Cmpt 473: Exercise 2

Group Members:
Fiona Cun (301434437),
Maisha Supritee Chowdhury (301401482),
Minh Thanh Tran (301416848)

Program to Test: Nested JSON to CSV Converter
(https://github.com/vinay20045/json-to-csv)

Usage

To run the json_to_csv program:
`python /path/to/json_to_csv.py node json_in_file_path csv_out_file_path`

To run all tests, in the root directory terminal just run:

```
pytest
```

Specification of Nested JSON to CSV Converter

The json_to_csv program is a Python script that reads a JSON file, flattens nested JSON objects, and writes the result to a CSV file.

Input requirements:

- Python file path : `/path/to/json_to_csv.py`
- Name of node within the json file
    - If no node, can be any valid string. Cannot be empty.
- A file containing valid JSON. File extension should be .json.
- An empty csv file. File extension should be .csv.
    - If the csv file is not empty, the contents will be overwritten.

JSON Input Specifications:

- JSON data can be an array of node objects (an unordered set of key/value pairs)

For example:

```
//Simple
{
    "node":
        [{
            "item_1":"value_1",
            "item_2":"value_2"
            }]
}

//Nested
 {
        "node":[
```

```
            {
                "item_1":"value_1",
                "item_2":"value_2",
                "item_3":["sub_value_31", "sub_value_32"]
            },
            {
                "item_1":"value_4",
                "item_2":"value_5"
            }
        ]
    }
```

- JSON data can be a list of dictionaries/key value pairs.

For example:

```
 [
            {
                "item_1":"value_1",
                "item_2":"value_2",
                "item_3":["sub_value_31", "sub_value_32"]
            },
            {
                "item_1":"value_4",
                "item_2":"value_5"
            }
]
```

- Keys must be strings with double quotes (W3Schools.com, n.d.).
- Values must be a string, number (int or float), object (JSON object), array, boolean, or null (W3Schools.com, n.d.).
- String values can include special characters like \t, \n, \" \". The program will also include starting and trailing whitespaces.

CSV Output Specifications:

- The output of the program is a CSV file with flattened key names and their respective values, each field separated by commas.
- The key names are formed as follows: node_itemKey, and for arrays: node_itemKey_arrayIndex. (Converting Nested JSON to CSV 8 December 2013 - Vinay NP, n.d.)
- All similar keys stay next to each other as they are sorted alphabetically which means the original ordering of the JSON is not maintained.

For example:

```
//JSON
{
    "node":
        [{
            "item_1":"value_1",
            "item_2":["value_2", "value_3"]
            }]
}
```

```
//OUTPUT CSV
"node_item_1","node_item_2_0","node_item_2_1"

"value_11","value_12","value_13"
```

Input edge case:

- If the node name written during input doesn't match the node actually in the JSON file, the program will still run but the output csv will be incorrect as it only has the node names.

For example:

```
//JSON
{
    "fruit":[
        {
            "name":"Apple",
            "binomial name":"Malus domestica",
            "major_producers":[
                "China",
                "United States",
                "Turkey"
            ],
            "nutrition":{
                "carbohydrates":"13.81g",
                "fat":"0.17g",
                "protein":"0.26g"
```

```
            }
        }
    ]
}
```

```
//OUTPUT CSV

"items"

"fruit"
```

Edge Cases of JSON Structure:

- The program will accept JSON data with just a single object, but the produced csv output file for this is incorrect as it only has the keys and not the values. For example:

```
//JSON
{
    "item_1":"value_11",
    "item_2":"value_12",
    "item_3":["sub_value_14", "sub_value_15"]
}
```

```
//OUTPUT CSV
"items"

"item_1"

"item_2"

"item_3"
```

Input Space Partitioning

Components:

- 3 parameters when running the program from a terminal:
    - Node: <node>
    - JSON file path: <json_in_file_path>
    - CSV file path: <csv_out_file_path>

Characteristics:

- JSON structure
    - How the dictionaries are structured in the JSON file
    - (Array(Nested), List(Nested))
- JSON keys
    - JSON keys can be:
    - (Alphabetical_Order, Mixed_Order, Unique, Duplicate_Same_Obj, Duplicate_Diff_Obj)
- JSON values (Non-Empty)
    - What the values of the JSON file contain
    - (Special Characters, Null, Regular, Large_Number(?))
- Array/List length
    - Can be 1 or > 1
- Object/Dictionary length
    - Can be 1 or > 1
- Node name exists in JSON
    - Whether the node name passed in the parameter exists in the JSON file
    - Values: (true, false)

Input Domain Model & Constraints:

