Infix to Postfix

$a + b \Rightarrow ab+$

Ex: $x + y * z \Rightarrow (x + (y * z)) \Rightarrow (x + (yz*)) \Rightarrow xyz*+$

Algorithm:

1. create two empty stacks one for operands and one for operators
2. start iterating over element x
3. If x is "(" add it to the operators stack
4. If x is ")", Traverse the whole operators stack, perform all the operators operation until you find "(" once "(" is found, remove it
5. If x is from [+, −, *, /, ^]
   - 5.1 If operators stack is empty Add the operator x to the stack
   - 5.2 If stack is not empty
     - 5.2.1 If x's precedence is higher than the top most operator in the operators stack, Add it the stack
     - 5.2.2 If x's precedence is lower than the top most operator in the operators stack
       - 5.2.2.1 perform all the operations in operator's stack and then Add the element x into the operators stack.
6. Return the top most element from operand stack

```
def InfixToPostfix (expression)
    operators = [ ]
    operands = [ ]
    i = 0
    while (i<len(expression)):
        e = expression[i]
        if e == "(":
            operators.append(e):
        elif e==")":
            while operators and operators[-1] != "(":
                operator = operators.pop()
                value2 = operands.pop()
                value1 = operands.pop()
                operation = value1 + value2 + operator
            // Remove "("
            operators.pop()
        elif e in {"+", "-", '*', '/', '^'}
            while operators and self.pre(e) > self.pre([operators[-1]):
                operator = operators.pop()
                value2 = operands.pop()
                value1 = operands.pop()
                operation = value1 + value2 + operator
                operands.append (operation)
            // Add the current operator
            operators.appen(e)
        // if it is a number
        else:
            operands.append(e)
```

```
// if still there are operators left in the stack.
while operators:
        operator = operators.pop()
        value2 =   operands.pop()
        value1 =   operands.pop()
        operation =  value1 + value2 + operator
        operands.append(operation)
// return top most element
return operands[-1]
```

```
// Returns the precedence of an operator
def pre(self, op):
        if op == "+" or op == "-":
            return 0
        else:
            return 1
```