Pycalc

Mark Silvis

CS 1699 – FINAL DELIVERABLE

Summary

For my final deliverable, I developed an app using in the test drive development style: write tests, write code to pass tests, refactor. My application is a command line driven calculator written in Python 3 using the plyplus lexer and parser library. I decided to develop this app, because I thought it would be an excellent way to try writing software using TDD. A calculator is challenging enough to make the development thought-provoking, since it requires designing a grammar and configuring a parser, yet straightforward enough to break into explicit methods.

Being that my app handles mathematical calculations, many of my functions were either going to return the correct result or throw an exception, resulting in many straightforward tests ensuring proper error messages. Handling user input, on the other hand, forced me to rigorously test any methods that deal with processing input, ensuring incorrect input would not break the program.

The other difficulty was dealing with the library that I imported. I decided not to write any tests for its methods, since it's relatively popular and most likely stable enough to have to worry, especially given the scope of the project. Writing a few additional tests to ensure that the library handles my input correctly would be one area of improvement, though. In addition, since the user drives the input, this would actually be a good example of where manual testing would prove useful.

Issues

I found that writing tests before coding is not easy, since one has to have their methods rather well planned to ensure that the tests are valid and useful. Perhaps I should have planned better before beginning to test, but I found myself changing the structure of the output (especially at first), which made me also change what my tests were verifying. A major cause of this problem was the fact that Python is dynamically typed, thus, I would often have to switch types between a string and a list depending on what I had to do to transform it to valid input.

I also found that I didn't care for TDD's notion of "write code to pass the tests". It seems to me that writing in order to match the tests isn't always a good indication of matching user expectations or real-world use. It's almost as though I was coding in a bubble. It reminded me of those physics problems where you find the speed of a train, but you assume there is no friction or gravity and that the train is a perfect sphere. Writing to match tests seems good in a corporate- or development-centric world, but not the user/human one.

Assessment

There was a problem area regarding storing user-declared variables, however, that was fixed and has not appeared since. There is also a small quirk of the grammar that I noticed; it appears that exponentiation is not high enough in the order of operations. This is not a small issue, as it can result in incorrect calculations. This is a pretty serious issue, since our users will

not be verifying that the output is correct. Otherwise, there were no explicit issues regarding calculation. This app is not quite ready for release until the issue with exponentiation is fixed, but the time until deployment is short.

Code can be found at https://github.com/markalan7/pycalc

Screenshots

First test run (verbose mode enabled to show detailed output)

```
mark@UX31A:~/git/pycalc$ python3 -m unittest calc_tests.py -v
testAddToHistoryValue (calc_tests.CalcTests)
Ensures answer is correctly added to history ... ok
testAddtoHistoryKey (calc_tests.CalcTests)
Ensures history is correctly added to ... ok
testGetCommands (calc_tests.CalcTests)
Ensures command list is returned correctly ... ok
testGetConstants (calc_tests.CalcTests)
Ensures all constants are returned by get_constants ... ok
testGetHistoryNoHistory (calc_tests.CalcTests)
Ensures history is clear when no calculations have been input ... ok
testGetHistoryWithHistory (calc_tests.CalcTests)
Ensures history returns correct values when calculations have been input ... ok
testGetPrevious (calc_tests.CalcTests)
Ensures get_previous returns last calculation ... ok
testGetPreviousWithAnswer (calc_tests.CalcTests)
Ensures get_previous returns the answer to the last calculation when requested ... ok
testGetVariablesNoVariables (calc_tests.CalcTests)
Ensures no variables are returned by get_variables if none have been added ... ok
testGetVariablesWithVariables (calc_tests.CalcTests)
Ensures all variables are returned by get_variables after variables have been added ... ok
testSetAnswer (calc_tests.CalcTests)
Ensures previous answer constant gets set ... ok
testSetVariable (calc_tests.CalcTests)
Ensures that variables can be set correctly ... ok
testSetVariableInvalidKey (calc_tests.CalcTests)
Ensures only a valid variable may be added (an identifier) ... ok
testSetVariableInvalidValue (calc_tests.CalcTests)
Ensures only a valid value may be added (a number, integer or decimal) ... ok
testSetVariableKeyInVariablesDict (calc_tests.CalcTests)
Ensures setting variable adds key to variables dict ... ok
testSetVariableNoCommandOverride (calc_tests.CalcTests)
Ensures set_variable does not override commands ... ok
testSetVariableNoConstantOverride (calc_tests.CalcTests)
Ensures set_variable does not override constants ... ok
testSetVariableValueInVariablesDict (calc_tests.CalcTests)
Ensures setting variable addes value to variables dict ... ok

----------------------------------------------------------------------
Ran 18 tests in 0.003s

OK
```

