COLLEGE of CHARLESTON

Mission

UndefinedOffsetNine's mission was to create a testing framework for the string comparison project FuzzyWuzzy which was developed by SeatGeek. To accomplish this mission we needed to develop a method to test individual modules within the projec.t We also needed to work together as a team, which was a new experience for many of us. As a team we needed to meet real world deadlines, and deliver incremental deliverables.

FuzzyWuzzy

FuzzyWuzzy is an open source string matching program that depends on the difflib python library. Difflib is a python library that gives classes and function for comparing sequences. In the case of FuzzyWuzzy, SequenceMatcher() is used and then multiple string processings are condensed into a single operation, in order to compare two strings and produce a percentage of how alike the two strings are. Different methods compare the strings in different ways from comparing if the strings are exactly alike to comparing if the same substrings are present within the strings.

About UndefinedOffsetNine

UndefinedOffsetNine is made up of Cortney Mood, Courtney Profera, Katherine Vaughan, and Ronald Zielaznicki. All four are Computer Science majors attending school at the College of Charleston. For the project we wanted to work with Python, a language we are all familiar with and were thoroughly entertained by the project's name "FuzzyWuzzy" which came to be the deciding factor in choosing the project. The requirements for this project were laid out by Dr. Jim Bowring for the CSCI362 Software Engineering course. We all found the project challenging and enjoyed the journey throughout this semester.

