Final Project 2023S SSW567-WS

SSW 567-WS
Prof. Andre Bondi
Part2-UnitTesting
Team: C

Quality Analysts
Arun Rao Nayineni
Dhruv Patel
Ruchitha Paithankar

Part2-UnitTesting:

GitHub Repository: https://github.com/ArunRao1997/SSW-567-Final-Project

Code - Created and used two files as a part of unit testing.

MRTD.py - Contains the code for decode logic (decodes line 1 and line 2 and checks the value if they had proper check digits) and encode logic (encodes the given values and provides the result in the form of the line as expected).

MTTDtest.py - Contains the Python test cases which implement all the methods of MRTD.py.

Report and Results:

Python Code:

MRTD.py

1. **char_to_value(char):** Converts a given character to its corresponding numeric value based on the CHAR_DICT. Used in the check digit algorithm.
2. **get_check_digit(input_str):** Computes the check digit for a given input string of alphanumeric characters using a weighted sum.
3. **scan_passport():** A placeholder function that represents scanning a passport, returning a 'scanned' string.
4. **extract_line1(line):** Extracts data (issuing country, last name, and given name) from the first line of the passport string.
5. **extract_line2(line):** Extracts data (passport number, country code, birth date, sex, expiry date, and personal number) from the second line of the passport string.
6. **decode(string):** Parses a given MRZ string, extracting and validating the data, and returns the data as a JSON object if valid.
7. **encode(data):** Converts a JSON object containing MRZ data into a formatted MRZ string by encoding line1 and line2.
8. **encode_line1(data):** Accepts a dictionary containing line1 data and generates the corresponding MRZ string.

9. **encode_line2(data):** Accepts a dictionary containing line2 data and generates the corresponding MRZ string.

MTTDtest.py

1. **test_decode:** Tests if the decode function successfully decodes lines 1 and 2 of an MRTD.
2. **test_decode2:** Tests if the decode function handles different country codes correctly.
3. **test_find_val1:** Tests the extract_line1 function with valid input for line 1.
4. **test_find_val2:** Tests the extract_line1 function with input missing document type.
5. **test_find_val3:** Tests the extract_line2 function with valid input for line 2.
6. **test_extract_line1_no_middle_name:** Tests extract_line1 function when middle name is missing.
7. **test_encode_data_in_mrtd_format:** Tests the encode function with valid input data.
8. **test_encode_line1_data:** Tests the encode_line1 function with valid input data.
9. **test_encode_line1_missing_given_name:** Tests encode_line1 function when given name is missing.
10. **test_encode_line1_missing_last_name:** Tests encode_line1 function when last name is missing.
11. **test_encode_line1_missing_issuing_country:** Tests encode_line1 function when issuing country is missing.
12. **test_encode_line1_missing_all_data:** Tests encode_line1 function when all data is missing.
13. **test_encode_line2:** Tests the encoding of line 2 data in MRTD.
14. **test_encode_line2_missing_country_code:** Tests encoding of line 2 data when country code is missing.
15. **test_encode_line2_mutated_personal_number:** Tests encoding of line 2 data with a mutated personal number.
16. **test_encode_line2_mutated_expiration_date:** Tests encoding of line 2 data with a mutated expiration date.
17. **test_encode_line2_mutated_sex:** Tests encoding of line 2 data with a mutated sex value.
18. **test_encode_line2_empty:** Tests encoding an empty passport information line.
19. **test_scan_passport:** Tests if the passport is scanned successfully.
20. **test_extract_line2_invalid_check_digit:** Tests if extracting line 2 with an invalid check digit raises a ValueError.
21. **test_birthday_check_digit_invalid:** Tests if extracting line 2 with an invalid birthday check digit raises a ValueError.

22. **test_expiry_date_check_digit_invalid:** Tests if extracting line 2 with an invalid expiry date check digit raises a ValueError.
23. **test_personal_number_check_digit_invalid:** Tests if extracting line 2 with an invalid personal number check digit raises a ValueError.
24. **test_get_check_digit:** Tests if the get_check_digit method returns the correct check digits.
25. **test_get_check_digit_edge_cases:** Tests if the get_check_digit method handles edge cases correctly.
26. **test_sqlite3_success:** This test checks if the DataBaseClass creates a successful connection to an SQLite database with the correct name and asserts the connection status.
27. **test_sqlite3_fail:** This test verifies if the DataBaseClass handles a failed connection properly and asserts the connection status as 'connection failed'.
28. **test_sqlite3_connect_with_side_effect:** This test evaluates if the DataBaseClass handles different connection strings correctly by using a side_effect and checking the connection status based on the provided string.

