Team 6 Deliverable 4

Experience:

The primary complication in finishing the test case suite was realizing the limitations of our existing testing framework, which we had initially assumed would be sufficient to easily implement the rest of the test cases. As it was however the framework could only take in arguments as strings, which was rarely a problem in testing Jython's string and url parsing functionalities, but needed to be expanded upon in order to work with numerical inputs as well. As such the formatting for our test cases was expanded upon to include “” to designate strings, as well as an escape character ($,) to differentiate commas used in strings from commas used to separate multiple arguments to a function. It was also noted that there is some inconvenience in testing functions which return lists and tuples, in that the expected output line of the test case must be input as a string precisely as python outputs the results to a string rather than iterating through the resultant list.

Nevertheless, in addition to implementing the additional test cases, the deliverable left us necessarily producing a more fleshed out and useful testing framework, shown below.

Script:

```
      #!/usr/bin/python

from sys import platform as _platform
import subprocess
import os.path
import sys
from importlib import import_module

def getTestCases():
    os.chdir("../testCases/")
    allFiles = os.listdir(".")
    return [TestCase(fileName) for fileName in allFiles if ".txt" in
fileName and fileName[-1] != '~']

def executeTest(testCase):
    importStatement = convertPathToImport(testCase.modulePath)
    sys.path.append(testCase.modulePath)
    module = __import__(importStatement, fromlist=[testCase.methodName])
    #Just gets method name itself, not parameters or other signature
    methodNameTrimmed =
testCase.methodName[0:testCase.methodName.find('(')]
    methodToTest = getattr(module, methodNameTrimmed)
    result = methodToTest(*testCase.inputValue)
    testCase.actualResult = result

    #actual test
    if str(result) == testCase.expectedResult:
        testCase.testPassed = True
    return testCase

def convertPathToImport(path):
    path = path.replace("/", ".")
    path = path.replace(".py","")
    return path.split(".")[-1]
```

```
class TestCase:
    def __init__(self, fileName):
        file = open(fileName, 'r')
        fileContents = file.read().split("\n")
        self.id = fileContents[0]
        self.description = fileContents[1]
        self.modulePath = fileContents[2].strip()
        self.methodName = fileContents[3].strip()
        self.inputValue = fileContents[4].split('$,')
        #go through parameters and pluck out strings vs numbers
        for i in range(len(self.inputValue)):
            if('"' in self.inputValue[i]):
                self.inputValue[i] = self.inputValue[i][1:-1]
            else: #else it's a number
                self.inputValue[i] = eval(self.inputValue[i])
        self.expectedResult = fileContents[5]

        self.actualResult = ""
        self.testPassed = False

        file.close()


def clearTempFolder():
    #go back a directory, remove everyting from temp,
    #if nothing is in temp, the warning is suppressed by sending it to null
    subprocess.call("rm ../temp/* 2>/dev/null", shell=True)

def generateHtml(testResults):
    html = "<html>"
    html += "<body>"
    html += "<h1>Test Results</h1>"
    html += testResults
    html += "</body>"
    return html + "</html>"

def writeHtmlFile(html):
    os.chdir("..")
    os.chdir("./reports/")
    filename = "test_results.html"
    output = open(filename,'w')
    output.write(html)
    if _platform == "linux" or _platform == "linux2":
        # linux:
        subprocess.call("xdg-open " + filename, shell=True)
    elif _platform == "darwin":
        # OS X:
        subprocess.call("open " + filename, shell=True)


def main():
    clearTempFolder()
```

```
    testCases = getTestCases()
    testCases.sort(key= lambda x: eval(x.id))
    outputString = ""
    for testCase in testCases:
        executeTest(testCase)
        if testCase.testPassed:
            outputString += '<p style = "color:green">'
            outputString += "Test case "+testCase.id+" passed!"
            outputString += '</p>'
        else:
            outputString += '<p style = "color:red">'
            outputString += "Test case "+testCase.id+" FAILED!"
            outputString += '</p>'
        outputString += "<ul>"
        outputString += "<li>\tInputs: " + str(testCase.inputValue) +
"</li>"
        outputString += "<li>\tExpected Result: " + testCase.expectedResult
+ "</li>"
        outputString += "<li>\tActual Output: " +
str(testCase.actualResult) + "</li>"
        outputString += "</ul>"
    htmlBody = generateHtml(outputString)
    writeHtmlFile(htmlBody)


main()
```

Test Cases:

1
The upper method shall take in a string and return that same string entirely in upper case letters
jython/lib-python/2.7/string.py
upper(s)
"caT"
CAT

2
The strip method shall take in a string and return that same string without leading or trailing whitespaces
jython/lib-python/2.7/string.py
strip(s)
" this is a test for trim "
this is a test for trim

3
The lower method shall take in a string and return that same string entirely in lower case letters
jython/lib-python/2.7/string.py
lower(s)
"CAT"
cat

4
The swapcase method shall take in a string and return the same string with lower case letters capitalized and visa versa
jython/lib-python/2.7/string.py
swapcase(s)
"teSTING fOr Cases"
TEsting FoR cASES

5
The capwors method shall take in a string, and return that string with all of the words capitalized
jython/lib-python/2.7/string.py
capwords(s)
"this iS a tEst."
This Is A Test.

