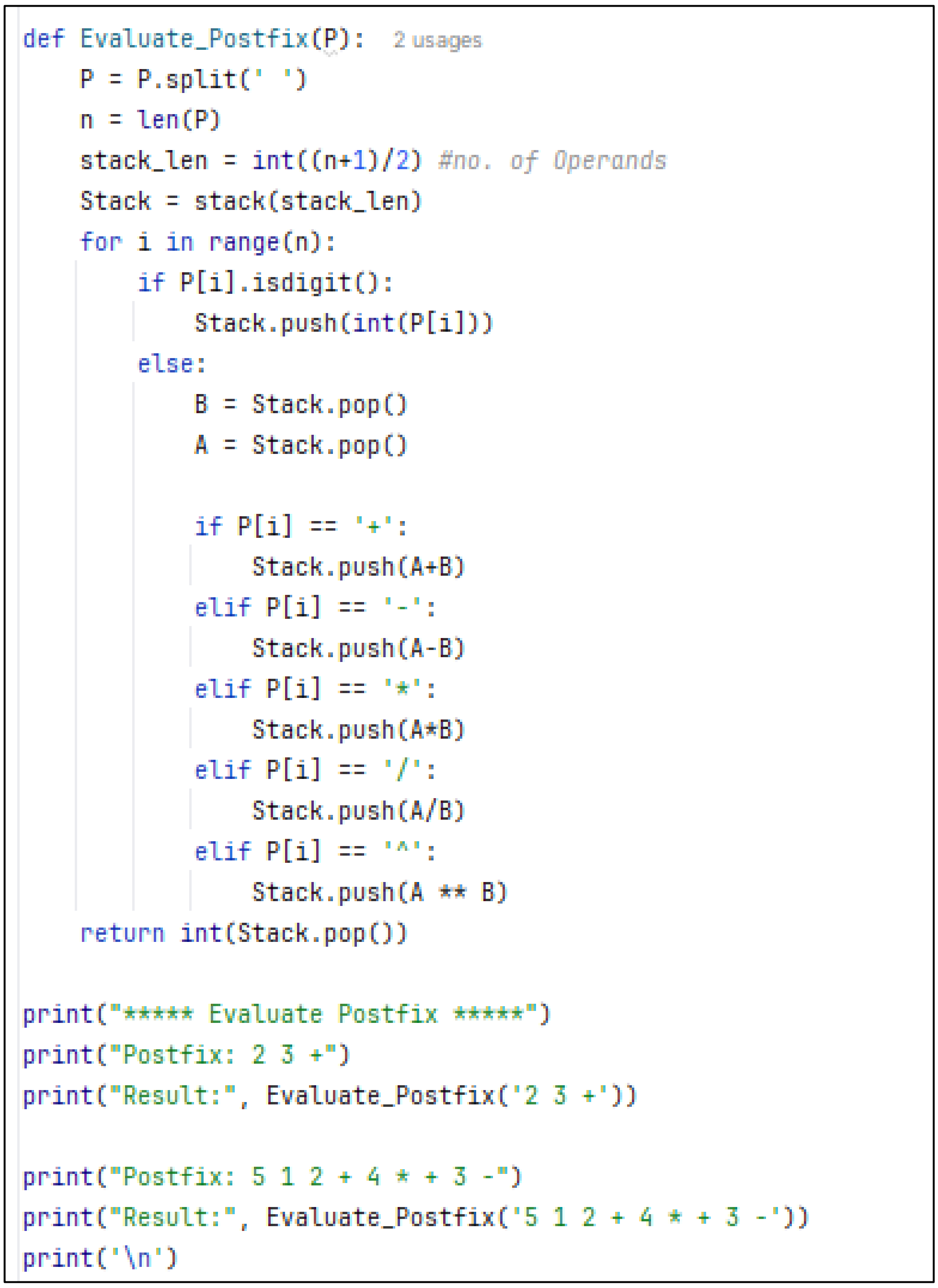
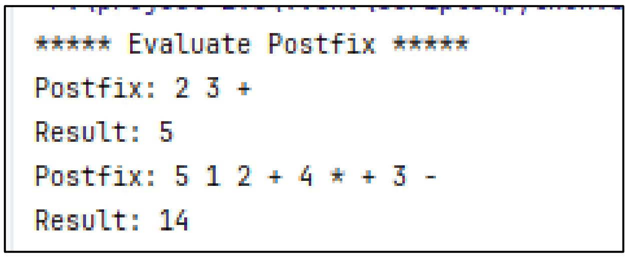
Exercise : "A"

