**Evaluate a Postfix Expression**

Given a postfix expression (also known as Reverse Polish Notation), your task is to evaluate the expression and return the result. The expression can contain integers and the four basic arithmetic operators +, -, \*, and /. Assume that the input is always valid and the division operator performs integer division, truncating towards zero.

**Input:**

* A string representing a postfix expression where operands and operators are separated by spaces.
* The string contains only integers (both positive and negative) and the operators +, -, \*, and /.

**Output:**

* An integer representing the result of evaluating the postfix expression.

**Examples:**

* Example 1  
  Input: "2 1 + 3 \*"

Output: 9  
Explanation:

* First, 2 and 1 are pushed onto the stack.
* Encountering '+', 1 and 2 are popped, added to get 3, and pushed back onto the stack.
* Then, 3 is pushed onto the stack.
* Encountering '\*', 3 and 3 are popped, multiplied to get 9, and pushed back onto the stack.
* The final result is 9.

**Constraints:**

* The input is always a valid postfix expression.
* The input contains only integers and the operators +, -, \*, /.
* The division operator / performs integer division, truncating toward zero.
* The length of the input string is between 1 and 1000 characters.

**Test Cases:**

1. Input: "5 6 +"

Output: 11

1. Input: "3 4 2 \* 1 5 - 2 3 ^ ^ / +"

Output: -1

1. Input: "-5 6 -"

Output: -11

1. Input: "15 7 1 1 + - / 3 \* 2 1 1 + + -"

Output: 5

1. Input: “5”

Output: 5

**Edge Cases:**

1. Single Operand: If the postfix expression consists of a single operand (e.g., "42"), the result should be the operand itself.
2. Negative Numbers: The expression can include negative integers.
3. Integer Division: Division should result in an integer truncated toward zero.
4. Multiple Operations: The expression can have multiple operations, ensuring that the stack handles operations in the correct order.