THE UNIVERSITY OF THE WEST INDIES
Department of Computing
COMP1127–Introduction to Computing II

Lab 4

Download tree.py file from OurVLE. tree.py should be imported in your lab 4 code file (e.g. lab4.py) by using the following command.

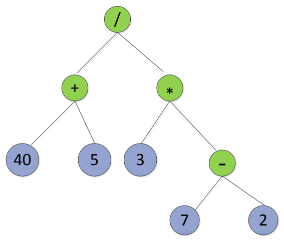
```
from tree import *
```

All functions specified in tree.py can now be used in your lab4.py

Problem1

In tree.py the tree ADT has been provided. Write a function `postorder()` which takes a tree and flattens the tree into a list.

For example, the given expression tree example `tree_ex` is included in tree.py. Flattening this tree into a list using the post order traversal would give [40, 5, '+', 3, 7, 2, '-', '*', '/']



```
>>> postorder(tree_ex)
[40, 5, '+', 3, 7, 2, '-', '*', '/']
```

Problem 2

In lab4.py implement the following functions:

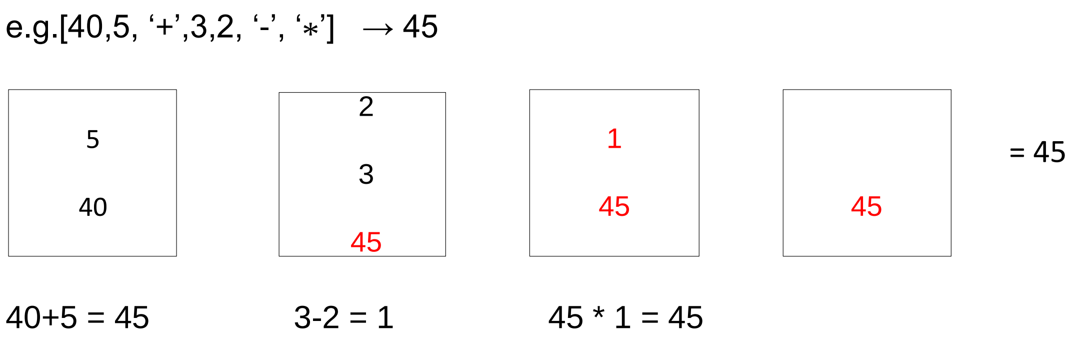
Type	**Name**	**Description**
Constructor	`stack()`	Creates a stack as a tuple, where first part of the tuple is a tag “stack” and second part of the tuple is a list.
Selector	`contents()`	Takes a stack and returns the structure where the elements in the stack are stored.
Selector	`top()`	Takes a stack and returns the element that is on top of the stack.
Predicate	`is_stack()`	Takes an object as input and returns True if the object is a tuple and the first part of the structure is a tag “stack”
Predicate	`stack_empty()`	Takes a stack and returns True if it is empty.
Mutator	`push()`	Takes a stack and an element and modifies the stack such that the element is added to the back.
Mutator	`pop()`	Takes a stack and removes the top element from the stack.

After writing the above functions, try these commands and carefully see what each one does:

```
>>> st=stack()
>>> st
>>> push(st,5)
>>> push(st,6)
>>> top(st)
>>> st
>>> pop(st)
>>> top(st)
>>> st
```

Problem 3

The `postorder()` method of flattening the example expression tree tree_ex would give [40, 5, '+', 3, 7, 2, '-', '*', '/']. A stack is often used to evaluate such an expression. While traversing through this list if an element is as operand then it is pushed onto a stack, and when an operator is found, then the stack is popped twice and the operator is evaluated with the popped value and its result is pushed back onto the stack.



1. Write a function `is_operator()` which returns True if the argument is an operator (i.e. "+", "-", "*" or "/") and False otherwise.
2. Write a function called `evalPostfix()` which takes an expression tree as an argument. It converts the expression tree to a list using a `postorder()` function. If the element in the list is a number it is added to the stack. If the element is an operator then it pops the stack twice and calls the function `apply_operator()`. This function returns a value which is pushed on the stack. When all the elements of the list have been processed then return the top of stack which would give you the result of the expression being calculated.
3. Write a function `apply_operator()` which takes three arguments, the operator, the second popped element and the first popped element and returns the result of applying the operator to the second and first popped elements. For example, `apply_operator('+',40,5)` would evaluate 40+5 and return 45.