Python Unit Tests:

Python unit tests are code snippets written with the unittest library to validate the functionality of other code, ensuring correctness and absence of bugs.

Importance:

- Unit tests help identify code issues, ensuring changes don't break intended functionality, saving time and effort in development.
- They contribute to a high-quality final product.

Python Code Coverage Code coverage measures the proportion of a Python project's code executed during testing, indicating the quality and effectiveness of tests.

Advantages:

- High code coverage provides confidence that tests cover all code, including edge cases and potential bugs.
- It helps prevent issues from being overlooked, ensuring high-quality software.
- Code coverage identifies areas needing more testing, leading to comprehensive evaluation.

```
Launching pytest with arguments C:/Users/Arun Rao Nayineni/SSW-567-Final-Project/MTTDtest.py --no-header --no-summary -q in C:\Users\Arun Rao

============================ test session starts ============================
collecting ... collected 28 items

MTTDtest.py::TestDecodeEncode::test_birthday_check_digit_invalid PASSED  [  3%]
MTTDtest.py::TestDecodeEncode::test_decode PASSED                        [  7%]
MTTDtest.py::TestDecodeEncode::test_decode2 PASSED                       [ 10%]Error: invalid MRZ data

MTTDtest.py::TestDecodeEncode::test_encode_data_in_mrtd_format PASSED    [ 14%]
MTTDtest.py::TestDecodeEncode::test_encode_line1_data PASSED             [ 17%]
MTTDtest.py::TestDecodeEncode::test_encode_line1_missing_all_data PASSED [ 21%]
MTTDtest.py::TestDecodeEncode::test_encode_line1_missing_given_name PASSED [ 25%]
MTTDtest.py::TestDecodeEncode::test_encode_line1_missing_issuing_country PASSED [ 28%]
MTTDtest.py::TestDecodeEncode::test_encode_line1_missing_last_name PASSED [ 32%]
MTTDtest.py::TestDecodeEncode::test_encode_line2 PASSED                  [ 35%]
MTTDtest.py::TestDecodeEncode::test_encode_line2_empty PASSED            [ 39%]
MTTDtest.py::TestDecodeEncode::test_encode_line2_missing_country_code PASSED [ 42%]
MTTDtest.py::TestDecodeEncode::test_encode_line2_mutated_expiration_date PASSED [ 46%]
MTTDtest.py::TestDecodeEncode::test_encode_line2_mutated_personal_number PASSED [ 50%]
MTTDtest.py::TestDecodeEncode::test_encode_line2_mutated_sex PASSED      [ 53%]
MTTDtest.py::TestDecodeEncode::test_expiry_date_check_digit_invalid PASSED [ 57%]
MTTDtest.py::TestDecodeEncode::test_extract_line1_no_middle_name PASSED  [ 60%]
MTTDtest.py::TestDecodeEncode::test_extract_line2_invalid_check_digit PASSED [ 64%]
MTTDtest.py::TestDecodeEncode::test_find_val1 PASSED                     [ 67%]
MTTDtest.py::TestDecodeEncode::test_find_val2 PASSED                     [ 71%]
MTTDtest.py::TestDecodeEncode::test_find_val3 PASSED                     [ 75%]
```

Fig1: Tests Passed

```
PS C:\Users\Arun Rao Nayineni\SSW-567-Final-Project-Group-C> coverage report
Name            Stmts   Miss  Cover
---------------------------------
MRTD.py            80      1    99%
MTTD_test.py       95      4    96%
---------------------------------
```

Fig2: Command Prompt Report

File | C:/Users/Arun%20Rao%20Nayineni/SSW-567-Final-Project/htmlcov/index.html

Spring JDBC Examp... datetime2: date an... Python List (With Ex... 5. Data Structures... 3. An Inf

Coverage report: 99%

Module ↑	statements	missing	excluded	coverage
MRTD.py	79	0	0	100%
MTTDtest.py	104	2	0	98%
Total	**183**	**2**	**0**	**99%**

Fig3: HTML Report After Optimizing the Code

Note: In **MRTD.py**, 100% of code coverage is achieved.

