CMPSC-122: Intermediate Programming
Summer 2018

Homework 1

Due Date: 06/01/2018, 11:59PM EST
100 pts

Instructions:

- The work in this assignment must be completed alone.
- Use the starter code provided on this CANVAS assignment. Do not change the function names or given started code on your script
- The file name must be HW1.py (incorrect name files will get a -10 point deduction)
- When any function returns an error, it must be a string containing "error"
- Do not include test code outside any function in the upload. Remove all your testing code before uploading your file.

Goal:

Write the function *calculator*(*expr*), where *expr* is a string. This function will compute the arithmetic expression given in *expr*. The arithmetic expression is a string of operands and operators that may include numeric values, four arithmetic operators (+, -, /, *) and extra spaces. An example of such expression is "-4.75   * 5   - 2.01 / 3   * 7 +   2   "
Notes:

- In the starter code provided on CANVAS, there are 4 additional functions (partially written) that will help *calculator*(*expr*) to evaluate the expression. Try to understand all the variables given in the *calculator*(*expr*) code provided.
- Except for *exeOpr*, you must code the empty segments so the five functions work completely.

Function requirements:

✓ The function must **return** the computed value if *expr* is a correct formula, otherwise it must return an error message.
✓ When any function returns a numeric value, it must be float
✓ Do not use *exec* or *eval* function. You will not receive credit if your program uses any of the two functions anywhere
✓ The five functions provided in the starter code must work

Grading Notes:

- *calculator*(*expr*) [60 pts]: The grading script will feed 4 randomly chosen test inputs, each for 15 points. One of them will be an input that should cause an error such as "4 * / 2 + 5", whose expected returned value is an error message.
- *findNextOpr*(*txt*) [20 pts]: 2 randomly chosen test inputs checking the correct returned values.
- *isNumber*(*txt*) [10 pt]: 2 randomly chosen test inputs checking the correct returned values.
- *getNextNumber*(*expr*, *index*) [10 pt]: 1 randomly chosen test input checking the correct returned value.

Deliverables:

- Include all the functions in your script named HW1.py. Submit it to the HW1 CANVAS assignment before the due date

Starter code appendix:

```
calculator(expr)
      input check
      initialization
         get the first operator and number before it
            newNumber, newOpr, oprPos by getNextNumber
         continue the initialization by if-elif-elif-…-else statements
           if newOperator is None then
                return newNumber
           elif ………

        pos  =  oprPos+1
        opr  =  newOpr

      while True:
        get netNumber, newOpr, oprPos
        if newNumber is None or …

        elif newOpr is None or …

        elif …

        elif…
```

Make a complete case analysis

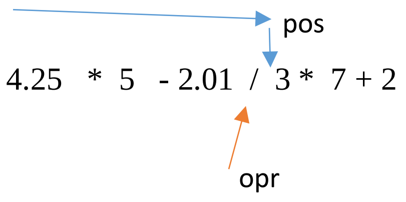
Complete case analysis: Any case is included and executed exactly once

If the maintenance of *pos* and *opr* is all the same for each case, you can do it outside the whole if-statement

If you know the current step is over, you can break or return to exit the loop

- A while True loop will execute when evaluating only valid expressions. This loop must update the values of *pos* and *opr* after an operation is performed, where:
  - pos = current position (Every time *pos* increases, it must be right after the current operator)
  - opr = the most recent operator

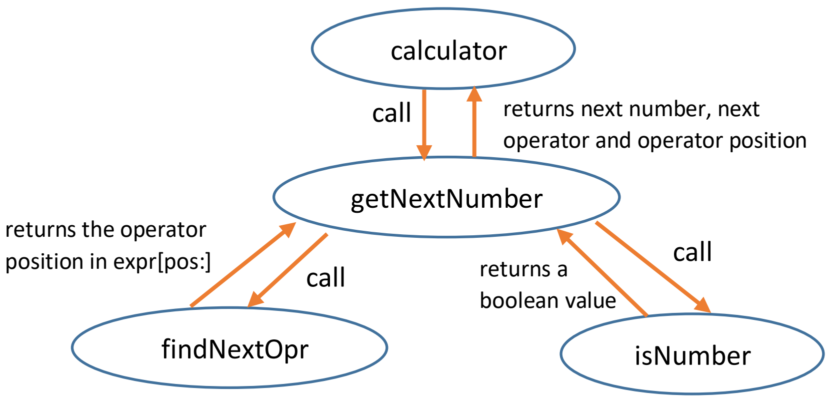
  For the example, when calling *calculator*("4.25 * 5 - 2.01 / 3 * 7 + 2"), in the next step opr="*" and pos is right after it



  The loop may exit with a break or return statement.

