12: Convert infix expression to postfix expression

Infix notation- when the operator is written in between the operands, then its known as infix notation.
Eg: (p+q)*(r+s)

Postfix notation - The postfix expression is an expression in which the operator is written after the operands. For example, the postfix expression of infix notation (2+3) can be written as 23+.

Infix to postfix Transformation
procedure
step 1: Scan the infix expression from left to right
Step 2: a) If the scanned symbol is left parenthesis, push it onto the stack.
b) If the scanned symbol is operand, then place directly in the postfix expression (output)
c) If the symbol scanned is a right paranthesis, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parenthesis.
d) If the scanned symbol is an operator, then go on removing all the operators from the stack and place them in postfix expression, if and only if the precedence if the scanned operator which is on top of the stack is greater than (or equal) to the precedence of the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack

Example:
Converting A*(B*C+D*E)+F :

	current Token	stack	postfix string
1	A		A
2	*	*	A
3	(	*(	A
4	B	*(	AB
5	*	*(*	AB
6	C	*(*	ABC
7	+	*(+	ABC*
8	D	*(+	ABC*D
9	*	*(+*	ABC*D
10	E	*(+*	ABC*DE
11	)	*	ABC*DE+*
12	+	+	ABC*DE+*
13	F	+	ABC*DE+*F
14			ABC*DE+*F+

postfix expression of A*(B*C+D*E)+F
= ABC*DE+*F+

Code:-

```
def infix to postfix(expression):
    stack = [ ]
    output = ''
    Operations = set(['+', '-', '*', '/', '(', ')', '^'])
    priority = {'+':1, '-':1, '*':2, '/':2, '^':3}
    for character in expression:
        if character not in Operators:
            Output += character
        elif character = '(':
            stack.append('(')
        elif character == ')':
            while stack and stack[-1] != '(':
                output = stack.pop()
            stack.pop()
        else:
            while stack and stack[-1] = '(' and priority[character]
                                                  <= priority
                output += stack.pop()
            stack.append(character)
    while stack:
        output = stack.pop()
    return output
expression = input('Enter infix expression')
print('infix notation :', expression)
print('postfix notation:', infix to postfix (expression))
```

Output -

```
Enter infix expression : (A-B)*(D/E)
      infix notation : (A-B)*(D/E)
   postfix notation : AB-DE/*
```