Methods and Tools in SW Development:

Homework 5

Github: https://github.com/joshuamoorexyz/Using-Pytest-Methods-and-Tools-22

I. Group Information

Group Number: 9
Group Member names/netIDs:
Joshua Moore : jjm702@msstate.edu
Rojal bishwokarma rb2298@msstate.edu
Sharad rann Sr2096@msstate.edu
Azim Bazarov ab4908@msstate.edu

II. Functions tested

1.Function Name:
openFile(filename)

Number of tests:
3

Items to test:
* Check that the function does in fact open a file if it exists
* Check that the function does not open a file when it does not exist. If the file does not exist, create it then open it thus passing the test
* Check that the program does not open a file when an invalid type (non string) is passed.Try to convert input to string then open file thus passing the test.

Inputs used for testing:
Path to file is saved in the FILE variable. FILE should exist on path, while FILE1 does not.

```
FILE="/home/{user}/Documents/Using-Pytest-Methods-and-Tools-22/tests
/testing.txt"
```

```
FILE1="/home/{user}/Documents/Using-Pytest-Methods-and-Tools-22/test
s/testing1.txt"
```

Corrections made:

The openFile function takes one parameter which is a string “filename”. This can be a string matching a file in the current directory, or a full path can be provided to the function. The function in its base state does in fact open a file if it exists when given a string matching a file. The function was changed however in the unit tests to create a file if given a string or path of a file that does not already exist. This means that the test can pass with a file being created. The function in functions.py was changed with try /except blocks to handle the possibility of the user inputting an invalid type. Here an integer input can be converted to string and then a file can be created. These changes help the function be more flexible to mistakes that the user can make.

```
## opens a file in read mode
## filename received as a parameter
def openFile(filename):
   try:
       string_int=str(filename)
       infile = open(string_int, "r")
       print("File opened.")
   except error:
       print("Please enter a string")
```

Tests Passed: 3
Tests Failed: 0

2.Function Name:
numbers(num1,num2)
Number of tests:
3

Items to test:
* Check that the function does in fact divide numbers properly
* Check that the function does not pass the test when comparing not equal return
* Check that the program does not open a file when an invalid type (non string) is passed

Inputs used for testing

Inputs are different for each test. Num1 = 3, num2 =1 in test one with expected output being 3. Same for test two but num2 is 0 to test divide by zero in python programming. The last test relies on invalid type being passed to the function.

Corrections made:

The numbers function takes two parameters which are integers used to divide then return the value. In its base state the function does not do any error correction so try/except block were added to the function to try and convert the input before returning the value. Otherwise, the function would fail due to bad input. Also the program is set to stop execution if num2 is 0.

```
def numbers(num1, num2):
   try:
       intfromstring=int(num1)
       intfromstring1=int(num2)
       if(intfromstring1==0):
           print("Please enter a non zero number for num2")
           sys.exit("Error message")
       else:
           return intfromstring / intfromstring1
   except error:
       print("Please enter an integer")
```

[redacted]

(optional) Failed tests:
test_numbers_dividebyzero() fails. This is supposed to happen for this function, to address the change was made to the numbers function to stop executing if 0 is inputted for num2.

Tests Passed: 2
Tests Failed: 1

3.Function Name:
dist()
Number of tests:
4

Items to test:
* Check that the function does give the distance between the values
* Check that the function does not pass the test when comparing not equal return distance
* Check that the function does equal when given equality to the first decimal point
* Check the function when given invalid type such as a string as input

Inputs used for testing
The input values used were 2 2 1 1 where (x1, y1, x2, y2). A string with the same input is passed to the last test to see if conversion happens correctly.

Corrections made:

The dist function takes 4 integers and returns the distance between the values. Corrections to the function include handling invalid type input from the user, and also using the ceil function to round up. I found that the decimal places made it very hard to track down when the comparison would be equal or not. By rounding up, this is not very hard.

```
def dist(x1, y1, x2, y2):
   try:
       intfromstringx1=int(x1) #changing style only bc its easier to keep
track of here :)
       intfromstringy1=int(y1)
       intfromstringx2=int(x2)
       intfromstringy2=int(y2)
       dist = (intfromstringx2 - intfromstringx1) ** 2 + (intfromstringy2
- intfromstringy1) ** 2
       dist = math.sqrt(dist)
```

```
        dist = math.ceil(dist)
        return dist
    except error:
        print("please enter integer values instead")
```

(optional) Failed tests:
Test_dist_noteq fails on purpose to make sure all tests are not passing automatically.

Test_dist_noteq fails on purpose due to a float being compared to an int. I want to make sure that using ceil, i do not have any float values being compared for testing purposes. In the real world, a different solution would most likely be the better one.