UndefinedOffsetNine The FuzzyWuzzy Project

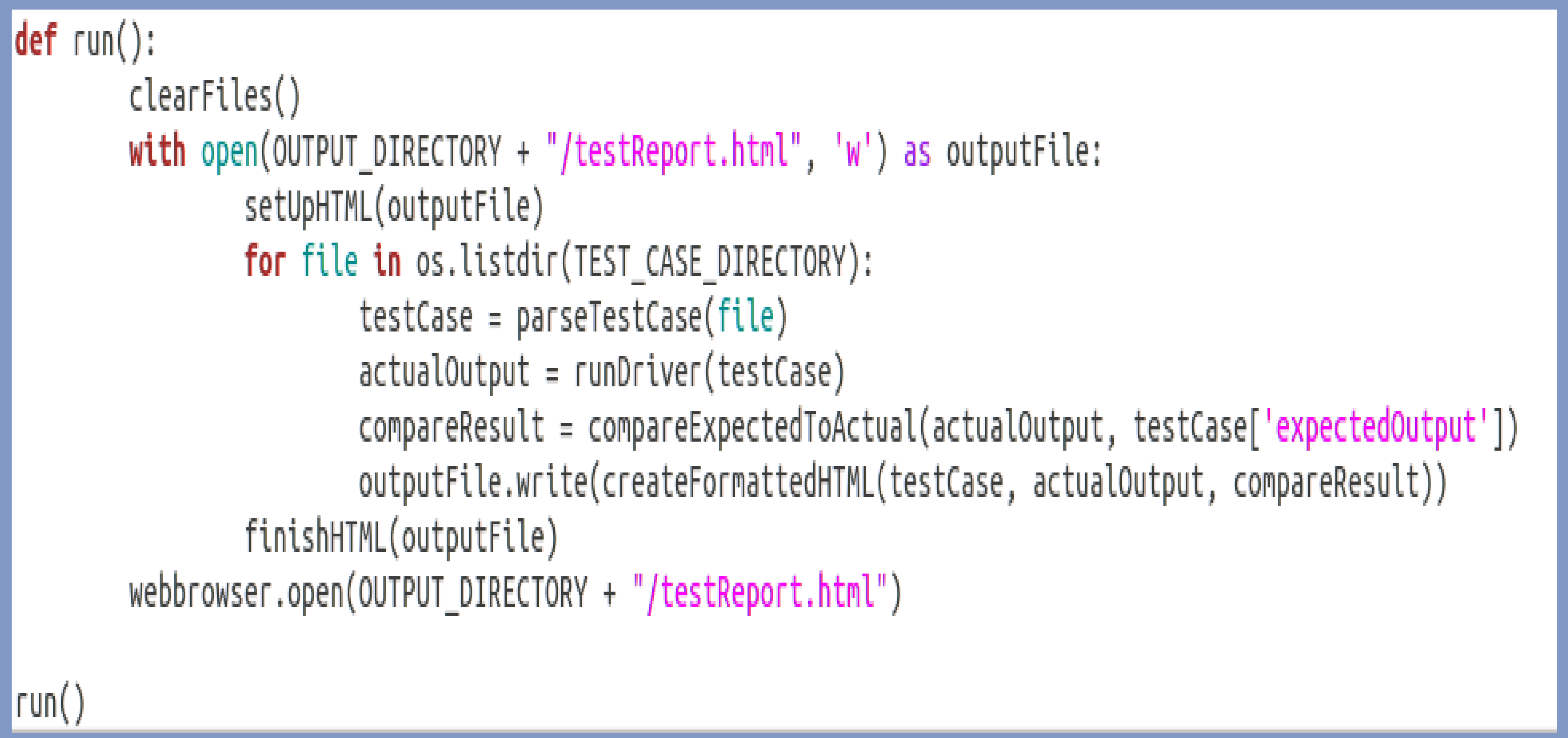
The Testing Framework

The framework operates through a script that runs a set of drivers, which are written in python. There is a driver for each method that is being tested: ratio, partial_ratio, token_set_ratio, and token_sort_ratio.

The script, runAllTests.py consists of methods that allow for the framework to work. Initially, the method clearFiles() will check if there is already a file with the name 'testReport.html' present, this is the file where the testing results are placed. If the file is present then it will be erased so that another blank file of the same name can be created in its place.

Then, with testReport.html set as the output file the basic html for the file, setUpHTML(outputFile), is written so the output is formatted into a table for ease of reading. The script moves to the TestCase directory where it parses the test cases, with parseTestCase(), to determine the test number, requirement, component, method, inputs, driver, and expected output.

From there each driver is run, runDriver(testCase), based on the driver given in the test case file. The driver evaluates the actual output received from the specified fuzzywuzzy method that is being tested and returns the actual output to be displayed in the testReport.html. The actual output and expected output are compared through compareExpectedToActual(actualOutput, expectedOutput) and a pass or fail rating is determined and sent to the html file.



```
def run():
        clearFiles()
        with open(OUTPUT_DIRECTORY + "/testReport.html", 'w') as outputFile:
                setUpHTML(outputFile)
                for file in os.listdir(TEST_CASE_DIRECTORY):
                        testCase = parseTestCase(file)
                        actualOutput = runDriver(testCase)
                        compareResult = compareExpectedToActual(actualOutput, testCase['expectedOutput'])
                        outputFile.write(createFormattedHTML(testCase, actualOutput, compareResult))
                finishHTML(outputFile)
        webbrowser.open(OUTPUT_DIRECTORY + "/testReport.html")

run()
```

Requirements Traceability and Inputs Chosen

Fuzzywuzzy is designed as a way to compare two strings. The methods must compare the accuracy of strings and how closely two strings are to each other in different fashions, from comparing if they are exactly the same or are similar, depending on the specific method.

Each of the methods is set to cover a different requirement and the tests have been created to reflect such with each test input set to cover a specific area of the vast input space available for string comparison testing.