6
The unquote method shall decode an encoded parameter of a url, replacing all %20 with spaces
jython/lib-python/2.7/urlparse.py
unquote(s)
"abc%20def"
abc def

7
The replace method takes in a string, and replaces all instances of a given substring with instances of a second given substring
jython/lib-python/2.7/string.py
replace(s, old, new)
"here is a test"$,"test"$,"TEST"
here is a TEST

8
The replace method takes in a string, and replaces all instances of a given substring with instances of a second given substring
jython/lib-python/2.7/string.py
replace(s, old, new)
"she sells seashells by the seashore"$,"she"$,""
 sells sealls by the seashore

9
Return the lowest index in s where substring sub is such that sub is contained within s[start,end]. arguments start and end are interpreted as in slice notation.
jython/lib-python/2.7/string.py
find(s, *args)
"looking for the first instance of the letter g"$,"g"
6

10
The count method takes in a string and returns a sum of all instances of a given substring occuring within that string
jython/lib-python/2.7/string.py
count(s, *args)
"example string with a good   number of characters in it"$," "
12

11
Return the lowest index in s where substring sub is such that sub is contained within s[start,end]. arguments start and end are interpreted as in slice notation.
jython/lib-python/2.7/string.py
find(s, *args)
"the letter after p will not be found in this string"$,"q"
-1

12
The gcd method takes two integers returns the greatest common denominator of the two integers given, for use by Jython's fraction functions
jython/lib-python/2.7/fractions.py
gcd(a,b)
12$,18
6

13
Return a right-justified version of s, in a field of specified width, padded with spaces as needed. The string is never truncated. If specified the fillchar is used instead of spaces.
jython/lib-python/2.7/string.py
rjust(s, width, *args)
"test"$,7
   test

14
The atoi method takes in a string representing an integer in a given base and returns the integer in base 10
jython/lib-python/2.7/string.py
atoi(s,base=10)
"00101100"$,2
44

15
The atoi method takes in a string representing an integer in a given base and returns the integer in base 10
jython/lib-python/2.7/string.py
atoi(s,base=10)
"1BC"$,16
444

16
The expandtabs method, given two arguments, will change the number of whitespaces that \t expands to when processing the given text
jython/lib-python/2.7/string.py
expandtabs(s, tabsize=8)
"hello	world"$,1
hello world

17
Return a left-justified version of s, in a field of specified width, padded with spaces as needed. The string is never truncated. If specified the fillchar is used instead of spaces.
jython/lib-python/2.7/string.py
ljust(s, width, *args)
"test"$,7

test

18
Pad a numeric string x with zeros on the left, to fill a field of the specified width. The string x is never truncated.
jython/lib-python/2.7/string.py
zfill(x, width)
123$,8
00000123

19
Return a center version of s, in a field of the specified width. padded with spaces as needed. The string is never truncated. If specified the fillchar is used instead of spaces.
jython/lib-python/2.7/string.py
center(s, width, *args)
"test"$,10
  test

20
The wrap method takes in a string and wraps it such that it fits in lined of no more than the given width, these lines are returned as elements of a list
jython/lib-python/2.7/textwrap.py
wrap(text, width=70, **kwargs)
"hello world and all who inhabit it"$,8
['hello', 'world', 'and all', 'who', 'inhabit', 'it']

21
The unquote method shall decode an encoded parameter of a url, replacing all %20 with spaces
jython/lib-python/2.7/urlparse.py
unquote(s)
"abc def"
abc def

22
The wrap method takes in a string and wraps it such that it fits in lined of no more than the given width, these lines are returned as elements of a list
jython/lib-python/2.7/textwrap.py
wrap(text, width=70, **kwargs)
"hello world and all who inhabit it"$,15
['hello world and', 'all who inhabit', 'it']

23
Pad a numeric string x with zeros on the left, to fill a field of the specified width. The string x is never truncated.
jython/lib-python/2.7/string.py
zfill(x, width)
123$,0
123

24
Return the highest index in s where substring sub is found, such that sub is contained within s[start,end]. Optional arguments start and end are interpreted as in slice notation. Return -1 on failure.
jython/lib-python/2.7/string.py
rfind(s, *args)

"looking for the last instance of the letter g"$,"g"
44

25
The replace method takes in a string, and replaces all instances of a given substring with instances of a second given substring
jython/lib-python/2.7/string.py
replace(s, old, new)
"how will   it handle   a   t on of useless space   ?"$," "$,"_"
how_will___it_handle___a___t_on__of_useless_space____?