Tests Passed: 2
Tests Failed: 2

4.Function Name:
isPalindrome()
Number of tests:
3

Items to test:

* Check that the function does return true if a palindrome has been inputted
* Check that the function does not pass the test when a non palindrome input has occurred.
* Check the function when given invalid type such as a int as input

Inputs used for testing
*Input 1,
Input 2

Corrections made:

```
def isPalindrome(temp):
   try:
       stringfromint=str(temp)
       test = stringfromint[::-1]

       if(test == stringfromint):
           return True

       else:
           return False
   except error:
       print("please enter integer values instead")
```

(optional) Failed tests:
(N/A)

Tests Passed: 3
Tests Failed: 0

5.Function Name:
divide()
Number of tests:
3

Items to test:
* Check that the function does pass if intended result is returned
* Check that the function does not pass the test when division by zero occurs
* Check the function when given invalid type such as a string as input

Inputs used for testing
*Input 1 integer value
Input 2 integer value which is not 0

Corrections made:
For the divide function the base function did not change much other than returning div. I did change the test functions however which i have included. Using monkeypatch in the test functions i was able to “mock” input “GEN” and pass in values as if i were the user. This allowed me to fully automate the testing process and perform small,concise unit tests for this function. The last part I added was checking to see if num2 was zero and halting execution if so.

```
def divide():
   num1 = int(input("Enter a number: "))
   num2 = int(input("Enter another number: "))
   if(num2==0):
       print("Please enter a non zero number for num2")
       sys.exit("Error message")
   div = num1/num2
   print("Your numbers divided is:", div)
   return div
```

```
def test_divide(monkeypatch):
       monkeypatch.setattr('builtins.input',lambda _: next(GEN))
       assert divide() == 3.0

#will fail due to division by zero
#will pass if passing in GEN instead
def test_dividebyzero(monkeypatch):
   monkeypatch.setattr('builtins.input',lambda _: next(GEN1))
   assert divide() == 0

#fails due to int being passed in through monkeypatch which expects a
string
#will pass if correct type in list
def test_divide_invalidtype(monkeypatch):
       monkeypatch.setattr('builtins.input',lambda _: next(GEN2))
       assert divide() == 3.0
```

(optional) Failed tests:
Test_divide_dividebyzero fails on purpose due to its intent

Tests Passed: 2
Tests Failed: 1

Seeing as how this problem also mimics user input i thought to detail generate inputs for each test as well here.

```python
def geninputs():
    inputs = ["3","1"]

    for item in inputs:
        yield item

def geninputs1():
    inputs = ["3","0"]

    for item in inputs:
        yield item

def geninputs2():
    inputs = ['3','1']

    for item in inputs:
        yield item


#sets up our inputs
GEN = geninputs()
GEN1=geninputs1()
GEN2=geninputs2()
```

By using ‘ instead of “ “ for the last set of inputs, i was able to mimic a string being inputted by the user and test my invalidtype test!

6.Function Name:
sq
Number of tests:
3

Items to test:
* Check that the function does pass if intended result is returned
* Check that the function does not pass the test when division by zero occurs
* Check the function when given invalid type such as a string as input

Inputs used for testing
49 as an int and 49 as a string

Corrections made:
For this program i only added the intermediary step of casting int type to make the 3rd test “invalidtype_test” pass.

```
def sq(num):
   number=int(num)
   return math.sqrt(number)
```

(optional) Failed tests:
Test_sq_false fails due to incorrect assert. If it was changed to 7 it would pass.

Tests Passed: 2
Tests Failed: 1

7.Function Name:
greetUser(first,middle,last)
Number of tests:
3

Items to test:
* Check that the function does pass if intended output occurs
* Check that the function does not pass when a typo has occurred. String to string comparison so they must match perfectly
* Check the function when given invalid type such as a string as input

Inputs used for testing
My {first, middle, last} names were used as input into the function.

Corrections made:
This function does not have to have try/except blocks or cast a type due to the string comparison. The last test passes in ints instead of names, and the program just interprets it the same. If conversion happened, the program would not be able to effectively test string to string and another testing solution would have to be done.

```
def greetUser(first, middle, last):
    print("Hello!")
    print("Welcome to the program", first, middle, last)
    print("Glad to have you!")
```

(optional) Failed tests:
test_greetUser_typo fails on purpose as the string != captured output. This is to catch mistakes made by the programmer implementing the test. Really not needed too much.

Tests Passed: 2
Tests Failed: 1

8.Function Name:
displayItem(numbers, index)
Number of tests:
3

Items to test:
* Check that the function does pass if intended output occurs
* Check that the function does not pass when item at index is not equal
* Check the function when given index is outside of defined area

Inputs used for testing

2 Strings
1. “Your item at 2 index is stool”
2. “Your item at 2 index is chair”

Corrections made:

The correction made to this function was checking to make sure that the index number is not outside of the bounds.

```python
def displayItem(numbers, index):
   if(index < 0 or index > len(numbers)): #check to see if index size is greater than zero but not out of bounds
       sys.exit("Error message")
   else:
       print("Your item at", index, "index is", numbers[index])
```

