Assignment One

Erick Costigan - 301098933
Kenny Chetal – 301100888

Specification

The program we are testing is a CSV to JSON converter using the python programming language. The program we are testing was found on github under user perhenrik90. The URL to the software can be found here https://github.com/perhenrik90/csv2json. We have modified the code to allow keyboard arguments so that the software can be tested with a test harness.

The input/output requirement of this software is that we input a valid csv file, and outputs a valid json file.

Input Space Partitioning

1. We are testing a python script that takes one command line argument. The argument is a path to a csv file which contains a set of inputs in csv format that will be converted to json format. Any environment that can run python will also be able to run the software.

Input Domain: All possible ASCII, except comma.

Characteristic	**Block 1**	**Block 2**
CSV File Format (Is .csv?)	T	F
Valid CSV (Same number of columns for each row)	T	F
Number Inputs (13,2,3) (Does our inputs have numbers)	T	F
Letters Inputs (a,b,c) (Does our inputs have letters)	T	F
Symbol Inputs (#,@,etc) (Does our input have symbols)	T	F

Constraint 1: If CSV file is not a csv format, we do not bother testing further (others characteristics must be false). This prevents us from testing meaningless cases, as it would not have the correct file to create a valid JSON file. If it is valid then move onto next test.

Constraint 2: If csv is not valid, we do not bother testing further (further (others characteristics must be false), this prevents us from testing cases where the csv does meet specifications. Some example include not having the same number of columns for each row.

Component Documentation:

Directories:

***ACTS File*:**

- Directory that contains the ACTS file of our pairwise testing.

Documents:

- Contains the assignment one test reports, input space partitioning, and other such documentation.

Expected Output:

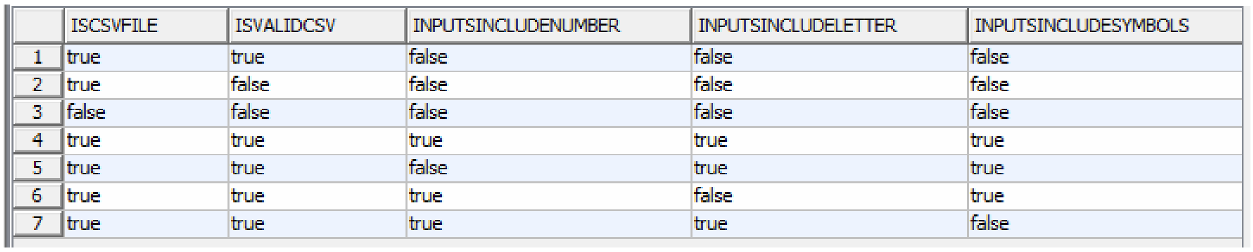
- Contains several json files that are the expected output for each of the seven tests. The test harness will automatically compare them to the output generated by our test suite.

Test Files:

- 473.py – Functional code to be tested. Takes in a csv file and outputs a corresponding json file.
- runtest.py – Test harness to run our suite of tests. Runs the seven tests and reports on the results by comparing them to the json files in "Expected output".
- Input files – Files t1.csv to t7.csv, input files for each test to be run.
- Output files – Files t1.json to t7.json, output files created by 473.py. Note that t3.json does not exist because it was the only test that failed. See test report for details.

Test Report:

Using our test harness (runtest.py) we ran seven pairwise tests in total generated in ACTS as seen



	ISCSVFILE	ISVALIDCSV	INPUTSINCLUDENUMBER	INPUTSINCLUDELETTER	INPUTSINCLUDESYMBOLS
1	true	true	false	false	false
2	true	false	false	false	false
3	false	false	false	false	false
4	true	true	true	true	true
5	true	true	false	true	true
6	true	true	true	false	true
7	true	true	true	true	false

bellow:

Of these tests all of them passed with expected output with the exception of test three. This is because it used a file format that was not CSV, and therefore it could not create an output json file, contrary to our expectations (we expected an error and empty json file to be generated).

We would have generated 10 tests if we did not use pair wise testing. For example if increase the strength from 2-wise to 3-wise testing we will see more tests and more combinations covered. Our tradeoff is that we are able to test all the interactions between our inputs, however we are missing certain combinations that could possibly have failed a test. For example in 3-wise testing True, True, False, False True appears as a test, but does not appear in 2-wise testing (pairwise).

We felt that this assignment was spot on in terms of the time required to complete in relation to the work necessary to complete it. As for difficulty, the hardest part was putting ourselves in the mindset necessary to divide the sample program into its input space partitions; while the writing of the actual test code took barely any time in comparison.