In **MTTDtest.py**, 98% of code coverage is achieved.

Pylint: Pylint is a tool measuring Python code quality.

```
Your code has been rated at 7.92/10 (previous run: 7.14/10, +0.78)

PS C:\Users\Arun Rao Nayineni\SSW-567-Final-Project> pylint MRTD.py
************* Module MRTD
MRTD.py:179:0: C0301: Line too long (110/100) (line-too-long)
MRTD.py:180:0: C0301: Line too long (101/100) (line-too-long)
MRTD.py:182:0: C0301: Line too long (104/100) (line-too-long)
MRTD.py:187:0: C0305: Trailing newlines (trailing-newlines)
MRTD.py:1:0: C0103: Module name "MRTD" doesn't conform to snake_case naming style (invalid-name)
MRTD.py:84:0: C0116: Missing function or method docstring (missing-function-docstring)
MRTD.py:141:0: C0116: Missing function or method docstring (missing-function-docstring)

------------------------------------------------------------------
Your code has been rated at 9.09/10 (previous run: 9.09/10, +0.00)
```

Fig4: Pylint coverage for MRTD.py

```
MTTDtest.py:144:4: C0116: Missing function or method docstring (missing-function-docstring)
MTTDtest.py:158:4: C0116: Missing function or method docstring (missing-function-docstring)
MTTDtest.py:263:4: C0116: Missing function or method docstring (missing-function-docstring)
MTTDtest.py:281:4: C0116: Missing function or method docstring (missing-function-docstring)
MTTDtest.py:286:4: C0116: Missing function or method docstring (missing-function-docstring)
MTTDtest.py:290:4: C0116: Missing function or method docstring (missing-function-docstring)
MTTDtest.py:351:0: R0903: Too few public methods (0/2) (too-few-public-methods)

------------------------------------------------------------------
Your code has been rated at 7.33/10 (previous run: 7.07/10, +0.26)

PS C:\Users\Arun Rao Nayineni\SSW-567-Final-Project>
```

Fig5: Pylint coverage for MTTDtest.py

```
PS C:\Users\Arun Rao Nayineni\SSW-567-Final-Project> pylint MTTDtest.py
************* Module MTTDtest
MTTDtest.py:292:0: C0301: Line too long (102/100) (line-too-long)
MTTDtest.py:335:0: C0301: Line too long (102/100) (line-too-long)
MTTDtest.py:342:0: C0301: Line too long (106/100) (line-too-long)
MTTDtest.py:1:0: C0114: Missing module docstring (missing-module-docstring)
MTTDtest.py:1:0: C0103: Module name "MTTDtest" doesn't conform to snake_case naming style (invalid-name)
MTTDtest.py:365:4: W0105: String statement has no effect (pointless-string-statement)
MTTDtest.py:9:0: R0904: Too many public methods (29/20) (too-many-public-methods)
MTTDtest.py:403:0: R0903: Too few public methods (0/2) (too-few-public-methods)

------------------------------------------------------------------
Your code has been rated at 9.31/10 (previous run: 9.31/10, +0.00)

PS C:\Users\Arun Rao Nayineni\SSW-567-Final-Project>
```

Fig6: Improved Pylint coverage for MTTDtest.py After Code Optimization

MutPy:

MutPy is a Python mutation testing tool that identifies code weaknesses by introducing mutations, and evaluating if tests can detect them.

Mutations: killed, survived, time out

- **Killed**: Tests detect and fail on the mutation, indicating correct code handling.
- **Survived**: Tests don't detect the mutation, revealing weakness and the need for additional tests.
- **Time out:** Mutations taking too long to run, not producing results within specified limits.

MutPy Score:

The MutPy score percentage measures unit test quality, calculated by dividing killed mutations by total mutations. Higher scores indicate better bug detection.

Results for MRTD.py and MTTDtest.py:

- 69 total mutants; 4 killed, 65 incompetent.

