Chapter Four:
Completing the Testing Framework

Changes Implemented from Previous Chapter

There were three issues with our partial testing framework:

- Unnecessary number of drivers
- The script knowing the test case names
- Formatting the test result output

The first issue was resolved incredibly easy, by simply making a generalized driver for each method. Thus, we have 5 methods, and their respective drivers.

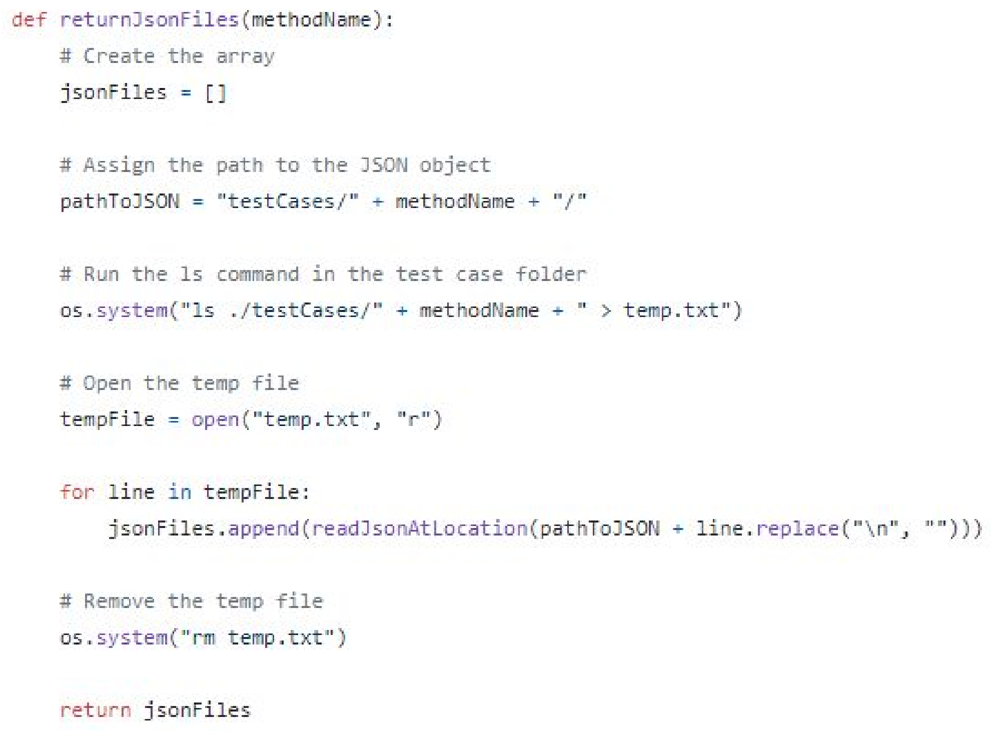
The script knowing the test case names was resolved by relying on the static nature of the file structure associated with our project. While the script shouldn't know anything about the actual test cases, it can find the location of the test cases, and iterate through the files present in the testCases directory, which are the test cases, and temporarily store them in an array methodNames.

returnJsonFiles takes a methodName, and gets the path to the testCase for that specific method name, based on the fact that all the testCases are located in testCases/*method-name*/.

It then runs ls on the ./testCases/ directory and stores the results in a temporary text file called temp.txt. It then will iterate through the temporary file and read the Json files by line. Finally it will remove the temporary file and return an array of the jsonFiles.

Thus, all other functions included in the script rely on reading in methodName for testing, which is now retrieved by way of traversing the file structure and looking for specific files rather than those files being hardcoded, which defeats the purpose of the automation.

