Program 1: Programming in Python Due: April 9

An Evaluator for Logical Expressions written in Postfix Notation

You are to write a Python program that computes the value of logical expressions provided in postfix notation. For example, given the string “0!1&1!0=!1/|” (infix: “!0&1|!(!1=0)/1”), your calculator will compute “1” as the answer. All strings provided will be valid. “1” stands for true while “0” stands for false.

The logic manipulation operators are:

"!"	logical NOT	**RIGHT** associative	Highest precedence
"&"	logical AND	**LEFT** associative	
"/"	logical NOT EQUAL	**LEFT** associative	
"="	logical EQUAL	**LEFT** associative	
"|"	logical OR	**LEFT** associative	Lowest precedence

- Your calculator will use a stack to compute/store all intermediate computations.
- You will implement your own push and pop stack operations
- The **ONLY** library or built-in method that you can use is *len*

- Obtain the python 3.4 interpreter from www.python.org. Additional elaboration and requirements will be forthcoming in class.

Your program must conform to the structure and specs given on the following page.

Python file

```
PF1
PF2
tos

  push (stack, element)
  [
  ]

  pop (stack)
  [
  return element
  ]

  LogicalEval (expression)
  [
    stack definition/allocation

    Not (expr)
    [
    return value
    ]

    And  (expr1, expr2)
    [
    return value
    ]

    Not_equal(expr1, expr2)
    [
    return value
    ]

    Equal (expr1, expr2)
    [
    return value
    ]

    Or (expr1, expr2)
    [
    return value
    ]

    :
    process  expression
    :
  ]

LogicalEval (PF1)
LogicalEval (PF2)
LogicalEval ('11=0/0!&1|')
```

- PF1 and PF2 are initialized (as strings) to the following postfix expressions, respectively:
  - "0!!!1/10&1!!&=1=0|"
    (infix: "!!!0/1=1&0&!!1=1|0")
  - "10=!10/&11!0/=|11!&|&0/"
    (infix: "!(1=0)&(1/0)&((1=!1/0|1)|1&!1)/0"
- "tos" is the *global* top-of-stack variable
- the "stack" is defined/allocated in LogicalEval
- note procedure nesting
- 'process expression' examines the expression character at a time
  - If character is an operator, stack is popped, appropriate operation is called and result is pushed back onto stack
  - If character is not an operator then corresponding Boolean is pushed onto stack

  When expression evaluation is complete, stack[0] is printed (as a 0 or 1).

LogicalEval is invoked using

PF1,
PF2 and
the following string: "11=0/0!&1|"
(infix: "((1=1)/(0))&!0|1")

- You can write additional functions if you so desire.