Algorithm "A8.1"



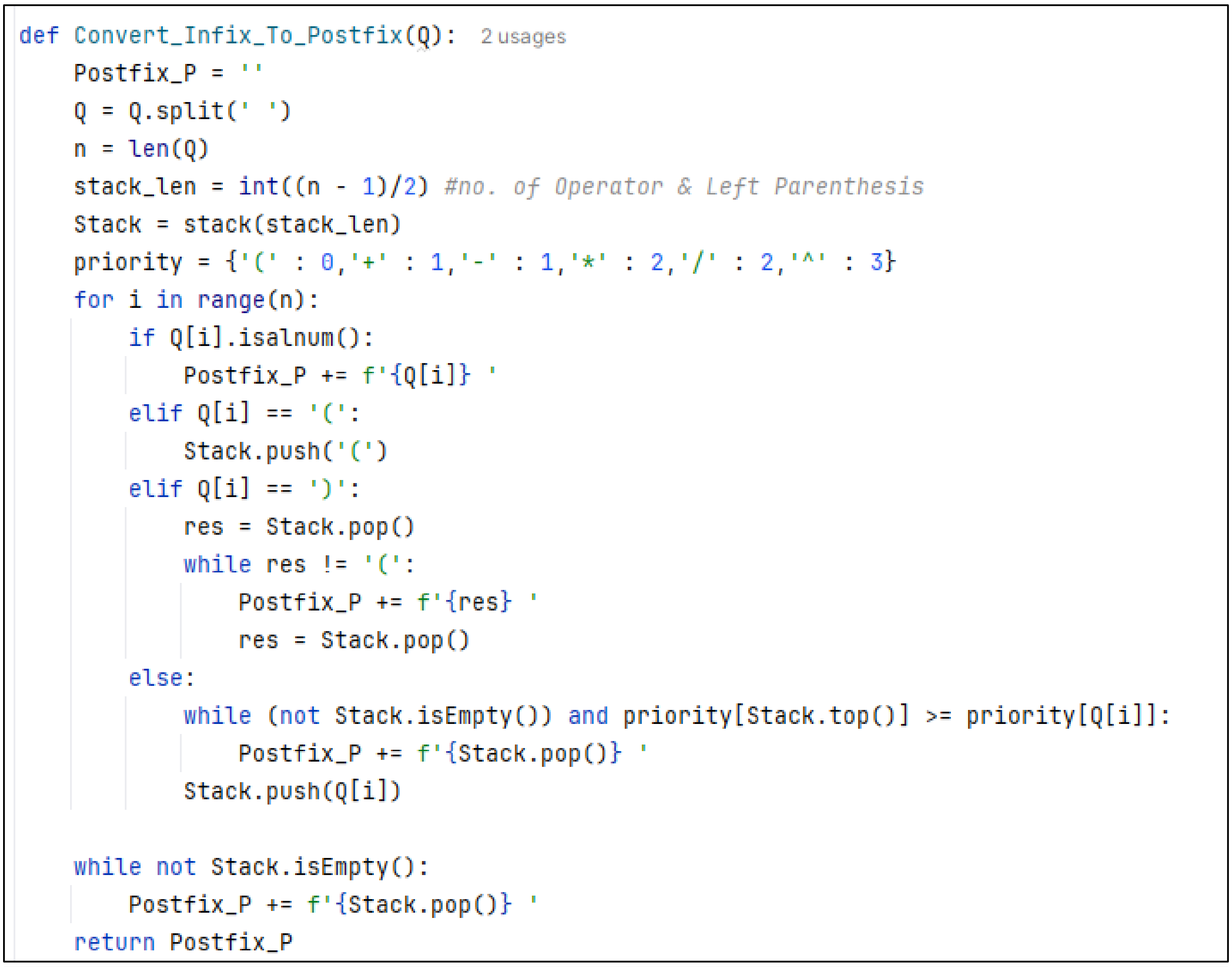
```
def Evaluate_Postfix(P):  2 usages
    P = P.split(' ')
    n = len(P)
    stack_len = int((n+1)/2) #no. of Operands
    Stack = stack(stack_len)
    for i in range(n):
        if P[i].isdigit():
            Stack.push(int(P[i]))
        else:
            B = Stack.pop()
            A = Stack.pop()

            if P[i] == '+':
                Stack.push(A+B)
            elif P[i] == '-':
                Stack.push(A-B)
            elif P[i] == '*':
                Stack.push(A*B)
            elif P[i] == '/':
                Stack.push(A/B)
            elif P[i] == '^':
                Stack.push(A ** B)
    return int(Stack.pop())

print("***** Evaluate Postfix *****")
print("Postfix: 2 3 +")
print("Result:", Evaluate_Postfix('2 3 +'))

print("Postfix: 5 1 2 + 4 * + 3 -")
print("Result:", Evaluate_Postfix('5 1 2 + 4 * + 3 -'))
print('\n')
```

