Sugar Labs

CSCI 362.2

Aaron Gaynor, Jake Hennessy, Aaron Walton, and Sabrina Yongue

College of Charleston

SugarLabs Intro

SugarLabs is software (written in python) that is used as a learning tool for children studying science and math. For our project we decided to test the calculator, which consists of a GUI that performs the operations on the data which is input by the user. To test the actual functions, we had to extract the functions.py and rational.py files.

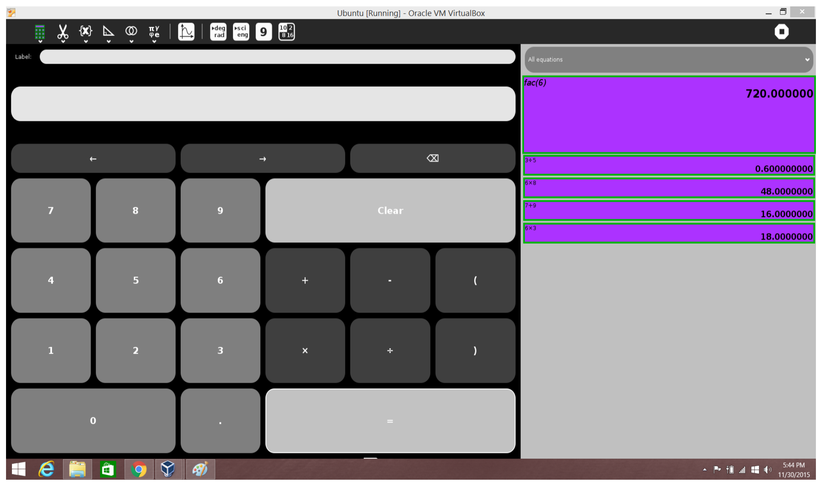

Figure 1. Screenshot of SugarLabs calculate GUI.

Functions.py and Rational.py

The files we decided to test were arithmetic functions from the SugarLab package including: add() [addition], sub() [subtraction], mul() [multiplication], div() [division], factorial() [factorial]. These functions are components of the calculate.py package. Essentially in each function the code goes and grabs instances of the data contained within the test files, and returns the calculated value. Rational.py allows data that is input to be interpreted as a rational number by the machine.

```
def add(x, y):
    if isinstance(x, _Decimal) or isinstance(y, _Decimal):
        x = _d(x)
        y = _d(y)
    return x + y
add.__doc__ = _('add(x, y), return x + y')
```

Figure 2. Source code from the addition function in calculate.py .

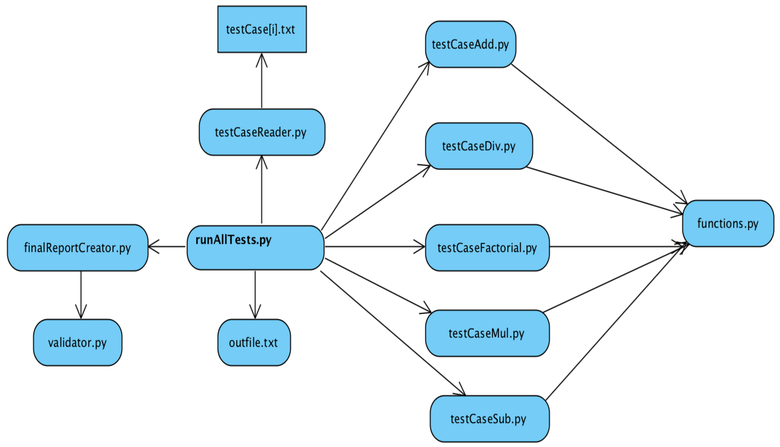

Figure 3. Diagram showing the graphical representation of the testing framework.

As it turns out, Functions.py only has a dependency upon one other file in the system, Rational.py

Rational.py has no dependencies, and so both functions.py and rational.py were removed. After we were able to isolate the files in question, a testing framework could be created.