How the Script Gets the Method Names

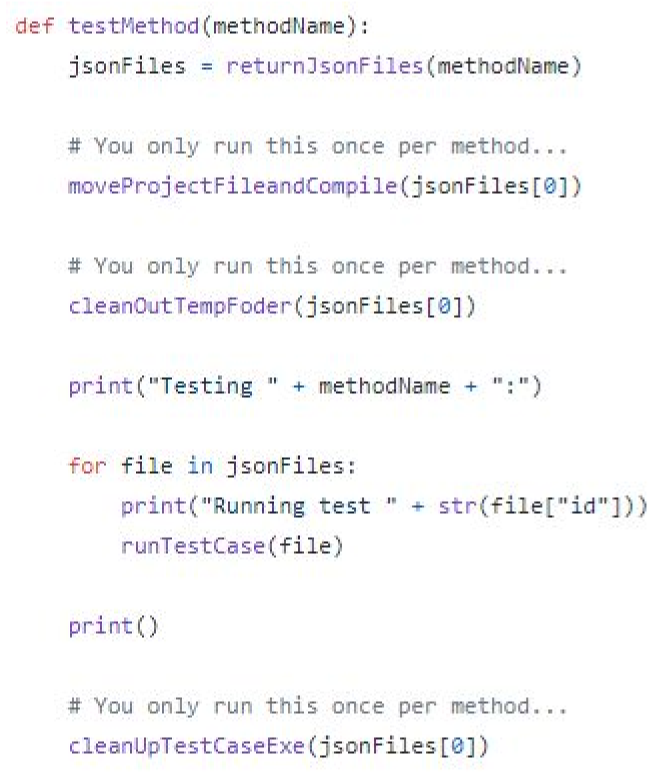


```
def returnJsonFiles(methodName):
    # Create the array
    jsonFiles = []

    # Assign the path to the JSON object
    pathToJSON = "testCases/" + methodName + "/"

    # Run the ls command in the test case folder
    os.system("ls ./testCases/" + methodName + " > temp.txt")

    # Open the temp file
    tempFile = open("temp.txt", "r")

    for line in tempFile:
        jsonFiles.append(readJsonAtLocation(pathToJSON + line.replace("\n", "")))

    # Remove the temp file
    os.system("rm temp.txt")

    return jsonFiles
```

Example Driver for Method calculateSlope

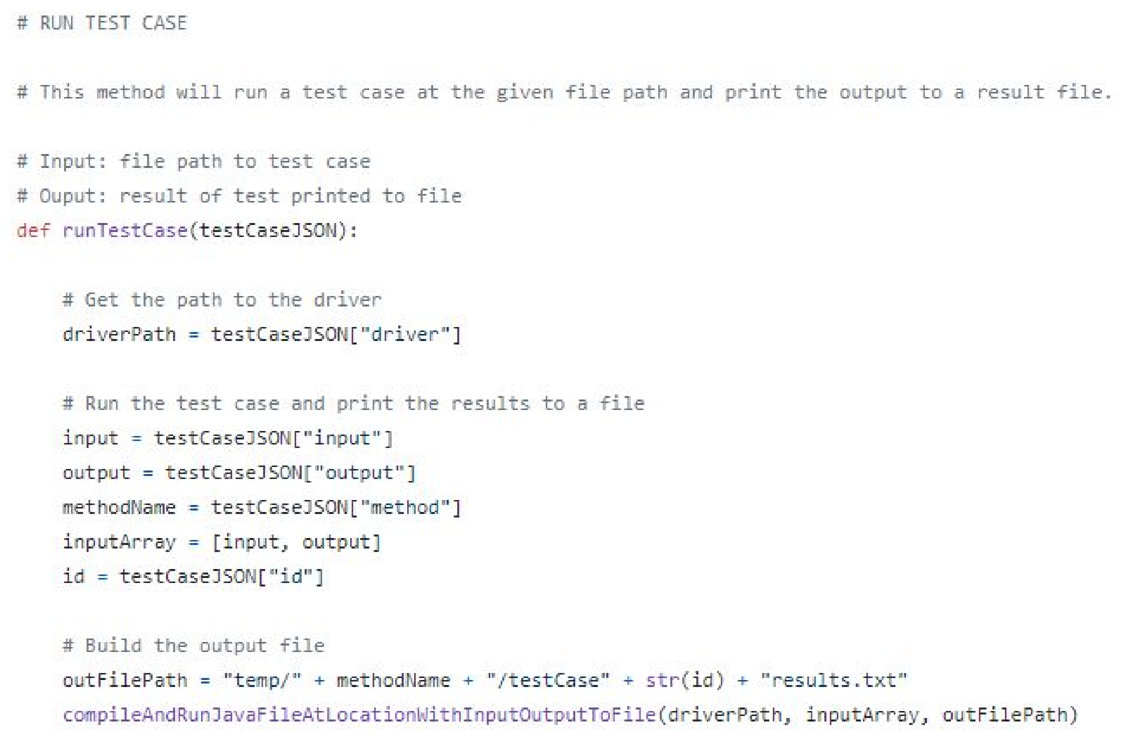
```java
public class testCase {
    public static void main(String[] args) {

        // Assign values from input
        Double x1 = Double.parseDouble(args[0]);
        Double y1 = Double.parseDouble(args[1]);
        Double x2 = Double.parseDouble(args[2]);
        Double y2 = Double.parseDouble(args[3]);

        // Print out test result
        double testOracle = Double.parseDouble(args[4]);

        List<Double> Xlist = new ArrayList<Double>();
        Xlist.add(new Double(x1));
        Xlist.add(new Double(x2));

        List<Double> Ylist = new ArrayList<Double>();
        Ylist.add(new Double(y1));
        Ylist.add(new Double(y2));

        // Instantiate the Binomial Distribution Utility class
        LinearLeastSquaresFit LinearLeastSquaresFitOBJ = new LinearLeastSquaresFit(Xlist, Ylist);

        double slope = LinearLeastSquaresFitOBJ.calculateSlope(Xlist, Ylist);

        // Print test number
        System.out.println("Test:");
        System.out.printf("Calculate slope bettween (%f, %f) and (%f, %f)\n", x1, y1, x2, y2);

        System.out.println("Result: " + slope);

        // Test passed
        if (slope == testOracle) {
            System.out.println("Oracle: " + testOracle);
            System.out.println("Pass");
        }
        // Test failed
        else if (slope != testOracle) {
            System.out.println("Oracle: " + testOracle);
            System.out.println("Fail");
        }
        // Test ERROR
        else {
            System.out.println("ERROR");
        }
    }
}
```