"Output"

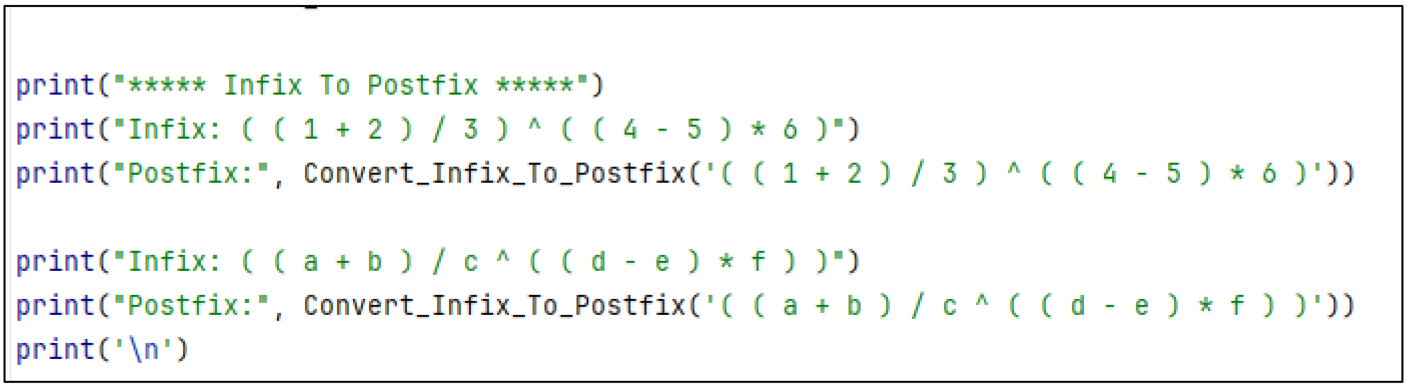


```
***** Evaluate Postfix *****
Postfix: 2 3 +
Result: 5
Postfix: 5 1 2 + 4 * + 3 -
Result: 14
```

Algorithm "A8.2"

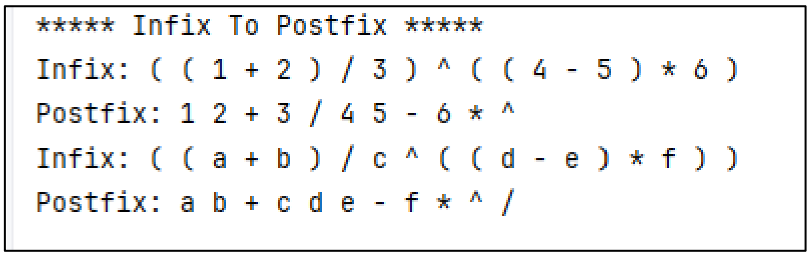


```python
def Convert_Infix_To_Postfix(Q):  2 usages
    Postfix_P = ''
    Q = Q.split(' ')
    n = len(Q)
    stack_len = int((n - 1)/2) #no. of Operator & Left Parenthesis
    Stack = stack(stack_len)
    priority = {'(' : 0,'+' : 1,'-' : 1,'*' : 2,'/' : 2,'^' : 3}
    for i in range(n):
        if Q[i].isalnum():
            Postfix_P += f'{Q[i]} '
        elif Q[i] == '(':
            Stack.push('(')
        elif Q[i] == ')':
            res = Stack.pop()
            while res != '(':
                Postfix_P += f'{res} '
                res = Stack.pop()
        else:
            while (not Stack.isEmpty()) and priority[Stack.top()] >= priority[Q[i]]:
                Postfix_P += f'{Stack.pop()} '
            Stack.push(Q[i])

    while not Stack.isEmpty():
        Postfix_P += f'{Stack.pop()} '
    return Postfix_P
```



```python
print("***** Infix To Postfix *****")
print("Infix: ( ( 1 + 2 ) / 3 ) ^ ( ( 4 - 5 ) * 6 )")
print("Postfix:", Convert_Infix_To_Postfix('( ( 1 + 2 ) / 3 ) ^ ( ( 4 - 5 ) * 6 )'))

print("Infix: ( ( a + b ) / c ^ ( ( d - e ) * f ) )")
print("Postfix:", Convert_Infix_To_Postfix('( ( a + b ) / c ^ ( ( d - e ) * f ) )'))
print('\n')
```

