CS417 Programming Assignment #8

- Due on Monday, November 23, before midnight.
- Late penalty: Tue 5%, Wed 10%, Thu 20%, Fri 50%

Getting Started

Begin by dowloading these two files:

- `calculator.py`
- `stack.py`

Calculator

In this assignment, you will implement a small subset of the python interpreter. The user should be able to write an expression, and your program will evaluate it.

For example, here is a sample run of the program (the user's input appears after the `>>>` prompt):

```
>>> r = 10
10.0
>>> pi= 3.1416
3.1416
>>> circumference = 2*     pi *r
62.832
>>> volume=(1 + 1/3) * pi * r*r*r
4188.8
```

Specifications

The calculator should accept ordinary infix expressions with the following parts:

- Variables

  Variable names must qualify as valid python variables:
  - they must start with a letter or an underscore
  - the remaining characters (if any) may be letters, digits, or underscores

- Numbers

Numbers should not raise an error when evaluated with `float()`. However, your code does **not** have to handle floats in mantissa-exponent notation such as `123.45e-6` .

- Operators

  The single-character binary operators should be implemented: `+`, `-`, `*`, `/`, and `=` . The last one is the assignment operator:
  - its left operand (the lvalue) is a variable name,
  - its right operand (the rvalue) is a valid expression, and
  - its value is the value of the right operand.

- Parentheses

  Parentheses `()` may be used to override the rules of operator precedence.

- Whitespace

  Operators and operands may be adjacent in the infix expression, or may be separated by one or more whitespace characters, which are blanks `' '` and tabs `\t` .

Behavior

The calculator should perform a read-evaluate-print loop (REPL), until end-of-file is reached on the input:

- get an infix expression from the input
- convert the expression into an internal form: a postfix expression
- evaluate the expression
- print the result

Symbol Table

Some expressions, like `1 + 2 * 3.14` , do not change the state of the calculator. But others, like `a = 1 + 2 * 3.14` have a side effect: a variable `a` is created or modified.

To implement this functionality, your calculator will need a symbol table, which is a `dict()` that maps strings (variable names) to floats (the variable's value). This symbol table should be passed into the code that evaluates an expression.

Code to be implemented

Implement your calculator in a module called `calculator.py` . Your module MUST implement the following functions (our testing code will call these functions):

- `tokenize(line, specials, whitespace)` : This function takes a string, and returns a list of the tokens in the string.

- `line` : an ordinary string (could be an infix or postfix expression)
- `whitespace` : a string, containing characters to be treated as whitespace. Typically `whitespace == " \t"`
- `specials` : a string, containing characters to be treated as single-character tokens. Typically `specials == "+-*/()"`

Example:

```
tokens = tokenize("1 +( total* 3/ 4.56)", "=+-*/()", " \t")
print (tokens)
```

should output

```
["1", "+", "(", "total", "*", "3", "/", "4.56", ")"]
```

- `to_postfix(infix_expression)` : This function takes a string, and returns a list. The input is an infix expression, and the output is the equivalent postfix expression. The postfix expression is a list of tokens. Example:

```
postfix = to_postfix("1 +( total* 3/ 4.56)")
print (postfix)
```

should output

```
[1, 'total', 3, '*', 4.56, '/', '+']
```

- `eval_postfix(postfix_expression, symbol_table)` : This function takes a list of tokens, and returns a `float`. The input is a postfix expression, and the output is its value. This function may change the state of the symbol table, which is passed in. Example:

```
value = eval_postfix([1, 2, 3, 4, '/', '*', '+'], symbols)
print (value)
```

should output `2.5` .

Turning in your work

When you are done, go to mycourses.unh.edu, and find CS417, and the assignment. Click the "Submit" button and upload `calculator.py`.