Simon Fraser University
School of Computing Science
CMPT 120, Summer 2018
Assignment 2

(Read the entire assignment, first!)

Objective

In our Assignment 2, our objective is to practise solving a problem by creating a whole Python program using what we have learnt so far, including:

1. Python building blocks such as:
   - Lists
   - Built-in functions including **split(), upper(), len()**etc.
   - **in**, **not in**, == etc. operators
   - String manipulation
   - Conditional statements (**if-elif-else**)
   - Iterative statements (**while**)
   - etc…
2. Comments
3. User interaction (input, prompts etc.)
4. Input validation and error handling code (also called "guardian code")
5. The 4 steps of the software development process.

Create a group of 2 on CourSys

1. We shall do Assignment 2 in pairs. So, let's find ourselves a partner.
   You can choose a partner from any lab section.
2. Once you have found a partner, start working on the assignment.
3. In the week 2 of your assignment once you are sure about your group member, go to CourSys and create a **"group" of 2** for assignment 2 submission. Name the group as follows: a1_firstname_firstname. For example, group name for Joe Smith and Andrew Paul should be: **a2_joe_andrew**.

   Note: If you do not want to do the assignment in pair (i.e., you would like to work on our own), you still **must** create a **"group" of 1** on CourSys for Assignment 2.

Develop a Language for Basic Arithmetic Operations - calThon

Problem Statement

In our Assignment 2, we are asked to develop a new language using Python. The new language would allow users to perform four basic arithmetic operations (+, -, * and /) with two numbers only. The calThon simulates a simple calculator, but it follows its own syntax. The input to calThon must be provided following its syntax, otherwise it shall generate an error message. The message, however, should be 'smart' enough to help the user in identifying the cause of the error. (See the **Sample Run** files **Assign2_Sample_Run_Invalid.txt** and **Assign2_Sample_Run_Valid.txt** for test cases and the exact description of the errors your program MUST generate.) Have a look at them now! Your Python program file must be called **calThon.py**.

Requirements (Requirements are the rules to which our Python program must abide.)

1. In solving this problem, we must have one loop that allow the user to use the calThon calculator until the user wants to terminate it. (see **Hint 1** below)
2. We **cannot** use the built-in functions **eval( )** or **exec()**.
3. We **cannot** use **break** or **continue** or **pass** or **sys.exit( )** in our Python program. Why? Because we want to learn how to construct conditions properly.
4. We must ask the user for the equation using **only one call** to the built-in function **input( )**. In other words, we cannot ask the user to enter operands and operators individually, calling the built-in function **input( )** for each of them.
5. We cannot have literal values assigned to our variables that represent our operands in our Python program. Operands and operators must be entered by the user.
6. Our program must print **exactly** the same output as the output represented in the Sample Runs, when the user enters **exactly** the same data (valid test data and invalid test data).

   Of course, our program will not be printing the same equation and result if the user enters a different equation (see Test Cases section below), but all the rest of the output must remain the same.
7. Finally, let's make sure that our Python program satisfies all the criteria listed in the **Marking** section below.

Test Cases (Test Data)

We must test our program using the valid and invalid test data represented in the Sample Runs. We can also use other valid and invalid test data as long as our valid test data abide to the syntax described in the Sample Runs, i.e.,

---
Welcome to my CalThon Language!

CalThon allows you to perform basic arithmetic.
The language generates a smart message if its syntax is not followed.

- add by using a statement: ADD N1 AND N2
- subtract by using a statement: SUB N2 FROM N1
- multiply by using a statement: MUL N1 BY N2
- divide by using a statement: DIV N1 BY N2

Note:
- N1 and N2 are operands - any numbers you wish.
- ADD, SUB, MUL and DIV are four operators.
- AND, FROM and BY are two keywords.
- Operators and keywords can be lowercase or mixed case.
- N2 should not be zero if the operation is DIV!

The equation you enter must follow this syntax:
<operator><space><operand><space><keyword><space><operand>
A <space> is a white space character.

_____

Enter an arithmetic statement or 'Q' or 'q' to Quit the Calculator:

Our **invalid** test data must match the invalid test data represented in the Sample Runs. This signifies that our program must have error handling code to sense and handle the following situations in which invalid data is entered by the user:

Category 1: When user enters a **single** value:

0. the user enters a string that is not a valid equation or 'Q' or 'q'.
1. the user does not enter anything (the user simply pressed the Enter key).
2. the user only enters an operator only followed by a space.
3. the user only enters a keyword only followed by a space.
4. the user only enters an operand only followed by a space.
5. the user only enters any other string followed by a space.

