## Automated Testing Framework: Adding New Logic for New Tests

Some changes were made in how our testing framework works since Chapter 3. Now, instead of using the names/numbers of the tests to run separate files in the testCaseExecutables directory, we run all tests using information in the .txt files found in the testCases directory.

testCase10.txt

2 ms

testCase1.txt

docs

Safari/537.36

1. 10 (1) Rorun

2. 11 (1) Roun

Tests completed in 26 milliseconds. 8 assertions of 9 passed, 1 failed.

Running Test Case 1...

```
testCase2.txt
opt
                                                        test_number: 10
                                                        requirement_being_tested: Reverse a string
                               testCase3.txt
project
                                                        component_being_tested: String reversal function in EnDe.is
                               testCase4.txt
reports
                                                        method_being_tested: reverse(src)
                                                        test_input: src = "Testing"
                               testCase5.txt
scripts
                                                        expected_outcome: "gnitseT"
temp
                               testCase6.txt
                                                        executable: whatever is
                             testCase7.txt
testCaseExecutables
                                                        method_invocation: EnDe.reverse(%s)
                             testCase8.txt
                                                        arity: 1
 testCases
                             testCase9.txt
testCasesBroken
                            testCase10.txt
                               testCase11.txt
                             testCase12.txt
                             testCase13.txt
```

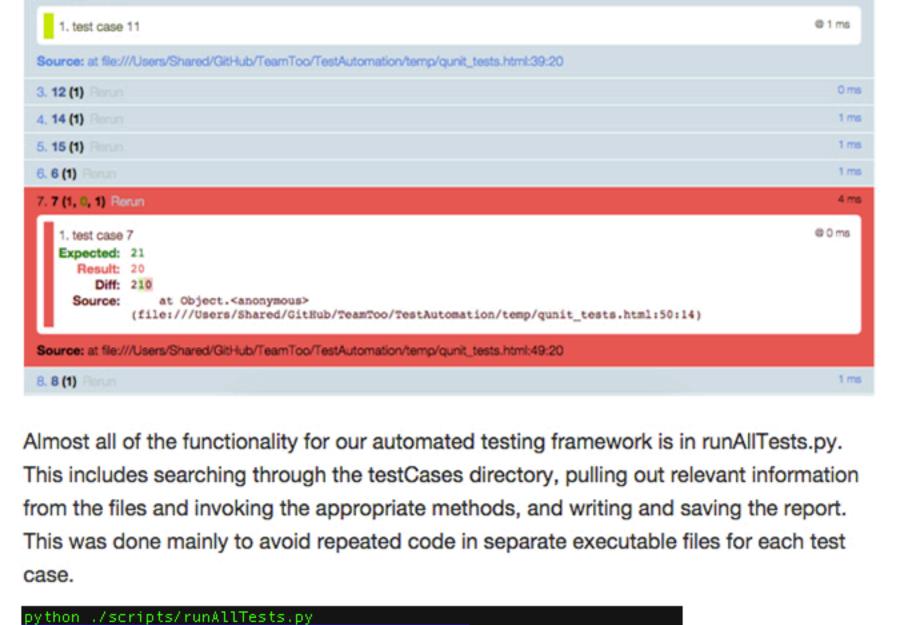
tests written in Python that use Selenium to simulate interactions in a browser window to test the user interface, and tests on the JavaScript code that EnDe is largely written in. In runAllTests.py, we check whether we need to be running Python code to test the user interface or JavaScript code to test methods directly. For the JavaScript tests we used

These files contain all information needed to execute the tests without needing repeated

code in the testCaseExecutables files. We use two different types of cases in our testing:

whether a test passed or failed using QUnit, then used those values in our report. **QUnit Example** ☐ Hide passed tests ☐ Check for Globals ☐ No try-catch Filter: Go QUnit 1.20.0; Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_10\_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.86

QUnit, which is a JavaScript unit testing framework. We were able to get the results of



test input: Euro expected outcome: RXVybw== EnDe result: RXVybw== Equal? True Running Test Case 13... test input: \$Monies\$ expected outcome: 1062582A77640053 EnDe result: 1062582A77640053 Equal? True Running Test Case 2...

```
test input: Hex Test
        expected outcome: 4865782054657374
        EnDe result: 48,65,78,20,54,65,73,74
        Equal? False
Running Test Case 3...
         test input: 42
         expected outcome: 101010
        EnDe result: 101010
        Equal? True
Running Test Case 4...
        test input: Testing
        expected outcome: Grfgvat
        EnDe result: Grfgvat
        Equal? True
Running Test Case 5...
         test input: SOS
        expected outcome: ...
                                                All infos
        EnDe result: ... ___ ..
                                                used are
        Equal? False
                                                from each
Compling JavaScript tests...
                                                testCase#.txt
Test 10...
Test 11...
Test 12...
Test 14...
                         Javascript Tests
Test 15...
Test 6...
Test 7...
Test 8...
Test 9.
Compiled...running
The Test Case Text Files
Our test case text files contain different information depending on whether the
information will be used in a Python user interface test or a JavaScript test for functions.
All test case files contain a test number, requirement being tested, component being
tested, test input, and expected outcome (oracle). For the user interface tests, we also
include a list number which allows us to locate which item in the list of types of
```

