Name

**RAUNAK RAJESH SHAH**

Email

shahrrs2004@gmail.com

Cohort

**Cohort-29 FSN**

TOPIC

**Data Structure and Algorithms**

College

**Walchand Institute of Technology**

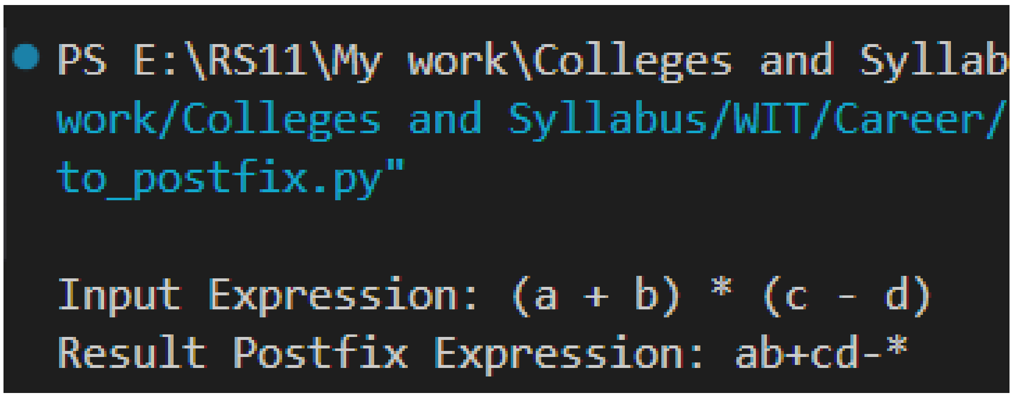
# Assignment 3:

**Implementation of Infix to Postfix Convertor using Stack.**

## Code:

```python
# Assignment 3: Infix to Postix Convertor by RAUNAK SHAH using PYTHON
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return len(self.items) < 1

    def push(self, element):
        self.items.append(element)

    def pop(self):
        if not self.isEmpty():
            return self.items.pop()
        else:
            raise IndexError("Pop method cannot be done when stack is empty.
No Element to pop.")

    def peek(self):
        if not self.isEmpty():
            return self.items[-1]
        else:
            raise IndexError("No Element to peek in the stack.")

    def size(self):
        return len(self.items)

    def join(self) -> str:
        expression = ""
        for element in self.items:
            expression = expression + str(element)

        return expression


# infix to postfix Convertor function
def infixToPostfix(expression):
    expression = "(" + expression + ")"
    def precedence(operator):
        if operator == '+' or operator == '-':
            return 0
        elif operator == '*' or operator == '/':
            return 1
        else:
            return 2
    operators = Stack()
    postfix = Stack()
    i = 0
    while i < len(expression):
        if expression[i].isalpha():
            j = i
            while j < len(expression) and (expression[j].isalpha() or
expression[j] == '.'):
                j += 1
            postfix.push(expression[i:j])
            i = j
        elif expression[i] in "+-*/":
            while (not operators.isEmpty() and operators.peek() in "+-*/" and
precedence(expression[i]) <= precedence(operators.peek())):
                operator = operators.pop()
                postfix.push(operator)
            operators.push(expression[i])
            i += 1
        elif expression[i] == "(":
            operators.push(expression[i])
            i += 1
        elif expression[i] == ")":
            while (not operators.isEmpty() and operators.peek() != '('):
                operator = operators.pop()
                postfix.push(operator)
            operators.pop()
            i += 1
        else:
            i += 1

    while not operators.isEmpty():
        postfix.push(operators.pop())

    # return the remaining element in postfix i.e the final answer
    return postfix.join()


# Converting an Infix expression to a Postfix expression.
expression = "(a + b) * (c - d)"
print(f"\nInput Expression: {expression}")
print(f"Result Postfix Expression: {infixToPostfix(expression)}")
```

Output:



```
● PS E:\RS11\My work\Colleges and Syllab
  work/Colleges and Syllabus/WIT/Career/
  to_postfix.py"

  Input Expression: (a + b) * (c - d)
  Result Postfix Expression: ab+cd-*
```