COSC 310
ASSIGNMENT 3

GROUP 24

Robot Robert
Unit Test Documentation

ABSTRACT

Ultimately test cases are there to make sure that everything still works after each major upgrade. We don't want to change a bunch of code only to find out that half the code no longer works because of our changes. Because of that, the focus of our unit tests are on making sure each component responds exactly as we would expect. Failures in these cases are most likely to be caused by failed imports in python, so it is a good idea to run these tests so that someone trying to run the code can figure out what imports have failed, and then test to make sure they're working as intended afterwards.

To call the tests, simply run the unitTest.py file located in the repository.

We've used Python's built in unittest function, which is modeled to work like Java's JUnit tests. More information can be found at https://docs.python.org/3/library/unittest.html

UNIT TESTS

testResponse:

This is the first major test, this checks to make sure that when given a proper opcode and value, the response.py file's formulateResponse() function correctly returns the string we expect. This is important, because we want to make sure that the system can find the correct response to a given decoded input. This also tests to make sure the response.py file can be imported, which is critical to the program.

testName:

Another major test, we want to make sure that if we send a valid opcode and value input to the phraseInput.py file's interpolate() function (poorly named, I'm aware) that it will correctly call the response.py file's formulateResponse() and return the value we expect. This tests to make sure that both the phraseInput.py file can be correctly imported (which is important for the interface) as well as the phraseInput.py file correctly importing the response.py file.

testStem:

A pretty minor test to make sure that NLTK's Porter Stemmer is working properly. Since we use this to account for minor spelling errors in the input, we want to be absolutely sure that what is returned is correctly stemmed, or at least is consistent enough in it's stemming that it matches what we expect. This also tests for a proper import of the phraseinput.py file.

testPOS & testPOS2:

More minor tests to make sure our code to isolate the part of speech of a given string is working. The first test checks that it can correctly tag a noun as such, which is important because we use this to check the user's input for both name and food. testPOS2 checks the opposite, that is, will it not incorrectly tag something that isn't a noun. We pass it a verb and check to see if the value is a noun. If it is, the test fails. Otherwise, it returns a pass. This also tests for a proper import of the phraseinput.py file.

testNER:

We use the named entity recognition function in order to isolate the name of an organization from a string of text. To make sure this is working correctly, we pass it an example string that we expect the user might input, namely “I really like the Edmonton Oilers, they rock!” which, despite being something an insane person would say, is a realistic input. The test checks if the function returns “Edmonton Oilers” rather than ‘N/A’ which should only occur when the function can’t find an organization. This also tests for a proper import of the phraseinput.py file.

ADDITIONAL INFORMATION

There is one large notable exception in the test cases, that of the interface. Unfortunately, due to how python’s tkinter function works. Because our interface file doesn’t return anything, we can’t test it with our current setup.

Another note is that I mentioned imports *a lot*. This is because 1) Python’s NLTK has some oddities with how installing it works, and 2) having our code be in separate files to make it easier to work on does mean there’s a risk that imports might not work properly. Small odds, but always a risk.