CamelCased Testing Framework

Some changes were made to the test file structure. The test file is now structure as follows:

1. test #
2. module being tested
3. the class being tested with inputs
4. the method being tested with inputs
5. the expected result
6. a short sentence describing the test

As an example:

```
4
django.core.paginator
Paginator([1,3,4],2)
validate_number(0)
error
should throw error when a number less than 1 is passed
```

Adding A Test

Adding a test is as easy as creating a new test file and conforming to the test file structure specified above. Make sure spelling of module, class, and method names is correct. Also ensure parameter input is correct. Valid parameters are: strings, integers, lists, tuples, dictionaries, boolean, and floating point. An invalid parameter input will cause that parameter to be evaluated as a string.

The same as above goes for the expected outcome. If an error is expected, the expected result (line 5) should read: 'error' or 'err'. This is not case sensitive.

Running The Tests

Tests can be run by running *./scrips/runAllTests.* Running all tests executes the *./testCaseExecutables/testCaseDriver.py* for each test text file in the testCases folder. Currently the only valid directory to run tests from is the root directory of the framework: *./CamelCased_TESTING*.

A web browser will be opened displaying the results from each test. The output will look something like this:

```
4 should throw error when a number less than 1 is passed success
```

Running Individual Tests

There is currently no way to run an individual test using the test text files. If an individual test needs to be run, execute ./testCaseExecuteables/testCaseDriver.py with 4 string arguments. The arguments would be line 2-5 of the corresponding text file. Example:

```
./testCaseExecuteables/testCaseDriver.py “django.core.paginator” “Paginator([1,3,4],2)” “validate_number(0)” “error”
```

The response will either be a “success” “failure” or “skipped” in the terminal.
Adding an easier way to run individual tests will be added in the future.

Known Issues

The main issue is that a universal driver, while allowing for simple test case creation, greatly reduces the flexibility of the framework. While we may never face this issue in our project, there are cases where the current executable will not work. For instance, connecting to and testing an interface to a database or a local server.

As a solution, a driver could be created for each module or class to add more flexibility.