Infix Evaluation

Given a string s which represents an expression, evaluate this expression and return its value

Input: s = "2 + 4 * 2"

Output: 8

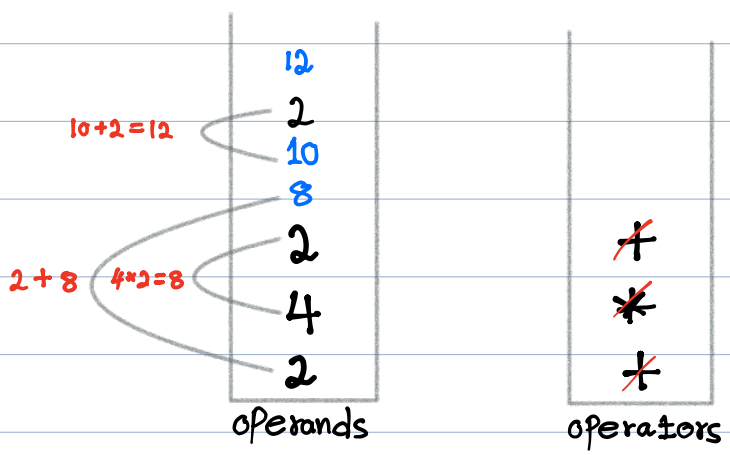
Input: s = "2 + 4 / 2"

Output: 4

key Idea -1: take care of precedence order, first solve the operands around operators that has highest precedence

key Idea - 2: the operator with lowest precedence divide the expression into two half, the half on the left side of the operator can be solved once we reach to a point where current operator's precedence is lower than the operator we have seen already.

Example: 2 + 4 * 2 + 2



Algorithm :

1. Create two stacks, one for operands and for operators.
2. Check if current element is a number, if number, check if there are strings of numbers.
   - 2.i use a loop to join strings of digit into an integer number.
3. Add the number to the operands stack.
4. check if current element is an operator, if so
   - 4.i check if current operator's precedence is lower than the operator's stack top element, if so, do the operations of operators which are before current operator.
   - 4.ii if not greater than the top element, just add it to the operator's stack.
5. continue the above steps until all elements are processed.

```
def calculate(s):
    while( i < len(s):
        //combine multiple digits
        ch = s[i]
        if "0" <= ch <= "9":
            num = 0
            while i <= len(s) and "0" <= s[i] <= "9":
                num = num * 10 + int(s[i])
                i = i + 1
            // Add the combined digits to the operands stack.
            operands.append(num)
            // while combining the digits, we incremented i upto the operator
            i = i - 1
        // if current element is an operator
        elif ch in {"+", "-", "/", "*"}:
            // if the operators stack is not empty and precedence rule is followed
            while operators and self.precedence(operators[-1]) => self.precedence(ch):
                // Do the previous operators operations
                operator = operators.pop()
                value2 = operends.pop()
                value1 = operands.pop()
                result = self.calc(value1, value2, operator)
                operands.append(result)
            // Add the current operator either after doing the operation or
               without doing the given operations
            operators.append(ch)
        // Increment i to move to the next element
        i = i + 1
```

```
// if the operator stack still have some operators
while operators:
        // Do the previous operators operations
        operator = operators.pop()
        value2 = operends.pop()
        value1 = operands.pop()
        result = self.calc(value1, value2, operator)
        operands.append(result)
// Add the current operator either after doing the operation or
   without doing the given operations
operators.append(ch)
return operands[0]
```

```
def precedence(self, op):
        if op == "+" or op == "-":
            return 0
        else:
            return 1
```

```
def calc(op1, op2, oper):
        if oper == "+":
            return op1+op2
        elif oper == "-":
            return op1-op2
        elif oper == "*":
            return op1*op2
        else:
            return op1/op2
```