Part 2 – Unit Testing

Code Repository: [Tyler-Seliber/SSW-567-Project (github.com)]

Functions Created:

- checkField(): helper function to verify that the check digits for a given field are correct
- query_db(): unimplemented function that queries a database of stored travelers for requirement 3
- scanMRZ(): unimplemented function that scans a traveler document and provides strings for the two lines in the machine-readable zone
- decodeMRZ(): function to decode the two lines from the machine-readable zone and return Python dictionaries of all the data
- encodeTravelInfo(): function to lookup a travler's information by their personal number in the traveler database and encode their information into the two lines of the machine-readable zone
- checkMRZ(): function to check the fields of the machine-readable zones that have check digits and returns which check digits are invalid

Test Cases Created:

- test_scanMRZ: tests the scanMRZ() function by mocking a travel document that will be scanned by the designated hardware and verifies that the lines on the document are readable
- test_decodeMRZ: tests the decodeMRZ() function by passing in two lines of machine-readable strings and verifying that the provided output matches what is expected
- test_encode_travel_info: tests the encodeTravelInfo() function by mocking database return values and verifying the data is properly encoded
- test_checkMRZ: tests the checkMRZ() function to verify that the check digits are properly calculated and checked for particular fields of the machine-readable lines

Coverage Report:

```
tyler@Tylers-MacBook-Pro SSW-567-Project % coverage run MTTDtest.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.002s

OK
tyler@Tylers-MacBook-Pro SSW-567-Project % coverage report -m
Name          Stmts   Miss  Cover   Missing
-------------------------------------------
MRTD.py          53      2    96%   70, 75
MTTDtest.py     134      0   100%
-------------------------------------------
TOTAL           187      2    99%
```

- The two lines that are missing are 'pass' lines that are parts of the unimplemented functions for requirements 1 and 3. Had these functions been completely implemented, then the code coverage would be 100%.

I pledge my honor that I have abided by the Stevens Honor System.

MutPy Report:

```
[*] Mutation score [28.83307 s]: 0.0%
   - all: 63
   - killed: 0 (0.0%)
   - survived: 63 (100.0%)
   - incompetent: 0 (0.0%)
   - timeout: 0 (0.0%)
```

- The mutation score was surprisingly 0%. This means that the test cases unsuccessfully killed any single mutant created by MutPy.
- This was interesting as it appeared that manually making any of the changes MutPy did would cause at least one of the tests to fail.
- This discrepancy may be because the test cases are only testing specific portions of the final result. If the tests were modified to check for some of the intermediate values being generated or processed in the functions, perhaps the mutants would not have survived.
- Doing so would require extensive re-writing and expansions of the test cases, which requires a significant time investment. The additional time and effort needed for mutation testing is one of the practice's largest compromises, but it can result in having more robust and code and test cases with better coverage.

I pledge my honor that I have abided by the Stevens Honor System.