Test Method



```
def testMethod(methodName):
    jsonFiles = returnJsonFiles(methodName)

    # You only run this once per method...
    moveProjectFileandCompile(jsonFiles[0])

    # You only run this once per method...
    cleanOutTempFoder(jsonFiles[0])

    print("Testing " + methodName + ":")

    for file in jsonFiles:
        print("Running test " + str(file["id"]))
        runTestCase(file)

    print()

    # You only run this once per method...
    cleanUpTestCaseExe(jsonFiles[0])
```

Executing a Test Case

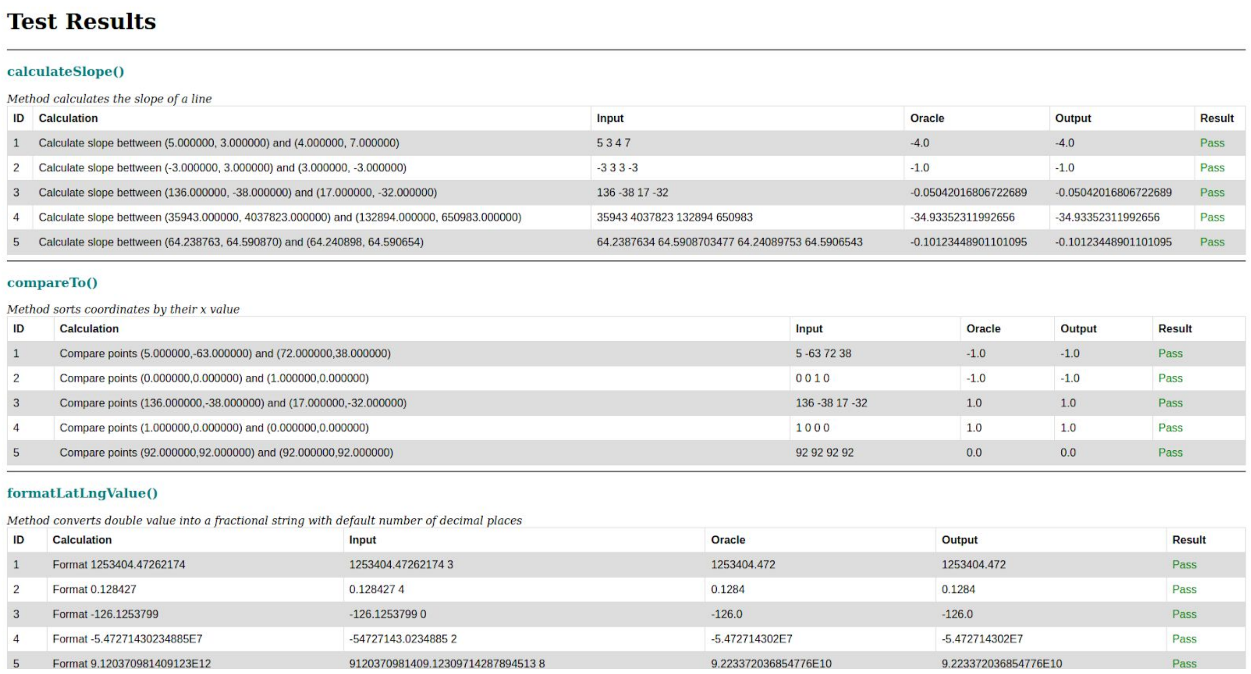


```
# RUN TEST CASE

# This method will run a test case at the given file path and print the output to a result file.

# Input: file path to test case
# Ouput: result of test printed to file
def runTestCase(testCaseJSON):

    # Get the path to the driver
    driverPath = testCaseJSON["driver"]

    # Run the test case and print the results to a file
    input = testCaseJSON["input"]
    output = testCaseJSON["output"]
    methodName = testCaseJSON["method"]
    inputArray = [input, output]
    id = testCaseJSON["id"]

    # Build the output file
    outFilePath = "temp/" + methodName + "/testCase" + str(id) + "results.txt"
    compileAndRunJavaFileAtLocationWithInputOutputToFile(driverPath, inputArray, outFilePath)
```