runAllTests.py

```
from TestCaseAdd import testAdd
from TestCaseDiv import testDiv
from TestCaseMul import testMul
from TestCaseSub import testSub
from TestCaseFactorial import testFactorial

currentworkingdirectory = currentworkingdirectory.replace('/testCasesExecutables', '')

for i in range(1, (numberTestCases + 1)):

        #get current test case name & path
        currentTestCase = currentworkingdirectory + '/testCases/testCase' + str(i)
        currentOutFile = currentworkingdirectory + '/temp/testCaseOutput' + str(i)

        #get method to be tested
        testCasei = open((currentTestCase + '.txt'), 'r')
        testCaseContents = testCasei.read()
        testCaseLines = testCaseContents.split('\n')
        methodToTest = testCaseLines[3]
        methodToTest = methodToTest.replace('Method: ', '')

        if(methodToTest == 'add'):
                x, y = testCaseExtractor(currentTestCase)
                result = testAdd(x, y)
                intToText(currentOutFile, result)
        elif(methodToTest == 'mul'):
                x, y = testCaseExtractor(currentTestCase)
                result = testMul(x, y)
                intToText(currentOutFile, result)
        elif(methodToTest == 'div'):
                x, y = testCaseExtractor(currentTestCase)
                result = testDiv(x, y)
                intToText(currentOutFile, result)
        elif(methodToTest == 'sub'):
                x, y = testCaseExtractor(currentTestCase)
                result = testSub(x, y)
                intToText(currentOutFile, result)
        elif(methodToTest == 'factorial'):
                x = testCaseExtractor(currentTestCase)
                result = testFactorial(x)
                intToText(currentOutFile, result)
        else:
                print("INCORRECT METHOD TYPE: TEST CASE " + i)

reportUrl = createReport(numberTestCases)
webbrowser.open_new_tab(reportUrl)
```

Figure 4. runAllTests.py screenshot showing how input is obtained and files are naviigated.

Testing Framework

The framework is designed to allow the file runAllTests.py to extract data from the test case input files, and pass them into the test case reader. Next those values are used as arguments for the various functions that will be tested. Finally those values get compared to the actual (correct) values and all of the data (methods, test case number, pass/fail, etc.)

file:///hom...Report.html

file:///home/aaron/Desktop/TeamUnknownTestCaseSwitch/TestAutomation/reports/finalReport.html

TestId:	Requirement:	Component:	Method:	Input:	Expected Outcome:	Actual Outcome:	Test Results:
1	add handles positive integers	Functions	add	10 10	20	20	Passed
25	factorial handles negative integer	Functions	factorial	-3	-6	-3	Failed
3	add handles a negative and positive integer	Functions	add	10 -10	0	0	Passed
11	mul handles positive integers	Functions	mul	10 10	100	100	Passed
5	add handles two negative integers	Functions	add	-10 -10	-20	-20	Passed
6	sub handles positive integers	Functions	sub	10 5	5	5	Passed
7	sub handles a positive integer and zero	Functions	sub	10 0	10	10	Passed
8	sub handles zero and a positive integer	Functions	sub	0 10	-10	-10	Passed
20	div handles a positive and negative integer	Functions	div	10 -10	-1	-1	Passed
10	sub handles negative integers	Functions	sub	-10 -10	0	0	Passed
17	div handles positive integers	Functions	div	10 1	10	10	Passed
12	mul handles a positive and negative integer	Functions	mul	10 -10	-100	-100	Passed
22	factorial handles positive integer	Functions	factorial	8	40320	40320	Passed
14	mul handles a negative integer and zero	Functions	mul	-10 0	0	0	Passed
15	mul handles a positive integer and zero	Functions	mul	10 0	0	0	Passed
16	div handles positive integers	Functions	div	10 10	1	1	Passed
4	add handles a negative integer and zero	Functions	add	-10 0	-10	-10	Passed
18	div handles positive integer and zero	Functions	div	10 0	ERROR	ERROR	Passed
19	div handles positive integers	Functions	div	10 1	10	10	Passed
9	sub handles a positive and negative integer	Functions	sub	10 -10	20	20	Passed
21	factorial handles positive integer	Functions	factorial	1	1	1	Passed
13	mul handles negative integers	Functions	mul	-10 -10	100	100	Passed
23	factorial handles positive integer	Functions	factorial	3	6	6	Passed
24	factorial handles zero	Functions	factorial	0	1	1	Passed
2	add handles a positive integer and zero	Functions	add	10 0	10	10	Passed

Figure 5. Output table in html containing test data.

Conclusions

The overall functions and implementing them were the easy portion of the project, but the coordination of the group, and the communication between members in order to have tasks accomplished was the most rewarding part of this assignment. After numerous clarifications made on requirements we had to alter several files in the program, centering mostly in the efficient navigation between files in our system and the numbering of test cases. In total, this project was an effective exercise in team planning and requirements gathering

References

SugarLabs, https://www.sugarlabs.org .