Part 2 Report

Team Members:

1. Nikhil Kumar G
2. Ruifeng Zhang
3. Shiv Chirayu Shah
4. Shashank Ramesh Kumar

Repo URL : https://github.com/nikhil-01a/567FinalProject

1. **Function Implementation:**

   - **class HardwareScanner():** An empty class function for hardware scanner that will be mocked to return actual values during testing.
   - **decode():** A function that takes 2 MRZ Strings as its inputs from the mocked HardwareScanner() and returns a string of '4 check digits concatenated together' and a dictionary of all the fields.
   - **class dummydatabase():** An empty class function for database scanner that will be mocked to return actual values during testing.
   - **encode():** A function that takes 2 dictionary lines as input from the mocked dummydatabase() and returns a string of ' 4 check digits concatenated together' and a string of MRZ lines joined by a semi-colon ';'.
   - **getCheckCode(inputString: str) -> int**: return the correct check code for the given string.
   - **compare_EndDec():** A function that compares the check digit output of both encode() and decode() and then it reports if any mismatch is found in the corresponding fields of those check digits.

2. **Test Case Functions:**

   - **test_decode_firstmiddle_name():** (Used mock with this)
     Testing decode function with Mocking HardwareScanner() and giving two strings as input to the decode function. This string contains a given name with first and middle names inside it.

   - **test_encode_firstmiddle_name():** (Used mock with this)
     Testing encode function with Mocking the dummydatabase() and giving two dictionary lines containing all the information fields of a person. Here, again the given name of the person includes first and middle name in it.

- **test_decode_givenname():** (Used mock with this)
  Testing decode function with Mocking HardwareScanner() and giving two strings as input to the decode function. This string contains a single given name.

- **test_encode_givenname():** (Used mock with this)
  Testing encode function with Mocking the dummydatabase() and giving two dictionary lines containing all the information fields of a person. This person has a single given name.

- **test_getCheckCode():**
  Testing the check digit generator function by giving a passport number as an input.

- **test_NoMismatch():**
  Testing the 'encoder-decoder check digits comparing' function with both equal check digits.

- **test_MismatchPassNo():**
  Testing the 'encoder-decoder check digits comparing' function with 'passport number' not matching (the first digit).

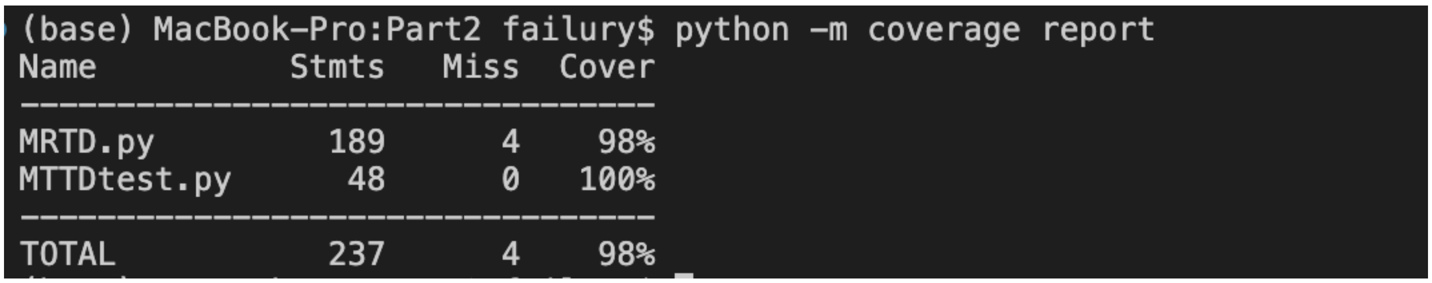
- **test_MismatchDOB():**
  Testing the 'encoder-decoder check digits comparing' function with 'Date of Birth' not matching (the second digit).

- **test_MismatchDOE():**
  Testing the 'encoder-decoder check digits comparing' function with 'Date of Expiration' not matching (the third digit).