JSON File Structure	Nested (Array of nodes)	Nested (List of dictionaries)	
JSON Keys Order	Ordered alphabetically	Mixed order	
Duplicate keys in same object	True (All unique keys == False & Object length > 1)	False	
Duplicate keys in different objects	True (All unique keys == False & Array/List length > 1)	False	
All unique keys	True (Duplicate keys == False)	False	
Object/Dictionary length	1	>1	
Array/List Length	1	>1	
Json Value Contains Empty String	True	False !(All other contains == False)	
Json Value Contains special characters like \t, \n, “”, etc.	True !(Object length == 1 & Any other contains == True)	False !(All other contains == False)	
Json Value Contains Null	True !(Object length == 1 & Any other contains == True)	False !(All other contains == False)	
Json Value Contains regular input	True !(Object length == 1 & Any other contains == True)	False !(All other contains == False)	
Node name parameter exists in	True (JSON File Structure	False (JSON File Structure	

JSON	Nested Array of Nodes)	Nested Array of Nodes)	

Constraints:

Duplicate keys can only occur if the keys are not all unique and vice versa. Duplicate keys can only occur in different objects, i.e. multiple objects, if the array has a size larger than one. Duplicate keys in the same object/dictionary can only occur if its size is larger than one.

For the four JSON Value Contains, one of them has to be true at all times. If the object length is one, then only one can be true as there will only be a single value. A value cannot be empty and contain special characters, for example.

Node name is dependent on the structure of the JSON file. If it is a list, then Node Name is always false as lists do not use nodes.

Test Report

Number of Tests Generated:

Using ACTS, 13 tests were generated with our input domain model and constraints.

Number of Tests Passed:

6

Number of Tests Failed:

5
The cases that failed should fail as there are some cases that are not handled accurately by the original program, especially the case where the keys are not ordered alphabetically the program alphabetizes them in the output csv instead of keeping original order.

Number of Tests that would have been generated without pairwise testing:

Each Choice: 3 tests. This number feels like it doesn't provide adequate coverage of the system. With only 3 tests, it is difficult to have confidence that everything is working as intended without any issues.

Every Combination (Ignoring constraints): (2^12) x 3 = 12,288. Over 12,000 tests is a large amount, especially if they need to be run frequently. Because this is without considering constraints, some of those tests are impossible and therefore contain no value. They are essentially useless.

Tradeoffs of Pairwise Testing:

Pros:

Pairwise testing can be faster than testing every possible combination while providing more coverage than Each Choice. So while still ensuring confidence in the program we tested. It tests the relationships between two inputs and makes bugs more likely to be found.

Cons:

Missing Tests for Empty and Flat JSON File Structure.

For example:

- The program accepts empty JSON content such as {}, [{}], { "fruit": [{ },{ }] }, etc.

- The program also accepts flat JSON input even though it's coded to only format nested JSON input into csv. Such as {}, {"item1":"value1", "item2": "value2"}. The program should throw an error in this case as flat json values are not converted accurately or it shouldn't accept such input at all.

We can't test these cases using the pairwise tests, due to the constraints we created, none of those cases were generated. So some test cases are not possible through pairwise testing, and that's why constraints are added. So additional test cases would need to be added to supplement this..

Pairwise testing requires more thought and planning when identifying inputs and the relationships between them. Missing a constraint, for example, can lead to tests for situations that cannot occur. We repeatedly ran into that issue where we thought we had created all the necessary constraints, only to realize we needed more.

Reflection:

Coming up with a functional-based input domain model requires more thinking and planning than we initially expected. Previous experiences in class were with simple functions, not an entire program, so there was a struggle to define all the possible inputs.

Using ACTS was the easy part. With an input domain model built, you translate all that information into ACTS and it generates the pairwise tests for you.

A difficulty we encountered was realizing our input domain model required more refining and revisions during the ACTS portion. Sometimes we would come across a generated test that was impossible, f.e. having all the keys be unique *and* contain duplicates, so we had to add more constraints to counter these issues.

As mentioned earlier, we included empty input but then came across the problem where it was not included in any of the generated tests. For example, we have a characteristic for what the JSON values contain, and we had it so that all the contain values had to be false if empty was true. But what this did was make it impossible to form a pair between empty and any of the values in JSON contains. Therefore, empty could not be a valid pairwise test.

What would have made things easier at a technical level is going through more complex examples in class and perhaps some pointers for ways to identify the characteristics of an input domain. When we made our characteristics, they kept growing when we realized there was more we didn't consider. For example, length was something we didn't think about until much later.

References

Converting Nested JSON to CSV 8 December 2013 - Vinay NP. (n.d.). https://askvinay.com/post/converting-nested-json-to-csv-8-december-2013.html

W3Schools.com. (n.d.). https://www.w3schools.com/js/js_json_datatypes.asp