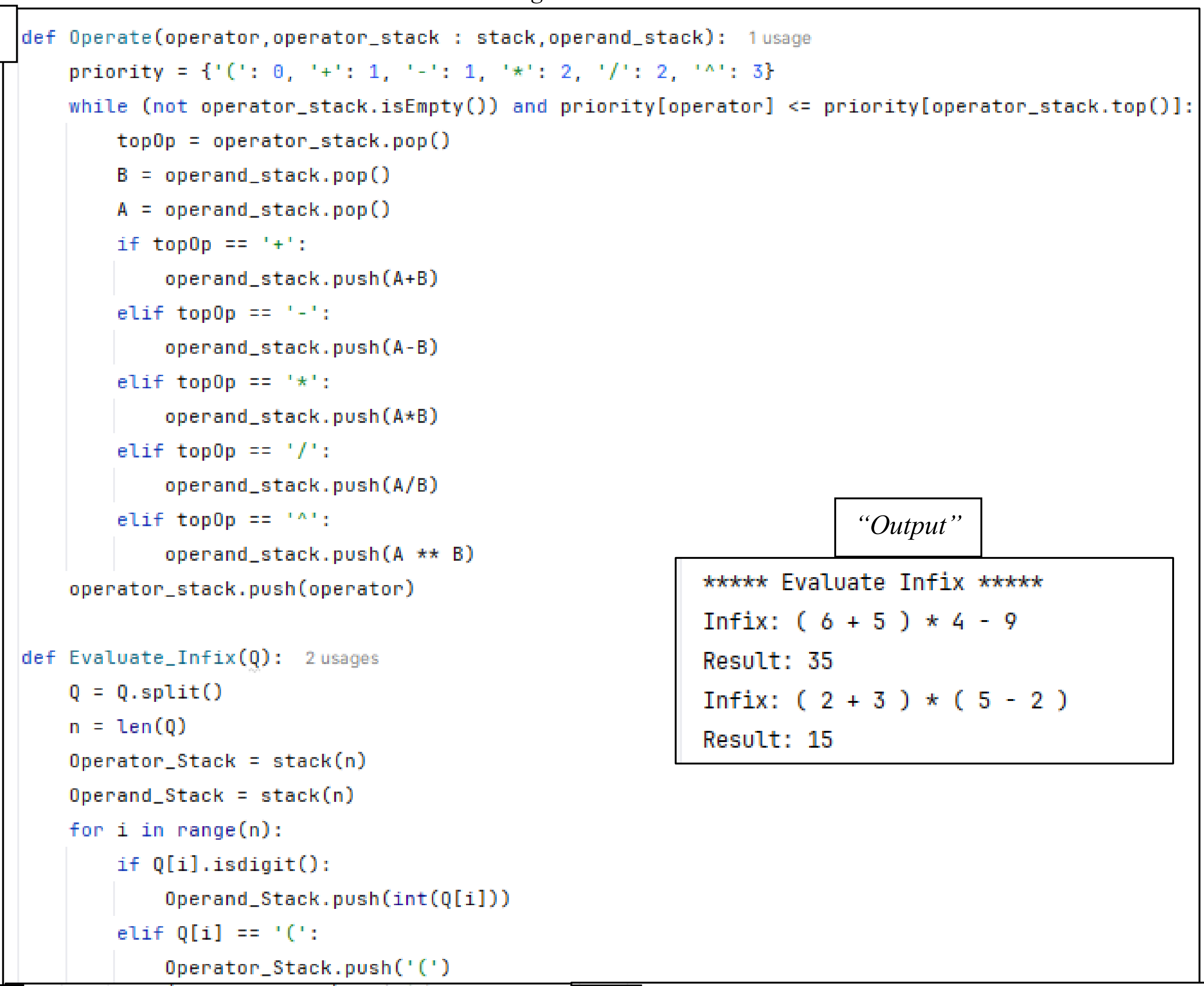
"Output"



```
***** Infix To Postfix *****
Infix: ( ( 1 + 2 ) / 3 ) ^ ( ( 4 - 5 ) * 6 )
Postfix: 1 2 + 3 / 4 5 - 6 * ^
Infix: ( ( a + b ) / c ^ ( ( d - e ) * f ) )
Postfix: a b + c d e - f * ^ /
```