Category 2: When user enters **two** values:

6. the user enters two operators.
7. the user enters two keywords.
8. the user enters an operator or keyword.

9. the user enters an operator and an operand.
10. the user enters a keyword and an operand.
11. the user enters an operator and some invalid string.
12. the user enters a keyword and some invalid string.
13. the user enters two operands.
14. the user enters an operand and some invalid string.
15. the user enters two invalid strings.

Category 3: When user enters **three** values:

16. the user enters three operators.
17. the user enters three keywords.
18. the user enters three operands,
19. the user enters an operator and two operands.
20. the user enters a keyword and two operands.
21. the user enters three invalid strings.
22. the user enters two invalid strings and an operand.
23. the user enters an invalid string, a keyword and an operand.
24. the user enters an operator, and invalid string, and an operand.

Category 4: When user enters **four** values:

25. the user enters everything correctly, except uses a wrong keyword.
26. the user enters four or more incorrect values.

The bottom line

Let's make sure our Python program does not crash when the user enters any of the above invalid data. To prevent our program from crashing, we must make use of user-input validation code (a.k.a. error handling code or guardian code).

Important - About invalid test cases

There are **no** other invalid data a user can enter into our program aside from the 26 described above and illustrated in the Sample Runs (we shall not concern ourselves with other invalid test data.)

We can assume that the user will always enter an operand that is an integer, as opposed to a float. So, once we know that the operand the user has entered is not a string, we can assume that it is an integer and not a float value.

When the TA will mark our Assignment 2, she/he will only test the Assignment 2 using the valid and invalid data listed in Sample run files, nothing else.

Hint 1

In order for the user to use our Python calThon the way it is illustrated in our Sample Runs, we will need to use an iterative statement. I suggest we use this one as illustrated in the following Python code fragment:

```
# Set this variable at the top of our program
carryON = True

# We may put some code here.

# Loop until the user has entered either 'Q' or 'q'
while carryON :

  # Put most of our calculator code here.
```

Question: The loop will stop iterating when the condition carryON will become False, so when would we set carryON = False in our program?

Hint 2

When printing float data type, let's use **%g** in our print statement.

Incremental Development

I would strongly suggest we implement and test our Python program in an incremental fashion. Suggestion:

1. First, let's implement and test our Python program such that it deals with addition successfully.
2. Then, let's add code to our Python program (and test it) such that it also deals with subtraction successfully.
3. Then, let's deal with the multiplication.
4. Then, let's deal with division.
5. Then, let's add code to our Python program (and test it) such that it also deals with invalid data such as when the user enters nothing.
6. Then, let's add code to our Python program (and test it) such that it also deals with invalid data such as when the user enters "banana".
7. etc… I think we got the gist of it.

Time Management

I would strongly suggest we start Assignment 2 as early as possible, perhaps scheduling a day for each of the feature described in the **Incremental Development** section above.

Submission

- Submit our file: **calThon.py** on CourSys by **Friday, June 29** no later than 16:30.
- We can submit our work to CourSys as many times as we wish, but it is the last submission that counts. So, let's make sure that our last submission is done before the deadline.
- No late submission will be accepted: any submission made after the due date and time stated above will not be marked for grades. However, they will be marked to provide feedback to students.
    - The only exception is for medical reasons. If we are not able to submit our assignment on time due to a medical reason, we must notify the instructor by email right away and provide her with a doctor's note as soon as possible.

Marking

When marking Assignment 2, we will look at some of the following criteria:

- Are the 26 requirements (rules to which our Python program must abide – described above.) satisfied?
- Have the GPS described on our course web site been used?
- Header comment block: does it contain the name of the file, a description of the program, the name of the author and the creation date of the program?
- Is the program reading the user's equation as a string? Is the program using appropriate strings/lists functions/methods/operators in order to process the equation and evaluate it successfully?
- Code: is the code easy to read? Is it commented? Are we using **appropriate** Python building blocks, conditional and iterative statements?
- The program is a Python program and not a screenshot of the Python Interpreter Shell window.
- The program executes: are there any syntax and/or runtime errors?
- The program solves the problem: are there any semantic errors?

Have fun!