Second test run

One failure occurred

```
mark@UX31A:~/git/pycalc$ python3 -m unittest calc_tests.py
.....................F....
======================================================================
FAIL: testSetVariableInvalidValueLong (calc_tests.CalcTests)
Ensures only a valid value may be added (integer or decimal)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/mark/git/pycalc/calc_tests.py", line 111, in testSetVariableInvalidValueLong
    self.assertEqual(actual, expected)
AssertionError: 't set to 9324u2389423723742384' != 'Error: Value must be an integer or decimal'
- t set to 9324u2389423723742384
+ Error: Value must be an integer or decimal


----------------------------------------------------------------------
Ran 27 tests in 0.004s

FAILED (failures=1)
```

Third test run (output was too large, so I had to copy and paste)

Verbose mode enabled to show detailed output

Testing occurred after refactor

python3 -m unittest calc_tests.py -v
testAddToHistoryValue (calc_tests.CalcTests)
Ensures answer is correctly added to history ... ok
testAddtoHistoryKey (calc_tests.CalcTests)
Ensures history is correctly added to ... ok
testCalculateReturnsCorrectAdd (calc_tests.CalcTests)
Ensures calculation is correct for addition ... ok
testCalculateReturnsCorrectSub (calc_tests.CalcTests)
Ensures calculation is correct for subtraction ... ok
testCheckEqualParensEqual (calc_tests.CalcTests)
Ensures method returns true when input has equal number of left and right parentheses ... ok
testCheckEqualParensNotEqual (calc_tests.CalcTests)
Ensures method returns false when input has inequal number of left and right parentheses ...
ok
testCheckForVariableValidAnswer (calc_tests.CalcTests)
Ensures the previous answer constant is added ... ok
testCheckForVariableValidConstant (calc_tests.CalcTests)
Ensures that constants are replaced with their appropriate value ... ok
testCheckForVariablesDoesNotExist (calc_tests.CalcTests)

If input contains a variable or constant, it replaces it with its value ... ok
testCheckForVariablesValidVariable (calc_tests.CalcTests)
If input contains a variable, it should be replaced with its value ... ok
testGetCommands (calc_tests.CalcTests)
Ensures command list is returned correctly ... ok
testGetConstants (calc_tests.CalcTests)
Ensures all constants are returned by get_constants ... ok
testGetHistoryNoHistory (calc_tests.CalcTests)
Ensures history is clear when no calculations have been input ... ok
testGetHistoryWithHistory (calc_tests.CalcTests)
Ensures history returns correct values when calculations have been input ... ok
testGetPrevious (calc_tests.CalcTests)
Ensures get_previous returns last calculation ... ok
testGetPreviousWithAnswer (calc_tests.CalcTests)
Ensures get_previous returns the answer to the last calculation when requested ... ok
testGetVariablesNoVariables (calc_tests.CalcTests)
Ensures no variables are returned by get_variables if none have been added ... ok
testGetVariablesWithVariables (calc_tests.CalcTests)
Ensures all variables are returned by get_variables after variables have been added ... ok
testSetAnswer (calc_tests.CalcTests)
Ensures previous answer constant gets set ... ok
testSetVariable (calc_tests.CalcTests)
Ensures that variables can be set correctly ... ok
testSetVariableInvalidKey (calc_tests.CalcTests)
Ensures only a valid variable may be added (an identifier) ... ok
testSetVariableInvalidValue (calc_tests.CalcTests)
Ensures only a valid value may be added (a number, integer or decimal) ... ok
testSetVariableInvalidValueLong (calc_tests.CalcTests)
Ensures only a valid value may be added (integer or decimal) ... ok
testSetVariableKeyInVariablesDict (calc_tests.CalcTests)
Ensures setting variable adds key to variables dict ... ok
testSetVariableNoCommandOverride (calc_tests.CalcTests)
Ensures set_variable does not override commands ... ok
testSetVariableNoConstantOverride (calc_tests.CalcTests)
Ensures set_variable does not override constants ... ok
testSetVariableValueInVariablesDict (calc_tests.CalcTests)
Ensures setting variable addes value to variables dict ... ok

