Evaluate Reverse Polish Notation

Evaluate Reverse Polish Notation

Medium

You are given an array of strings `tokens` that represents a valid arithmetic expression in Reverse Polish Notation. - mathematical expression without parentheses or operator precedence rules (so all operators are written after the numbers)

Return the integer that represents the evaluation of the expression.

The operands may be integers or the results of other operations. The operators include `'+'`, `'-'`, `'*'`, and `'/'`.Assume that division between integers always truncates toward zero.

- The operands may be integers or the results of other operations.
- The operators include `'+'`, `'-'`, `'*'`, and `'/'`.
- Assume that division between integers always truncates toward zero.

Example 1:

```
Input: tokens = ["1","2","+","3","*","4","-"]

Output: 5

Explanation: ((1 + 2) * 3) - 4 = 5
```

Copy

Constraints:

`1 <= tokens.length <= 1000`.tokens[i] is `"+"`, `"-"`, `"*"`, or `"/"`, or a string representing an integer in the range `[-100, 100]`.

- `1 <= tokens.length <= 1000`.
- tokens[i] is `"+"`, `"-"`, `"*"`, or `"/"`, or a string representing an integer in the range `[-100, 100]`.

```
class Solution:
    def evalRPN(self, tokens: List[str]) -> int:

        stack = []
        operator_list = {'+': lambda a, b: a + b,
                         '-': lambda a, b: a - b,
                         '*': lambda a, b: a * b,
                         '/': lambda a, b: int(a / b)}
        for i in tokens:
            if i.lstrip('-').isdigit():
                stack.append(int(i))
            elif i in operator_list:
                latest = stack.pop()
                second_latest = stack.pop()
                result = operator_list[i](second_latest, latest)
                stack.append(result)

        return stack[0]
```

By using lambda functions, we can actually define the meaning of the operator list