Formatting Test Results

The last issue was the quality of the test result output, which was vastly improved between last deliverable and now. Addition of color coding specific to test cases pass/fail, cleaning up the tables, and generally making it more attractive was simple html.

HTML Output



Test Results

calculateSlope()

Method calculates the slope of a line

ID	Calculation	Input	Oracle	Output	Result
1	Calculate slope bettween (5.000000, 3.000000) and (4.000000, 7.000000)	5 3 4 7	-4.0	-4.0	Pass
2	Calculate slope bettween (-3.000000, 3.000000) and (3.000000, -3.000000)	-3 3 3 -3	-1.0	-1.0	Pass
3	Calculate slope bettween (136.000000, -38.000000) and (17.000000, -32.000000)	136 -38 17 -32	-0.05042016806722689	-0.05042016806722689	Pass
4	Calculate slope bettween (35943.000000, 4037823.000000) and (132894.000000, 650983.000000)	35943 4037823 132894 650983	-34.93352311992656	-34.93352311992656	Pass
5	Calculate slope bettween (64.238763, 64.590870) and (64.240898, 64.590654)	64.2387634 64.5908703477 64.24089753 64.5906543	-0.10123448901101095	-0.10123448901101095	Pass

compareTo()

Method sorts coordinates by their x value

ID	Calculation	Input	Oracle	Output	Result
1	Compare points (5.000000,-63.000000) and (72.000000,38.000000)	5 -63 72 38	-1.0	-1.0	Pass
2	Compare points (0.000000,0.000000) and (1.000000,0.000000)	0 0 1 0	-1.0	-1.0	Pass
3	Compare points (136.000000,-38.000000) and (17.000000,-32.000000)	136 -38 17 -32	1.0	1.0	Pass
4	Compare points (1.000000,0.000000) and (0.000000,0.000000)	1 0 0 0	1.0	1.0	Pass
5	Compare points (92.000000,92.000000) and (92.000000,92.000000)	92 92 92 92	0.0	0.0	Pass

formatLatLngValue()