---

Ran 27 tests in 0.005s

OK

Fourth test run
Added additional tests, mostly to ensure that the correct answer is output

```
python3 -m unittest calc_tests.py -v
testAddToHistoryValue (calc_tests.CalcTests)
Ensures answer is correctly added to history ... ok
testAddtoHistoryKey (calc_tests.CalcTests)
Ensures history is correctly added to ... ok
testCalculateReturnsCorrectAdd (calc_tests.CalcTests)
Ensures calculation is correct for addition ... ok
testCalculateReturnsCorrectComplex (calc_tests.CalcTests)
Ensures calculation is correct for a more complex calculation ... ok
testCalculateReturnsCorrectConstant (calc_tests.CalcTests)
Ensures calculation is correct with constant ... ok
testCalculateReturnsCorrectDiv (calc_tests.CalcTests)
Ensures calculation is correct for division ... ok
testCalculateReturnsCorrectModulo (calc_tests.CalcTests)
Ensures calculation is correct for modulus ... ok
testCalculateReturnsCorrectMult (calc_tests.CalcTests)
Ensures calculation is correct for multiplication ... ok
testCalculateReturnsCorrectPower (calc_tests.CalcTests)
Ensures calculation is correct for eponentiation ... ok
testCalculateReturnsCorrectSub (calc_tests.CalcTests)
Ensures calculation is correct for subtraction ... ok
testCalculateReturnsCorrectVariable (calc_tests.CalcTests)
Ensures calculation is correct with variable ... ok
testCalculateReturnsCorrectVeryComplex (calc_tests.CalcTests)
Ensures calculation is correct with very complex calculation ... ok
testCheckEqualParensEqual (calc_tests.CalcTests)
Ensures method returns true when input has equal number of left and right parentheses ... ok
testCheckEqualParensNotEqual (calc_tests.CalcTests)
Ensures method returns false when input has inequal number of left and right parentheses ...
ok
testCheckForVariableValidAnswer (calc_tests.CalcTests)
Ensures the previous answer constant is added ... ok
testCheckForVariableValidConstant (calc_tests.CalcTests)
Ensures that constants are replaced with their appropriate value ... ok
testCheckForVariablesDoesNotExist (calc_tests.CalcTests)
If input contains a variable or constant, it replaces it with its value ... ok
testCheckForVariablesValidVariable (calc_tests.CalcTests)
If input contains a variable, it should be replaced with its value ... ok
testGetCommands (calc_tests.CalcTests)
Ensures command list is returned correctly ... ok
testGetConstants (calc_tests.CalcTests)
```

Ensures all constants are returned by get_constants ... ok
testGetHistoryNoHistory (calc_tests.CalcTests)
Ensures history is clear when no calculations have been input ... ok
testGetHistoryWithHistory (calc_tests.CalcTests)
Ensures history returns correct values when calculations have been input ... ok
testGetPrevious (calc_tests.CalcTests)
Ensures get_previous returns last calculation ... ok
testGetPreviousWithAnswer (calc_tests.CalcTests)
Ensures get_previous returns the answer to the last calculation when requested ... ok
testGetVariablesNoVariables (calc_tests.CalcTests)
Ensures no variables are returned by get_variables if none have been added ... ok
testGetVariablesWithVariables (calc_tests.CalcTests)
Ensures all variables are returned by get_variables after variables have been added ... ok
testSetAnswer (calc_tests.CalcTests)
Ensures previous answer constant gets set ... ok
testSetVariable (calc_tests.CalcTests)
Ensures that variables can be set correctly ... ok
testSetVariableInvalidKey (calc_tests.CalcTests)
Ensures only a valid variable may be added (an identifier) ... ok
testSetVariableInvalidValue (calc_tests.CalcTests)
Ensures only a valid value may be added (a number, integer or decimal) ... ok
testSetVariableInvalidValueLong (calc_tests.CalcTests)
Ensures only a valid value may be added (integer or decimal) ... ok
testSetVariableKeyInVariablesDict (calc_tests.CalcTests)
Ensures setting variable adds key to variables dict ... ok
testSetVariableNoCommandOverride (calc_tests.CalcTests)
Ensures set_variable does not override commands ... ok
testSetVariableNoConstantOverride (calc_tests.CalcTests)
Ensures set_variable does not override constants ... ok
testSetVariableValueInVariablesDict (calc_tests.CalcTests)
Ensures setting variable addes value to variables dict ... ok

---

Ran 35 tests in 0.009s

OK