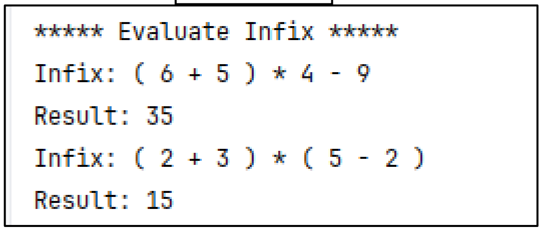
Algorithm “A8.3”

1 :



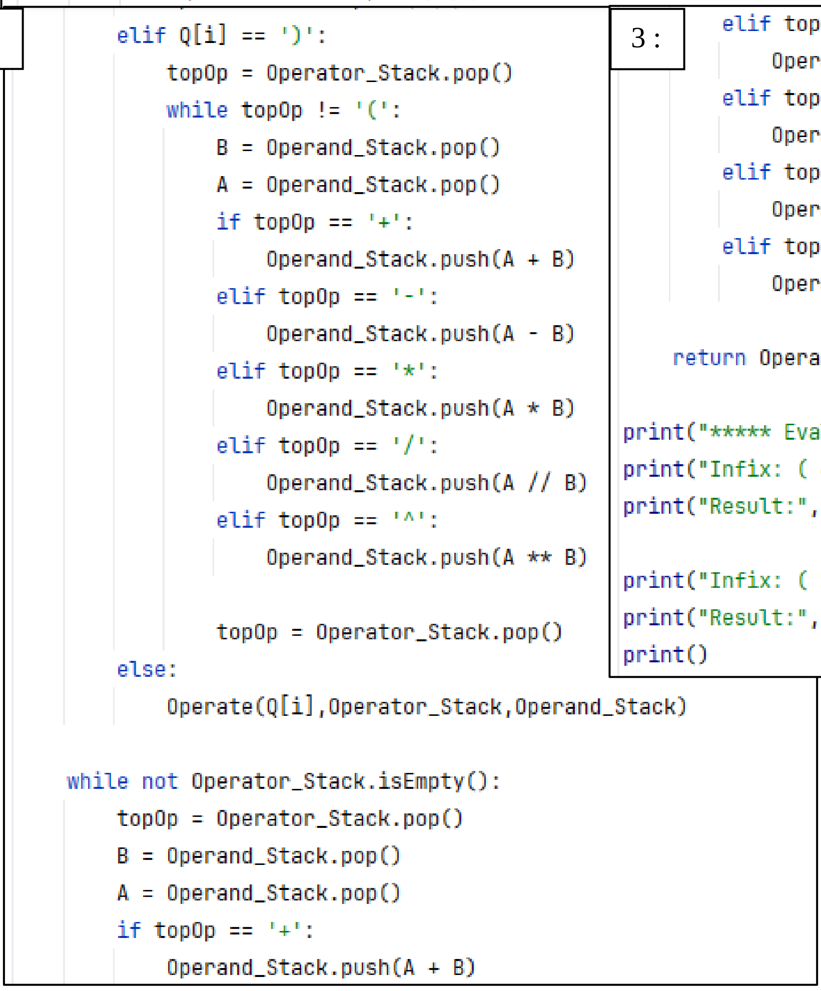
```
def Operate(operator,operator_stack : stack,operand_stack):  1 usage
    priority = {'(': 0, '+': 1, '-': 1, '*': 2, '/': 2, '^': 3}
    while (not operator_stack.isEmpty()) and priority[operator] <= priority[operator_stack.top()]:
        topOp = operator_stack.pop()
        B = operand_stack.pop()
        A = operand_stack.pop()
        if topOp == '+':
            operand_stack.push(A+B)
        elif topOp == '-':
            operand_stack.push(A-B)
        elif topOp == '*':
            operand_stack.push(A*B)
        elif topOp == '/':
            operand_stack.push(A/B)
        elif topOp == '^':
            operand_stack.push(A ** B)
    operator_stack.push(operator)

def Evaluate_Infix(Q):  2 usages
    Q = Q.split()
    n = len(Q)
    Operator_Stack = stack(n)
    Operand_Stack = stack(n)
    for i in range(n):
        if Q[i].isdigit():
            Operand_Stack.push(int(Q[i]))
        elif Q[i] == '(':
            Operator_Stack.push('(')
```