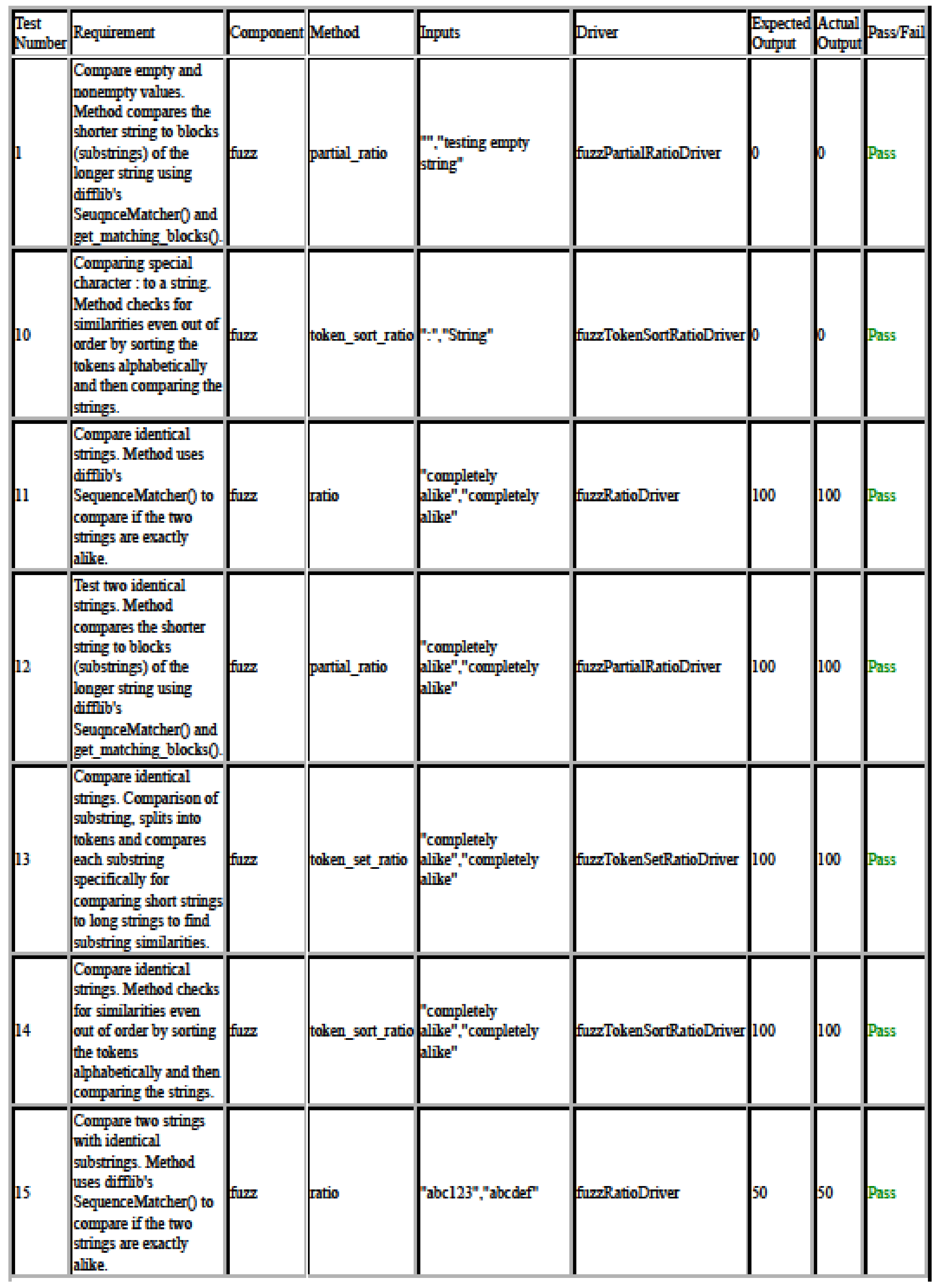
With string comparisons it can be tricky to divide up the input space to try to cover a wide range of territory to ensure that the testing presents a satisfactory sample that will prove that the method either works or does not. Basic ground that needed to be covered included empty strings, strings made of special characters, strings that matched exactly, strings with matching substrings, and strings that are completely unalike. It was decided that the for the most part, the four methods would be presented with the same input since each method compares strings differently and thus, in some cases, would produce differing results based on the input.

FuzzyWuzzy developed by

SeatGeek

Results

The overall result was that FuzzyWuzzy passed all of the tests that were presented within the testing framework. While it understandably handled basic strings well, it also was able to compare empty strings, and strings that included special characters like a colon.



Test Number	Requirement	Component	Method	Inputs	Driver	Expected Output	Actual Output	Pass/Fail
1	Compare empty and nonempty values. Method compares the shorter string to blocks (substrings) of the longer string using difflib's SeuqnceMatcher() and get_matching_blocks().	fuzz	partial_ratio	"","testing empty string"	fuzzPartialRatioDriver	0	0	Pass
10	Comparing special character : to a string. Method checks for similarities even out of order by sorting the tokens alphabetically and then comparing the strings.	fuzz	token_sort_ratio	":","String"	fuzzTokenSortRatioDriver	0	0	Pass
11	Compare identical strings. Method uses difflib's SequenceMatcher() to compare if the two strings are exactly alike.	fuzz	ratio	"completely alike","completely alike"	fuzzRatioDriver	100	100	Pass
12	Test two identical strings. Method compares the shorter string to blocks (substrings) of the longer string using difflib's SeuqnceMatcher() and get_matching_blocks().	fuzz	partial_ratio	"completely alike","completely alike"	fuzzPartialRatioDriver	100	100	Pass
13	Compare identical strings. Comparison of substring, splits into tokens and compares each substring specifically for comparing short strings to long strings to find substring similarities.	fuzz	token_set_ratio	"completely alike","completely alike"	fuzzTokenSetRatioDriver	100	100	Pass
14	Compare identical strings. Method checks for similarities even out of order by sorting the tokens alphabetically and then comparing the strings.	fuzz	token_sort_ratio	"completely alike","completely alike"	fuzzTokenSortRatioDriver	100	100	Pass
15	Compare two strings with identical substrings. Method uses difflib's SequenceMatcher() to compare if the two strings are exactly alike.	fuzz	ratio	"abc123","abcdef"	fuzzRatioDriver	50	50	Pass

Links for FuzzyWuzzy

Github: https://github.com/seatgeek/fuzzywuzzy

Difflib: https://docs.python.org/2/library/difflib.html

Explanation:http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/