Worksheet 8: White-Box Testing and Test Fixtures

Updated: 16th February, 2022

There are two versions of this worksheet. This is the Python version.

In this worksheet, you'll practice white-box test design, and the setting-up and tearing down of test fixtures. There are a range of production code functions for you to implement test code for. The code is shown below, and is also available in `Utils.py`.

For each production code function, do the following:

(a) Design your test cases:

- Identify the paths through the production code.
- Select test data for each test case. In other words, for each path, select inputs (parameters, console input, and/or input files) that will cause the production code to follow that path.
- For each test case, determine the expected results. This includes return values, exceptions thrown, console output, and output files.

(b) Implement your test cases.

It's good experience to continue to use the `unittest` module, although you can still perform the exercise without it.

(c) Run your tests against the production code.

To ensure that your test code is working, it's helpful to temporarily *break* the production code. You could do this by editing the production code to alter the output/results very slightly.

1. printCoordinates()

`printCoordinates()` takes in *x*, *y* and *z* coordinates, and prints them out in the format (*x*, *y*, *z*), with two decimal places each. The output will end in a new line.

(This is a trivial case for test design. Since there are no conditional statements, there is only one path, and hence one test case.)

```
def printCoordinates(x, y, z):
    print("({:.2f},{:.2f},{:.2f})".format(x, y, z))
```

Note: Remember to "tear down" everything that you set up. Specifically, to restore console output in Python:

```
import sys
...
sys.stdout = ...                # Redirect output
...                             # Do testing
sys.stdout = sys.__stdout__# Restore output
```

To delete temporary files:

```
import os
...
os.remove("somefile.txt")
```

Warning: Be very careful with code that deletes files! Ensure the file you specify is definitely the one you want to delete.

2. readChar()

`readChar()` reads a single "valid" character from the user. The user enters a character, but must re-enter their input if it's invalid; i.e., if the character they enter does not occur within the `validChars` parameter.

Hint: there are two paths, and hence two test cases here.

```
def readChar(validChars):
    line = input()
    while len(line) != 1 or validChars.find(line) == -1:
        line = input()
    return line[0]
```

Note: Tearing-down console input redirection in Python is much the same as for output:

```
import sys
...
sys.stdout = ...                # Redirect input
...                             # Do testing
sys.stdout = sys.__stdout__# Restore input
```

3. guessingGame()

`guessingGame()` runs a console-based number guessing game. The user is repeatedly asked to guess what the number is, and is told whether their guess is too high, too low, or correct (at which point the game ends).

Hint: the *result* of `guessingGame()` consists of all the console output, including the prompt asking for input. Also note that `println()` will print a newline character, while `print()` won't. This will be important when determining the expected output.

```
def guessingGame(number):
    guess = int(input("Enter an integer: "))

    while guess != number:
        if guess > number:
            print("Too high.")

        else:
            print("Too low.")

        guess = int(input("Enter an integer: "))

    print("Correct!")
```

4. sumFile()

`sumFile()`: opens a file (assumed to contain a list of numbers), and adds up the numbers, and returns the total. If the file could not be opened, it returns -1 instead.

Hint: the empty string `""` is an invalid filename that, by definition, cannot be opened.

```
def sumFile(filename):
    sum = 0.0
    try:
        with open(filename) as inFile:          # Raises OSError
            content = inFile.read().split()
            for word in content:
                sum += float(word)

    except OSError:
        sum = -1.0

    return sum
```

Note: Python's "`with`" statements have no effect on paths. (They're used to arrange for automatic clean-up of some sort of resource, such as the closing of a file after you're finished with it.)

5. saveData()

`saveData()` opens a given file, and writes several lines of text to it. Returns `true` if the file could be written, and `false` if not (e.g., because an invalid name was given).

```
def saveData(filename, name, health, score):
```

```
    success = True
    try:
        with open(filename, mode = "w") as writer: # Raises OSError
            writer.write("name: " + name + "\n")
            writer.write("health: " + str(health) + "\n")
            writer.write("score: " + str(score) + "\n")
            if health <= 0.0:
                writer.write("status: dead\n")
            else:
                if score >= 100:
                    writer.write("status: won\n")
                else:
                    writer.write("status: alive\n")

    except OSError:
        success = False

    return success
```

Hint: to verify that the file was written correctly:

```
...
with open("outfile.txt") as outFile:
    self.assertEqual("name: myname\n", outFile.readline())
    ... // More assertions
```

End of Worksheet