Improvements based on MutPy results:

1. Increase MutPy test coverage by adding diverse test cases.
2. Use MutPy results to identify insufficiently tested code areas, creating additional test cases.
3. Analyze MutPy results to remove redundant or unnecessary test cases.
4. Enhance the test suite by adding negative test cases for unexpected input or conditions.
5. Use MutPy results to detect potential bugs introduced by mutations and create test cases for them.
6. We have created additional test cases to achieve a 100% mutation score.

   List of added test cases:
   1. **test_encode_line1_missing_all_data:** Tests encode_line1 function when all data is missing.
   2. **test_encode_line2_mutated_personal_number:** Tests encoding of line 2 data with a mutated personal number.
   3. **test_encode_line2_mutated_expiration_date:** Tests encoding of line 2 data with a mutated expiration date.
   4. **test_encode_line2_mutated_sex:** Tests encoding of line 2 data with a mutated sex value.
   5. **test_encode_line2_empty:** Tests encoding an empty passport information line.
   6. **test_expiry_date_check_digit_invalid:** Tests if extracting line 2 with an invalid expiry date check digit raises a ValueError.
   7. **test_get_check_digit_edge_cases:** Tests if the get_check_digit method handles edge cases correctly.

These additional test cases helped increase the mutation score and ensured that the code is robust and reliable.

```
[*] Mutation score [12.79638 s]: 100.0%
   - all: 69
   - killed: 4 (5.8%)
   - survived: 0 (0.0%)
   - incompetent: 65 (94.2%)
   - timeout: 0 (0.0%)
```

Fig 6: Mutation Score report

Note: Mutation Score - 100%

CircleCi:

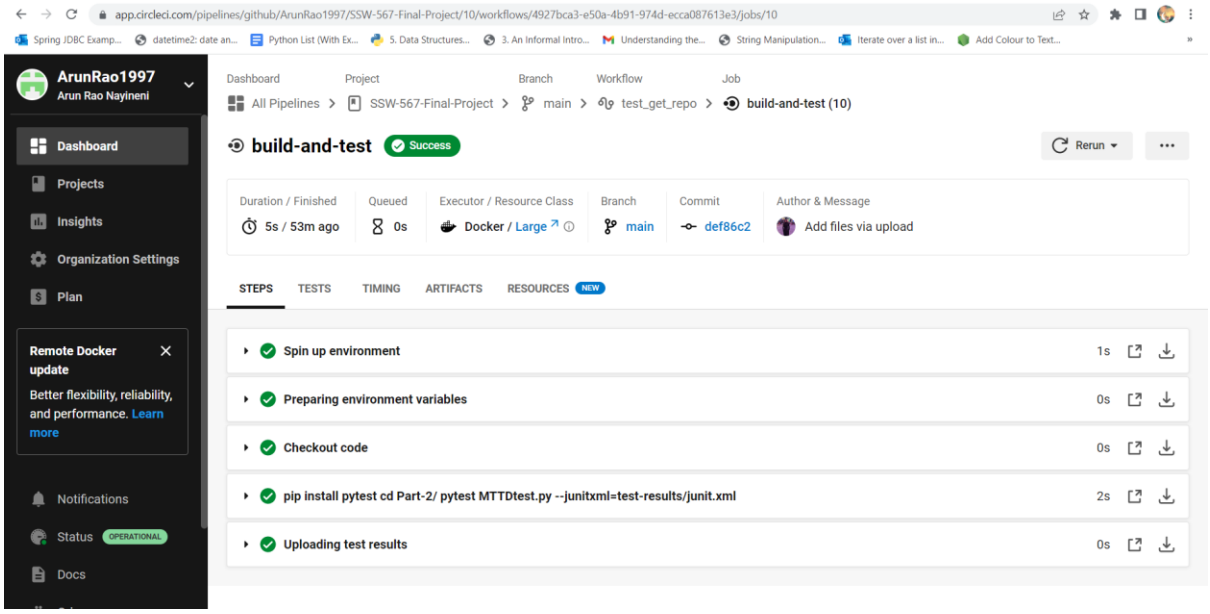

Fig 7: Build Passed on CircleCi for the tests