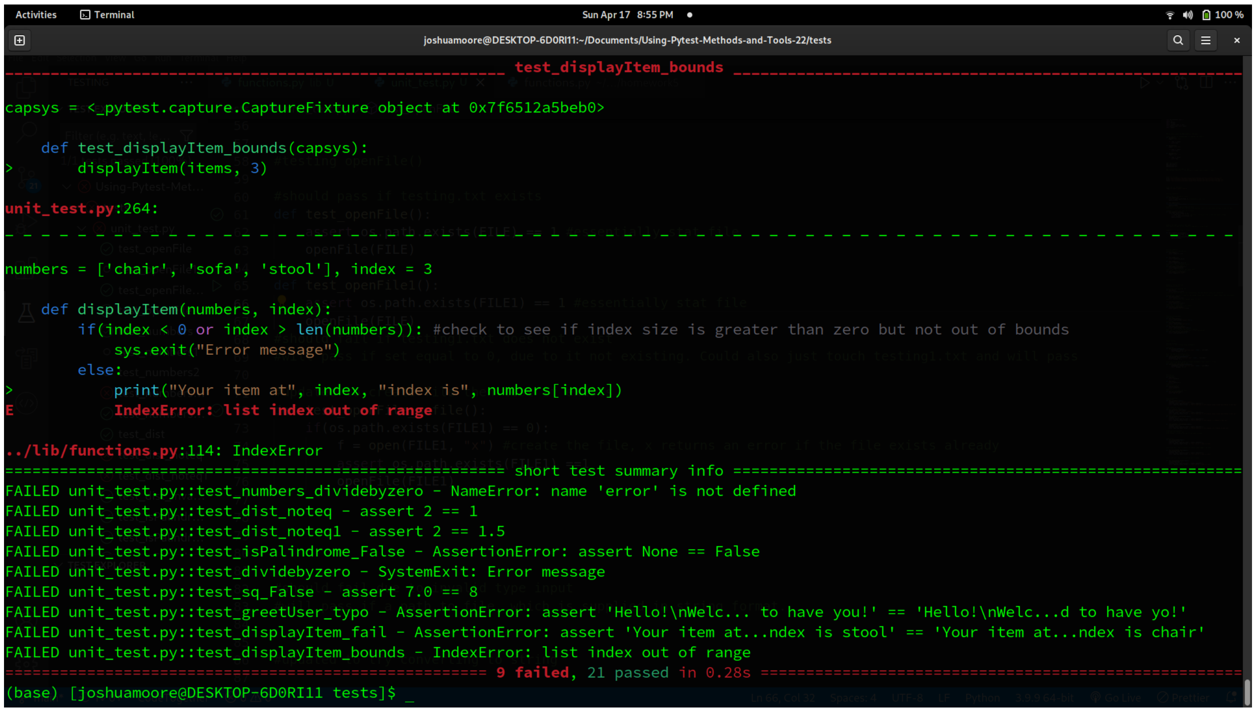
(optional) Failed tests:

test_displayItem_fail fails due a failed string comparison. The item at the index location is not a chair in this case.

test_displayItem_bounds Fails due to reading outside the bounds of the index by giving an index number of 3

Tests Passed: 1
Tests Failed: 2

III. Pytest Output



Activities
Terminal
Sun Apr 17 8:55 PM
100 %
joshuamoore@DESKTOP-6D0RI11:~/Documents/Using-Pytest-Methods-and-Tools-22/tests

```
____________________________________________________________ test_displayItem_bounds _____________________________________________________________

capsys = <_pytest.capture.CaptureFixture object at 0x7f6512a5beb0>

    def test_displayItem_bounds(capsys):
>       displayItem(items, 3)

unit_test.py:264:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

numbers = ['chair', 'sofa', 'stool'], index = 3

    def displayItem(numbers, index):
        if(index < 0 or index > len(numbers)): #check to see if index size is greater than zero but not out of bounds
            sys.exit("Error message")
        else:
>           print("Your item at", index, "index is", numbers[index])
E           IndexError: list index out of range

../lib/functions.py:114: IndexError
======================================================= short test summary info ========================================================
FAILED unit_test.py::test_numbers_dividebyzero - NameError: name 'error' is not defined
FAILED unit_test.py::test_dist_noteq - assert 2 == 1
FAILED unit_test.py::test_dist_noteq1 - assert 2 == 1.5
FAILED unit_test.py::test_isPalindrome_False - AssertionError: assert None == False
FAILED unit_test.py::test_dividebyzero - SystemExit: Error message
FAILED unit_test.py::test_sq_False - assert 7.0 == 8
FAILED unit_test.py::test_greetUser_typo - AssertionError: assert 'Hello!\nWelc... to have you!' == 'Hello!\nWelc...d to have yo!'
FAILED unit_test.py::test_displayItem_fail - AssertionError: assert 'Your item at...ndex is stool' == 'Your item at...ndex is chair'
FAILED unit_test.py::test_displayItem_bounds - IndexError: list index out of range
===================================================== 9 failed, 21 passed in 0.28s =====================================================
(base) [joshuamoore@DESKTOP-6D0RI11 tests]$ _
```