## testCase2.txt Rest\_number: 8 requirement\_being\_tested: Convert string to list of characters with prefix, delimiter and suffix

**Errors and Manual Checks** 

up to the amount required.

test\_number: 2
requirement\_being\_tested: Convert an input string into hexidecimal display the resulting hexidecimal value as a string.
component\_being\_tested: the EnDe graphical interface - index.html
method\_being\_tested: NA
test\_input: Mex Test
expected\_outcome: 4865782954657374
language: python
list number: 6 suffix
component\_being\_tested: Character encoding function in EnDe.is
method\_being\_tested: Character encoding function in EnDe.is
method\_being\_tested: chr(\_n1\_,\_n2\_\_\_n3\_,src,prefix,suffix,delimiter)
test\_input: \_n1 = null, \_n2 = null, \_n3 = null, src = "Testing", prefix = "~", suffix =
"!", delimiter = ".."
expected\_outcome: "~T!.~e!.~s!.~t!.~i!.~n!.~g!"
language: javascript
method\_invocation: EnDe.EN.chr(%s, %s, %s, %s, %s, %s, %s)
Arixx! 7 list\_number: 6 link\_id: EnDeDOM.EN.Actions.s.hex

We did find some issues with using the test case files, however. One issue arose when

characters. Of course, this means that the test did fail, but it would also cause the scripts

to break, and it would cause issues with the files being written (ie: the results.html). Since

this was causing the automated tests to fail, and scripts to be broken, this test had to be

testCase10\_broken.txt - Edited

implementing automated testing for values that were returned not as UTF-8 encoded

encryption we need to click in the user interface and a link id that allows us to click the

JavaScript tests, we include the method being tested and how to invoke the method. We

also include arity, so if a user enters too many arguments it will only accept the first ones

testCase8.txt

proper specific encryption/decryption method in a sub-menu under that type. For our

## checked manually to verify the issue. The culprit test case:

requirement\_being\_tested: AES-128 encoding - verifiying that output matches expected from web browser component\_being\_tested: the EnDe graphical interface - index.html method\_being\_tested: AES-128 encoding test\_input: heureca! expected\_outcome:  $\xa3\x98\x17\xc9\x26\x01\x00\x00\x2d\x7c\x4d\x3b\xfe\x1d\xc2\x01\x07$ executable: testCase9.py list\_number: 9 link\_id: EnDeDOM.EN.Actions.s.aes128 SERIOUSLY. DON'T USE IT. IT WILL BREAK EVERYTHING. The error being thrown:

expected outcome: \xa3\x98\x17\xc9\x26\x01\x00\x00\x2d\x7c\x4d\x3b\xfe\x1d\xc2\x01\x07

UnicodeEncodeError: 'ascii' codec can't encode characters in position 302-304: ordinal not in range(120)

Notice how the output recorded in the console only shows 'some' non-standard

jkhviuyv3rcsdf832099874%!\$#5\* asldfkjasdhfibv== lk;'op,huoy8,,

characters. Other characters aren't shown because the console can't display them.

Encoding URI/URL

Decoding URI/URL

Unicode/UTF Base-N Coding Straight Numbers Characters Decryption

HTML-Entity (NCR)

JavaScript built-in Escape Characters

Fuzzy decoding

Special

HTML-Entity (NCR)

Running Test Case 10...

test input: heureca!

Sample manual test from the GUI:

Equal? False Traceback (most recent call last):

EnDe result: #懗P-X, <Ôôû

testCaseData['expected\_outcome'], val))

File "./scripts/runAllTests.py", line 129, in <module>

test\_number: DON'T USE THIS ONE

```
Unicode/UTF
Base-N
Coding
Straight
Numbers
Characters
Encryption
Hash/Checksum
JavaScript built-in
Escape Characters
Special
Symbols
                      Text Hex parsed
                                                                                           Window ..
                                                                                                       Scratchpad ..
Beautify
Repeat
```

1□, □P□□□-RCT□ ++s□ø}Û\*æ□□-p&±□mhpS□»áæ\□Ü□-\$□āYM□«Êâì3ós.1¶-□;}k>□□?θu,^ēô?-¨

Beautify Scratchpad .. Text Hex parsed Window .. NOTYET Repeat Notice in this one the non-standard characters that are shown as boxes. These are what is attempting to be added to the results.html file. The only reason they show up here as

like these. In order to prevent the error from popping up while testing, and causing issues in doing so, entering any value other than an numerical one will cause the testing to stop before

boxes is because that is what Firefox has implemented to catch non standard characters

```
the values are sent to test.
Traceback (most recent call last):
  File "./scripts/runAllTests.py", line 73, in <module>
```

ValueError: invalid literal for int() with base 10: "DON'T FREAKING USE THIS ONE"

testCaseIndex = int(testCaseData['test\_number'])