Problem Statement:

The ExpressionTree class should accept as input a postfix expression and output

a) the corresponding expression tree using the list format, and

b) the result of evaluating the expression tree.

Note:

The ExpressionTree class will need to be stored in ExpressionTree.py, while the generic Stack class must be stored in Stack.py

How to solve a given problem:

1) Postfix Expression - Google and find out if not sure what is a Postfix expression
2) Expression Tree – Google and found out Jenny's lecture
3) Logic/Algorithm to convert Postfix expression to Expression Tree – Another video
4) Tree Traveral and Evaluate an Expression Tree - Google
5) Design the Algorithm/solution in English

Found these Out:

1)Postfix

Postfix ---> 2 3 4 * +

2)Expression Tree

An expression (operators & Operands) in the form of a Tree (root and leaves)

3)Logic/Algorithm to covert Postfix to Expression Tree

4)Tree Traveral and Evaluate an Expression Tree - Video posted on Skype

6) **Design the Algorithm/solution in English**

Input -> postfix expression

Output →

Expression Tree - Stack with one Root node

Result of evaluating the expression tree.

Data Structures or Containers(Classes) Needed:

Stack

List → as data element

Constructor - to initialze an empty Stack

Methods in Stack:

pop()

push()

and so on.....

Expression Tree:

Methods in Expression Tree Class:

Constructor - to initialze the Expression tree

tree_eval()

Node

Data elements of Node

op – Can be the operand and Operator

left - Node

right - Node

Methods in Node Class:

Constructor - to initialize the Node Object

node_eval(self)

```
if self.op is Operator
  left = node_eval(self.left)  ---- recursive call
  right = node_eval(self.right) --- recursive call
else
  return self.op  ----- if Node's value is operand then return
end

return (The evaluation using lambda function based of op,left,right)
```

MAIN program - possible logic to use. I did not write any main as of now

```
while(true)
  print("Enter exp")
  read input
  IF No input
    BREAK
  Create expression Tree with input expression
  Evaluate Expression Tree
end Loop
```