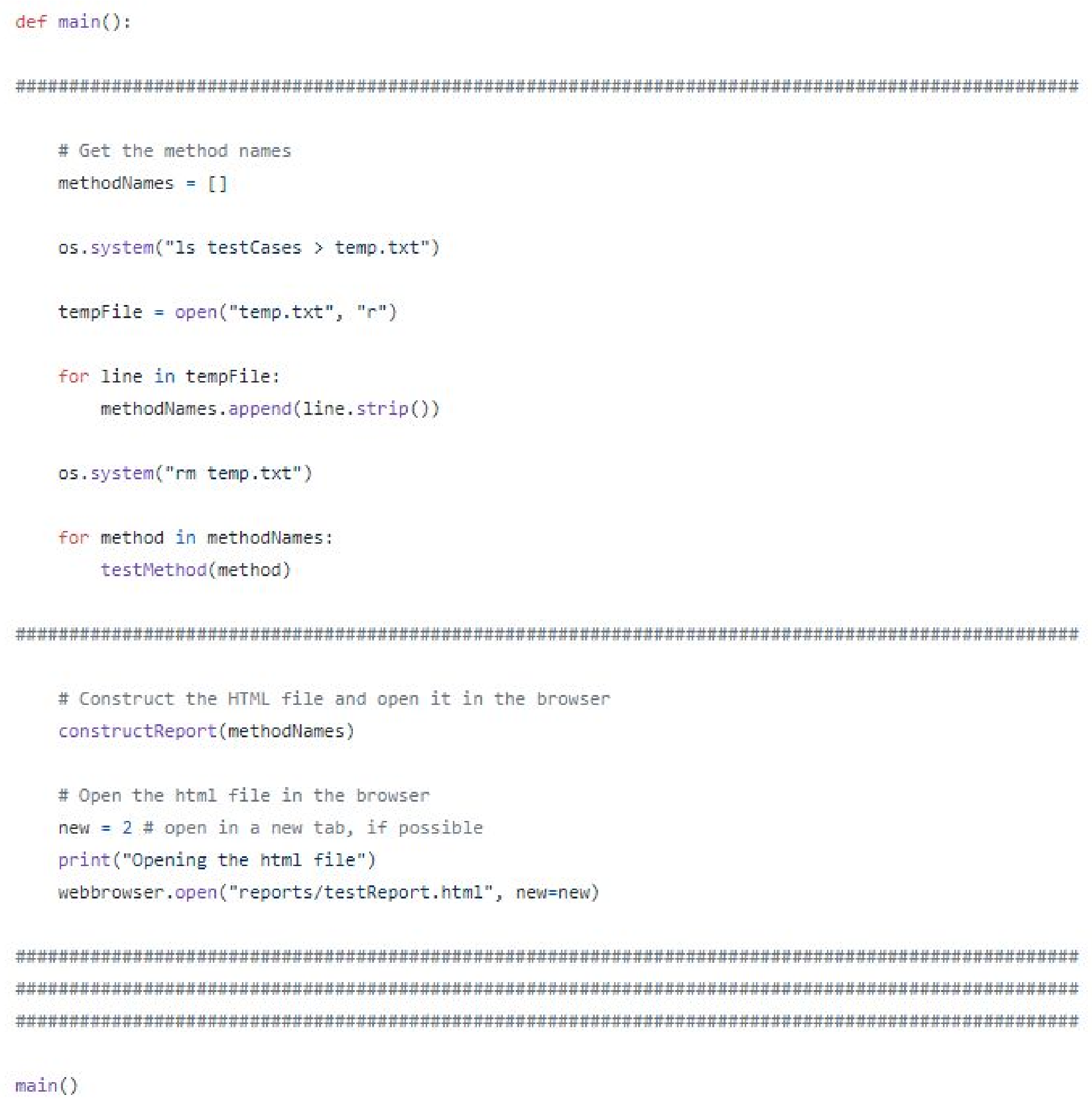
Method converts double value into a fractional string with default number of decimal places

ID	Calculation	Input	Oracle	Output	Result
1	Format 1253404.47262174	1253404.47262174 3	1253404.472	1253404.472	Pass
2	Format 0.128427	0.128427 4	0.1284	0.1284	Pass
3	Format -126.1253799	-126.1253799 0	-126.0	-126.0	Pass
4	Format -5.47271430234885E7	-54727143.0234885 2	-5.472714302E7	-5.472714302E7	Pass
5	Format 9.120370981409123E12	9120370981409.12309714287894513 8	9.223372036854776E10	9.223372036854776E10	Pass

Extending the Testing Framework to the Rest of the Methods

After resolving the issues above, extending the framework to include the four additional methods was simple, because each method is treated exactly the same with its respective driver being called in the script.

The majority of the time spent between chapter three, and the current chapter was working on generalizing the code, because it was too specific (having more than one driver, or the script having the method names hard coded), as well as optimizing the output file in terms of appearance and data presentation.



```
def main():

####################################################################################################

    # Get the method names
    methodNames = []

    os.system("ls testCases > temp.txt")

    tempFile = open("temp.txt", "r")

    for line in tempFile:
        methodNames.append(line.strip())

    os.system("rm temp.txt")

    for method in methodNames:
        testMethod(method)

####################################################################################################

    # Construct the HTML file and open it in the browser
    constructReport(methodNames)

    # Open the html file in the browser
    new = 2 # open in a new tab, if possible
    print("Opening the html file")
    webbrowser.open("reports/testReport.html", new=new)

####################################################################################################
####################################################################################################
####################################################################################################

main()
```

Executing all 25 test cases for the 5 methods is then done by iterating through the methodNames that have been pulled from the .json files, and executing each respective driver (5 total), gathering the data from these test cases, and then formatting and outputting the results via testReport.html.