Overall functionality:

Your program should not go back and forth because of the operator precedence. To keep a linear time algorithm (we will learn its exact meaning later), code must scan from the left to right, and your calculation should be done in place.


calculator
call
returns next number, next operator and operator position
getNextNumber
returns the operator position in expr[pos:]
call
call
returns a boolean value
findNextOpr
isNumber

findNextOpr(txt)

- Receives expr[pos:]

    calculator(“ 3*4 - 5 ”)

        expr = “ 3*4 - 5 ”, where:

            expr[0] = space
            expr[1] = 3
            expr[2] = *
            expr[3] = 4
            expr[4] = space
            expr[5] = -
            expr[6] = space
            expr[7] = 5
            expr[8] = space

        initially, pos = 3, thus, findNextOpr receives expr[pos:] = “4 – 5 ”=txt, where:

            expr[0] = 4
            expr[1] = space
            expr[2] = -
            expr[3] = space
            expr[4] = 5
            expr[5] = space

- It returns the position of the next operator (+, -, *, /) in *txt*. In the above, it is 2
- Internal process
    - Check the positions of all the four operators.
    - If exists, return their minimum
    - Otherwise return -1
- Tip: you can use the string.find method or your own custom while loop

isNumber(txt)

- It returns True if *txt* is a string convertible to float, otherwise False. Note that “   -25.22222 “ is a string convertible to float but “   -22   33 “ and “122 ; 45” are not.
- Internal process
    - If the first character is “-“, remove it.
    - The remaining must consist of 0 to 9 and at most 1 period.
    - If so, the function should return True, otherwise False (an easy way to check if str to float is possible is with a try-except block)

getNextNumber(expr, pos)

calculator(“ 3*4 - 5 ”)

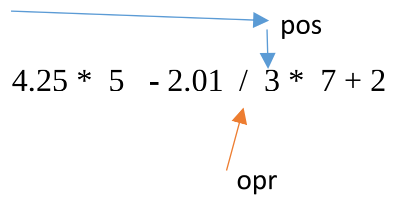
expr = “ 3*4 - 5 ”, where:

expr[0] = space
expr[1] = 3
expr[2] = *
expr[3] = 4
expr[4] = space
expr[5] = -
expr[6] = space
expr[7] = 5
expr[8] = space

initially, pos = 3

- It returns newNumber=4, newOpr=”-“ and oprPos=5
    - newOpr is the leftmost operator in *expr*[pos:]
    - oprPos is its position in *expr*, not in *expr*[pos:]
        - If no such operator, return None for both
    - newNumber is the number in expr[pos:oprPos]
        - If there is no single number in it, return None
- **It uses the functions *findNextOpr* and *isNumber* effectively**

How do you calculate everything in one linear scan on expr?



- When 4.25*5 is done, you have calculated 4.25*5=21.25. Save it.
    - Newly start 2.01/3, then multiply it by 7 to get 4.69.
    - Retrieve 21.25 and perform 21.25-4.69, repeat until you are done with the expression

Debugging is important!

- Check every function individually first
    - Input some parameters and print the returned value(s) at the bottom
- Use the Python debugger discussed on Module 2 to help you debug your code
- When checking calculator(expr), first try simple inputs such as expr = “ 2 + 3 * 4.0 ”, then gradually make it more complicated by trying “ -2.0 + 3 * 4.0 ”.
    - Use Python’s unittest module to run extensive cases on your code

Examples:

```
>>> calculator("   -4 +3 -2")
-3.0
>>> calculator("-4 +3 -2 / 2")
-2.0
>>> calculator("-4 +3   - 8 / 2")
-5.0
>>> calculator("   -4 +    3   - 8 / 2")
-5.0
>>> calculator("23 / 12 - 223 + 5.25 * 4 * 3423")
71661.91666666667
>>> calculator("2 - 3*4")
-10.0
>>> calculator("4 3 +2")
input formula error: line B in calculator
'input formula error: line B in calculator'
>>> calculator("4++ 3 +2")
error message  # message must contain the word error, all lowercase
'error message'
```