“Output”



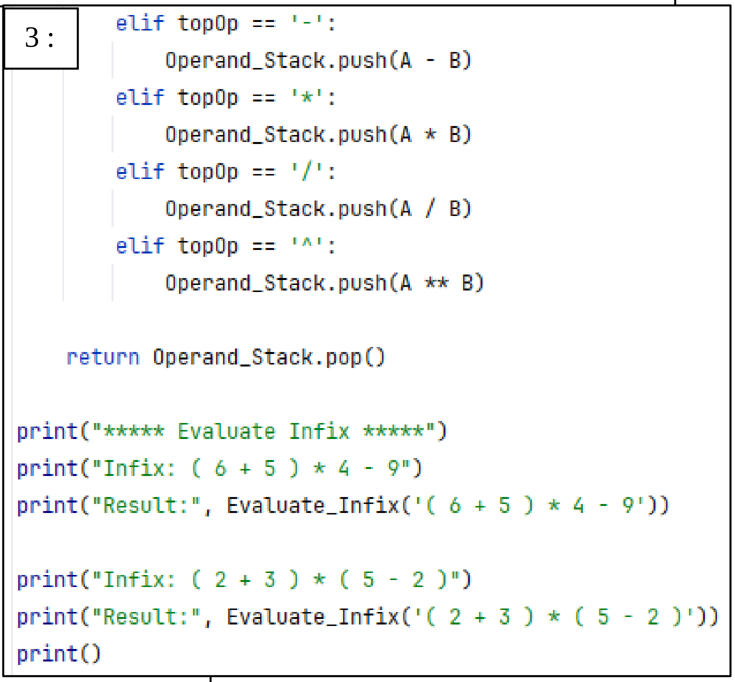
```
***** Evaluate Infix *****
Infix: ( 6 + 5 ) * 4 - 9
Result: 35
Infix: ( 2 + 3 ) * ( 5 - 2 )
Result: 15
```

2 :



```
        elif Q[i] == ')':
            topOp = Operator_Stack.pop()
            while topOp != '(':
                B = Operand_Stack.pop()
                A = Operand_Stack.pop()
                if topOp == '+':
                    Operand_Stack.push(A + B)
                elif topOp == '-':
                    Operand_Stack.push(A - B)
                elif topOp == '*':
                    Operand_Stack.push(A * B)
                elif topOp == '/':
                    Operand_Stack.push(A // B)
                elif topOp == '^':
                    Operand_Stack.push(A ** B)

                topOp = Operator_Stack.pop()
        else:
            Operate(Q[i],Operator_Stack,Operand_Stack)

    while not Operator_Stack.isEmpty():
        topOp = Operator_Stack.pop()
        B = Operand_Stack.pop()
        A = Operand_Stack.pop()
        if topOp == '+':
            Operand_Stack.push(A + B)
```

3 :



```
        elif topOp == '-':
            Operand_Stack.push(A - B)
        elif topOp == '*':
            Operand_Stack.push(A * B)
        elif topOp == '/':
            Operand_Stack.push(A / B)
        elif topOp == '^':
            Operand_Stack.push(A ** B)

    return Operand_Stack.pop()

print("***** Evaluate Infix *****")
print("Infix: ( 6 + 5 ) * 4 - 9")
print("Result:", Evaluate_Infix('( 6 + 5 ) * 4 - 9'))

print("Infix: ( 2 + 3 ) * ( 5 - 2 )")
print("Result:", Evaluate_Infix('( 2 + 3 ) * ( 5 - 2 )'))
print()
```