Office Hours ++ (Unit Testing and Python)

Due: Sunday, November 11th, 11:59PM (Hard Deadline)

Submission Instructions

Submit this assignment on Gradescope. You may find the free online tool PDFescape helpful to edit and fill out this PDF. You may also print, handwrite, and scan this assignment.

Adding Features to Your Python RPN Calculator

Building off of what was covered in lecture, work on using the test-driven development methodology and implement **three (3)** of the features from the list below. Note that this list is not comprehensive, so if there is a different feature that you would like to build that is non-trivial (e.g. implementing a function that would just change the sign on a number would be trivial), you may choose to build that instead.

In order to receive credit, you will need to prove the test cases for your additional functions initially fail, and then later pass when the function is fully implemented.

Feature List:

Choose **three (3)** of the following features to build. You may elect to build any number of custom features and each will count as a separate feature - you are not restricted to just one custom feature.

- ☐ **Implement additional calculator keys**

  For this feature, implement the following subfeatures:

  - ☐ Calculate percentages using the `%`. You may find this article helpful.
  - ☐ Calculate exponent using `^`.
  - ☐ Perform integer division using .
- ☐ **Implement bitwise operators (and, or and not)**
- ☐ **Implement a basic math library**

  For this feature, implement the following subfeatures:

  - ☐ Allow for usage of constants (`pi`, `e`, ...)
  - ☐ Binary functions
  - ☐ Unary functions (`sin`, `cos`, etc.)
- ☐ **Degrees and Radians Mode**

  This would follow from the above feature. Add a command/method which can set whether trigonometric operations use degrees or radians. For example, by entering the keyword `rad`, if the operation `3.1415 sin` is entered, the output would be `0`. Equivalently, if the mode is set using `deg`, if the operation `360 cos` is entered, the output would be `1`.

☐ **Use the Results of a Previous Calculation**

Add the ability to use the results of your previous calculation in the next calculation. For example, if we have the first input as `2 3 +` followed by the next command `:ans 7 +`, the output should be `12`. In order to implement this, you will need to define a language - in the example we used the colon(:) to denote this special variable

☐ **Summation**

Implement a command that find the sum of all of the elements on the stack and adds this result to the top of the stack.

☐ **Different Base Number Systems**

Add a command or method which will allow the user to use your calculator for calculations in different base number systems. For example, using an option `hex` and then entering the input `A A +` should result in the output `14`.

☐ **Factorial Operator**

Implement the factorial operator. For example, the input `4 !` should return the output `24`.

☐ **Error Handling for Division by Zero**

Implement an error handling method that perevents division by 0 errors and saves the user's existing state. For example, if the stack initually contained {`1`} and a user enters `4 0 /`, output a helpful error message and preserve the initial stack before these values were entered (i.e. {`1`}).

☐ **Session History**

Implement a command such that when it is called, it outputs the standard format of the previous operation. For example, if the previous input was `3 2 +`, calling this command will print `3 + 2 = 5`.

☐ **Convert Between Decimal and Fraction**

Implement a command to convert the item on the stack from decimal to fraction and vice-versa. For example, calling this command when the item at the top of the stack is `0.75`, the output should return `3/4`.

☐ **Repeat Operator**

Implement a repeat operator that will repeatedly carry out a provided binary operation on all items provided in the input line. For example, the input `4 2 6 * !` (where ! is the repeat operator) would result in the output `48`, regardless of whatever was previously on the stack.

☐ **Rotate**

Implement a command that will rotate the order of all items currently on the stack. For example, if the stack currently contains {`2 4 6`}, after calling this operator the stack should be {`6 4 2`}.

☐ **Copy**

Implement a command that will add a copy of the current top element to the stack. For example, if the stack currently contains {`2 4 6`}, after calling this operator the stack should be {`2 2 4 6`}.

☐ **Allow a Persistent Stack**

Make the stack for your RPN calculator persistent. For example, the input `1 2 3 +` should not produce any errors. From this, the prompt should also be customized to display information about the stack. Using the previous example, the prompt could now be customized to be `rpn calculator [1, 5]`, or something similar.

☐ **Add a Memory System**

Add a basic memory system to your calculator. This would be equivalent to the `M+`, `M-`, `MR` and `MC` on a regular calculator. This could be extended to add a near infinite amount of memory registers by defining another special character (as in the "Use the Results of a Previous Calculation). For example, these registers could be called using the `&` key: `&myval+`, `&c4cs-`, etc.

☐ **Read data in from an external file**

Add the ability to read in data from an external file for your calculator. This external file could be formatted in a style of your choice (csv, tsv, etc.).

☐ **Custom Feature!**

If there is a meaningful command not listed above and currently not already implemented by the calculator, you may choose to implement this instead. For example, you could choose to use other Python libraries (such as numbers, cmath, decimal, fractions, statistics, random, NumPy, SciPy, etc) to add new features.

Feature 1:

Test Code:

Screenshot of Failing Test:

Implementation Code:

Screenshot of Passing Test:

Feature 2:

Test Code:

Screenshot of Failing Test:

Implementation Code:

Screenshot of Passing Test:

Feature 3:

Test Code:

Screenshot of Failing Test:

Implementation Code:

Screenshot of Passing Test: