test Case5**

**Input**

"2-10-4+10"

**Output**

[-22, -14, -2, 6]

**Solution Approach**

The solution uses a divide and conquer approach to evaluate all possible ways to add parentheses in the expression. Here is the detailed approach:

1. **Base Case:** If the expression contains only a single number, return it as the only possible result.
2. **Divide:** For each operator in the expression, split the expression into two parts:
   * The left part contains the expression before the operator.
   * The right part contains the expression after the operator.
3. **Conquer:** Recursively solve the left and right parts to get all possible results.
4. **Combine:** For each result from the left part and each result from the right part, combine them using the operator.
5. **Memoization:** Use memoization to store and reuse the results of sub-expressions to optimize the solution.

Here is the Python code implementing this approach:

python

def diffWaysToCompute(expression):

memo = {}

def ways(expression):

if expression in memo:

return memo[expression]

if expression.isdigit():

return [int(expression)]

res = []

for i in range(len(expression)):

if expression[i] in "+-\*":

left = ways(expression[:i])

right = ways(expression[i+1:])

for l in left:

for r in right:

if expression[i] == '+':

res.append(l + r)

elif expression[i] == '-':

res.append(l - r)

elif expression[i] == '\*':

res.append(l \* r)

memo[expression] = res

return res

return ways(expression)

# Reading input

expression = input().strip()

# Computing the results

results = diffWaysToCompute(expression)

# Printing output

print(results)

This code reads the input expression, computes all possible results using the diffWaysToCompute function, and prints the results. The function ways handles the recursive computation with memoization to improve performance.