A Stack Machine to Evaluate Expressions

Purpose: Implement and use a Stack ADT to convert infix mathematical expressions to postfix, and then evaluate the postfix expressions. Input will be from a text file, and output will be written to a file.

Stack ADT (stack.py)

You will implement a Stack ADT (class Stack) that supports the following operations:

- push(item): push an item onto the stack. Size increases by 1.
- pop(): remove the top item from the stack and return it. Raise an IndexError if the stack is empty.
- top(): return the item on top of the stack without removing it. Raise an IndexError if the stack is empty.
- size(): return the number of items on the stack.
- clear(): empty the stack.

Pseudocode For Main Program (main.py)

Pseudocode for main program:

1. open file *data.txt*
2. read an infix expression from the file
3. display the infix expression
4. call function *in2post(expr)* which you write *in2post()* takes an infix expression as an input and returns an equivalent postfix expression as a string. If the expression is not valid, raise a SyntaxError. If the parameter expr is not a string, raise a ValueError.
5. display the postfix expression
6. call function *eval_postfix(expr)* which you write *eval_postfix()* takes a postfix string as input and returns a number. If the expression is not valid, raise a SyntaxError.
7. display the result of *eval_postfix()*

Output must match the format shown in Figure 1 below.

Program Output

```
infix: 4
postfix:  4
answer: 4.0

infix: 5  +7
postfix:  5 7 +
answer: 12.0

infix: 7*5
postfix:  7 5 *
answer: 35.0

infix: (5-3)
postfix:  5 3 -
answer: 2.0

infix: 5/5
postfix:  5 5 /
answer: 1.0

infix: 8*5+3
postfix:  8 5 * 3 +
answer: 43.0

infix: 8*(5+3)
postfix:  8 5 3 + *
answer: 64.0

infix: 8+3*5-7
postfix:  8 3 5 * + 7 -
answer: 16.0

infix: (8+3)*(5-6)
postfix:  8 3 + 5 6 - *
answer: -11.0

infix: ((8+3)*(2-7))
postfix:  8 3 + 2 7 - *
answer: -55.0

infix: ((8+3)*2)-7
postfix:  8 3 + 2 * 7 -
answer: 15.0

infix: (8*5)+((3-2)-7*3)
postfix:  8 5 * 3 2 - 7 3 * - +
answer: 20.0

infix: ((8*5+3)-7)-(5*3)
postfix:  8 5 * 3 + 7 - 5 3 * -
answer: 21.0

infix: 7*9+7-5*6+3-4
postfix:  7 9 * 7 + 5 6 * - 3 + 4 -
answer: 39.0
```

Figure 1. Example Program Output

A great explanation video for these following two algorithms is found at:

https://www.youtube.com/watch?v=HJOnJU77EUs

This has a C++ flavor, but can be very helpful to Python students as well.

Evaluate a Postfix Expression

```
1. Initialize a stack
2. if next input is a number:
        Read the next input and push it onto the stack
   else:
        Read the next character, which is an operator symbol
        Use top and pop to get the two numbers off the top of the stack
        Combine these two numbers with the operation
        Push the result onto the stack
3. goto #2 while there is more of the expression to read
4. there should be one element on the stack, which is the result. Return
this.
```

Infix to Postfix Pseudocode

- ```
  1. initialize stack to hold operation symbols and parenthesis

  2. if the next input is a left parenthesis:
         Read the left parenthesis and push it onto the stack
     else if the next input is a number or operand:
         Read the operand (or number) and write it to the output
     else if the next input is an operator:
         while (stack is not empty AND
                 stack's top is not left parenthesis AND
                 stack's top is an operation with equal or higher
  precedence
                                      than the next input symbol):
             Print the stack's top
             Pop the stack's top
          Push the next operation symbol onto the stack

     else:
         Read and discard the next input symbol (should be a right
  parenthesis)
         Print the top operation and pop it
         while stack's top is not a left parenthesis:
              Print next symbol on stack and pop stack
         Pop and discard the last left parenthesis
  3. Goto #2 while there is more of the expression to read
  4. Print and pop any remaining operations on the stack
     There should be no remaining left parentheses
  ```
  ```

Test Cases

Following is the set of assertion-based test cases that your code must pass, and on which your grade for the project will be based. *You will be given the pytest unit test code to run against your code as you develop it. This allows you to learn how test are written, and to know what your score is going to be when the code is graded before you submit it.*

NOTE: THIS IS A DRAFT SET OF TEST CASES, NOT READY FOR GRADING PURPOSES

Test Empty Stack

Create an empty Stack

assert size is 0 assert pop raises IndexError

Test Nominal Stack

Create an empty Stack

push 3 items onto the stack assert size is 3. test that pop returns the last item pushed but does not alter size. Verify that the data pushed is still on the stack and accessible with top and pop. verify that top and pop will both raise an IndexError when done on an empty stack.

push 100 item onto stack and the use clear. Verify that the size after clear is zero.

Test eval_postfix (expr)

14 different postfix expressions will be fed into eval_postfix() and the proper result will be tested.
eval_postfix() will be fed an invalid post fix string and it must raise a SyntaxError.
eval_postfix() will be fed a non-string as its parameter and it must raise a ValueError

Test in2post (expr)

test 14 different in-fix equation and verify that the proper post fix
an invalid infix expression is passed into in2post() and it should raise a SyntaxError
a non-string will be passed into in2post() and it should raise a ValueError

Test program output

Test that output if main() matches output as shown in Figure 1

Test Code Quality

PyLint of both main.py and stack.py must have a value of 8.5 or greater.

Grading (100 points)

The scores are a result of the Unit Tests

- Test Stack 15
- Test in2post() 25
- Test eval_postfix() 25
- Test output 15
- Test Code Quality 20

Files to turn in through Canvas

- stack.py
- main.py