JumpCloud Interview Assignment

Summary

This assignment is developed in Python and tested using python3 version 3.9.0. The code is organized as a project directory which has the source code and the associated unit test files.

Project Organization

The functionality is implemented as a Python module name 'my_jc'. This module is available as a sub-directory under the main 'jc_project' directory. The project directory structure is as follows:

```
---jc_project
        |-------------------my_jc
        |---jc_test.py      |---jc.py
        |---jc_run.py       |---json_proc.py
                            |---__init__.py
```

File details:

- jc.py: Main file implementing the required library class (ActionProcessor) for this assignment. It provides addAction(), getStats() and (an additional) getErrorMsg() methods
- json_proc.py: This file implements a lightweight JSON tokenizer/parser. It is extensible to add new functionality by adding new methods to the class (JsonProcessor). It is leveraged by ActionProcessor methods.
- jc_test.py: This file implements the unit test cases for this functionality. It is built leveraging Python unittest framework
- jc_run.py: sample file to do a standalone demo run
- __init__.py: module initialization file

Use

From Python Code:

```
import my_jc
        -or-
from my_jc import ActionProcessor

o = ActionProcessor()
v = o.addAction('json-string')
v = o.addAction('json-string')
.
.
.
s = o.getStats()
print(s)
```

Standalone:

The code can also be run standalone to get a sample output as follows:

```
# cd jc_project
# python3 jc_run.py
```

Testing

The code was unit tested with a mix of negative, functionality and functionality+negative cases. The python unittest framework was leveraged to build unit-test automation.

Here are the details:

- Total Test Cases: 50
  - Negative Tests: 32
  - Functionality Tests: 11
  - Functionality + Negative Tests: 7
- Unit test automation: jc_project/jc_test.py

Unit tests can be executed as follows:
cd jc_project
python3 jc_test.py

Library APIs

- addAction() takes json serialized string and returns 0 on success or -1 on error. A detailed error message can be retrieved by calling getErrorMsg() class method.
- getStats() returns a json serialized string as per requirement
- getErrorMsg(): Returns an string detailing the error encountered during addAction() processing

Assumptions

The functionality is implemented using the following assumptions. Constants are not hardcoded and can be changed easily

- Leading and trailing whitespaces are removed during processing. For example, “action “, “ action” and “ action “ are treated same as “action”
- Strings with whitespaces only are treated as null strings. For example, “action”:” “ will be processed as “action”:”” and flagged as error
- String lengths are limited to 20 characters in length. This is controlled through a macro and easily modified/removed
- Numbers are limited to 10 numbers in length. This is controlled through a macro and easily modified/removed
- Only non-negative numbers (integers and floats) are supported.
- If the average is a floating number then result is rounded to 2 decimal. Care is taken to avoid displaying decimal point when not needed (average for 10 and 20 is displayed as 15 and not 15.0)
- A value of 0 (or 0.0) is not used when calculating average. For example if there are two time measurement inputs with 10 and 0, the average is calculated as 10 and not 5 (10 + 0/2). This is easily modifiable/removed

Sample Outputs

```
# Input:
# addAction('{" action":"run", "time":10}')
# addAction('{" action":"jump", "time":100}')
# addAction('{"action ":"run", "time":20}')

# cd jc_project
# python3 jc_run.py
[
        {"action":"run", "avg":15},
        {"action":"jump", "avg":100}
]

# cd jc_project
# python3 jc_test.py
..................................................
----------------------------------------------------------------------
Ran 50 tests in 0.006s

OK
```