**1. Dynamic calculator without oop concept: -**

* **Introduction:**
  + The program evaluates a mathematical expression given as a string (example: "2\*3/6-3+8").
  + It scans the string to find operators and performs the corresponding operation.
  + It collects intermediate results in a list to understand step-by-step calculations.
* **Concepts Used:**

1. String Iteration (for loop with enumerate ()):

* for x, i in enumerate(n): → goes through each character of the string with both index (x) and value (i).
* Example: in "2\*3", at position 1 we get "\*" operator.

1. Operator Handling (if-elif):

* Checks each character to see if it is \*, /, +, or -.
* Based on the operator, the correct mathematical operation is performed.

1. Indexing (x-1, x+1):

* To get numbers around the operator:
* n[x-1] → left number.
* n[x+1] → right number.
* Example: in "2\*3", operator "\*" at index 1 → left = 2, right = 3.

1. Type Conversion (int()):

* Characters in string are text, so int() is used to convert them into numbers before calculation.

1. List Storage (results.append()):

* After each operation, the result is stored in a list called results.
* Helps track all intermediate results clearly.
* **Step-by-Step Execution Example (n = "2\*3/6-3+8"):**

1. 2 \* 3 = 6 → stored in results.
2. 3 / 6 = 0.5 → stored in results.
3. 6 - 3 = 3 → stored in results.
4. 3 + 8 = 11 → stored in results.

* Final intermediate results list: **[6, 0.5, 3, 11]**
* **Conclusion:**
  + Learned how to scan a string and detect operators.
  + Learned to use **loops, if-else, and indexing** to extract numbers around operators.
  + Understood how int() converts characters into usable numbers.
  + Practiced storing intermediate values in a list for better debugging.
  + Observed that this program doesn’t fully follow BODMAS, since it always takes numbers from the **original string** instead of updating the result.

**2.Dynamic Calculator with OOP: -**

* **Introduction: -**
  + This is a custom calculator class that evaluates string expressions.
  + It follows **BODMAS rule**:
* Brackets
* Multiplication & Division
* Addition & Subtraction
* It parses numbers and operators from the string without using Python’s built-in eval().
* **Code Explanation Step by Step:**

1. **Constructor:**

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

self.expr = expression.replace(" ", "")

* Takes an expression like "8 - (2\*3) + 6/2".
* Removes spaces → stored in self.expr.

1. **Solve Method (Main Driver):**

def solve(self):

* Responsible for solving the expression in **3 phases**:

**Step 1 → Brackets:**

while "(" in self.expr:

close = self.expr.find(")")

open = self.expr.rfind("(", 0, close)

inside = self.expr[open+1:close]

result = SimpleCalculator(inside).solve()

self.expr = self.expr[:open] + str(result) + self.expr[close+1:]

* Finds innermost bracket ( ... ).
* Extracts inside expression → solves recursively.
* Replaces bracket with solved result.
* **Example: "8 - (2\*3) + 6/2" → first becomes "8 - 6 + 6/2".**

**Step 2 → Multiplication & Division:**

**self.expr = self.\_solve\_mul\_div(self.expr)**

* Handles \* and / left to right.
* Uses helper \_get\_left\_number and \_get\_right\_number to fetch full numbers (not just one digit).
* Replaces that part with computed result.
* Repeats until no more \* or /.
* **Example: "8 - 6 + 6/2" → becomes "8 - 6 + 3".**

**Step 3 → Addition & Subtraction:**

**self.expr = self.\_solve\_add\_sub(self.expr)**

* Similar logic but for + and -
* Resolves everything left to right.
* **Example: "8 - 6 + 3" → first "2 + 3" → then "5".**

**Final Return:**

**return int(self.expr)**

* Converts final string result to integer.
* Output: 5

**Key Learnings:**

* Built a calculator without using Python’s eval()
* Learned recursive solving of **brackets**
* Learned **string parsing** with custom left/right number extraction
* Applied **BODMAS order** properly.
* Program works even with multi-digit numbers.