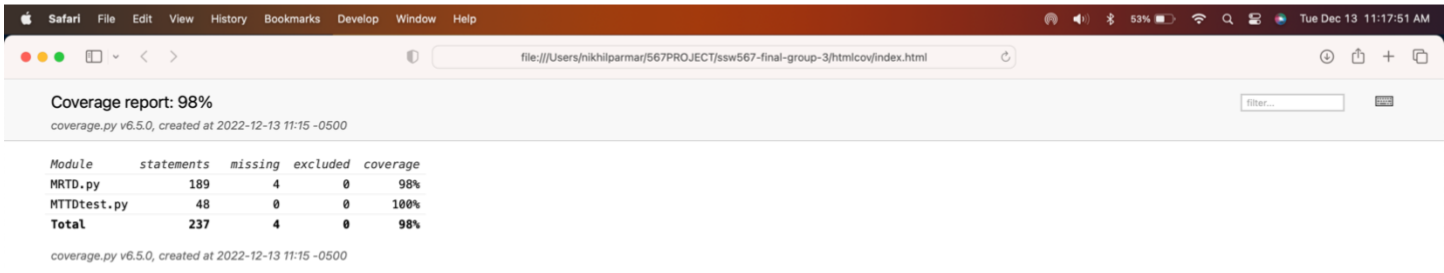
- **test_MismatchPersNo():**
  Testing the 'encoder-decoder check digits comparing' function with 'Personal Number' not matching (the fourth digit).

3. Unit Testing:

Coverage report:



```
(base) MacBook-Pro:Part2 failury$ python -m coverage report
Name          Stmts   Miss  Cover
---------------------------------
MRTD.py         189      4    98%
MTTDtest.py      48      0   100%
---------------------------------
TOTAL           237      4    98%
```


Safari File Edit View History Bookmarks Develop Window Help
Tue Dec 13 11:17:51 AM
file:///Users/nikhilparmar/567PROJECT/ssw567-final-group-3/htmlcov/index.html
filter...

Coverage report: 98%

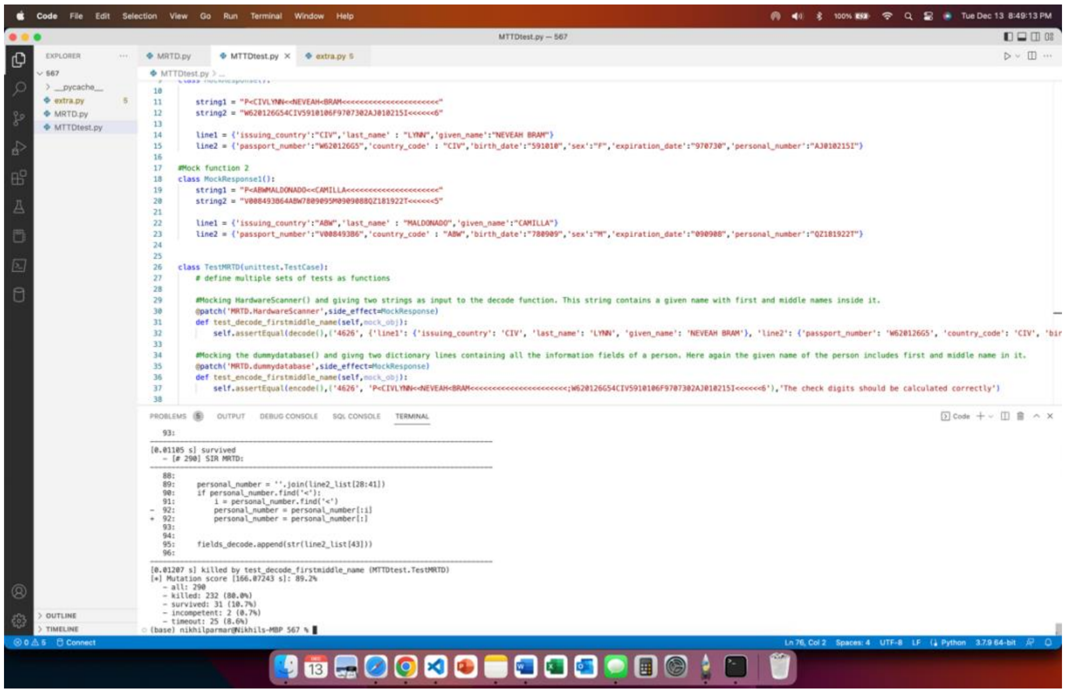
coverage.py v6.5.0, created at 2022-12-13 11:15 -0500

Module	statements	missing	excluded	coverage
MRTD.py	189	4	0	98%
MTTDtest.py	48	0	0	100%
Total	**237**	**4**	**0**	**98%**

coverage.py v6.5.0, created at 2022-12-13 11:15 -0500

4. Mutation testing

MutPy Report:



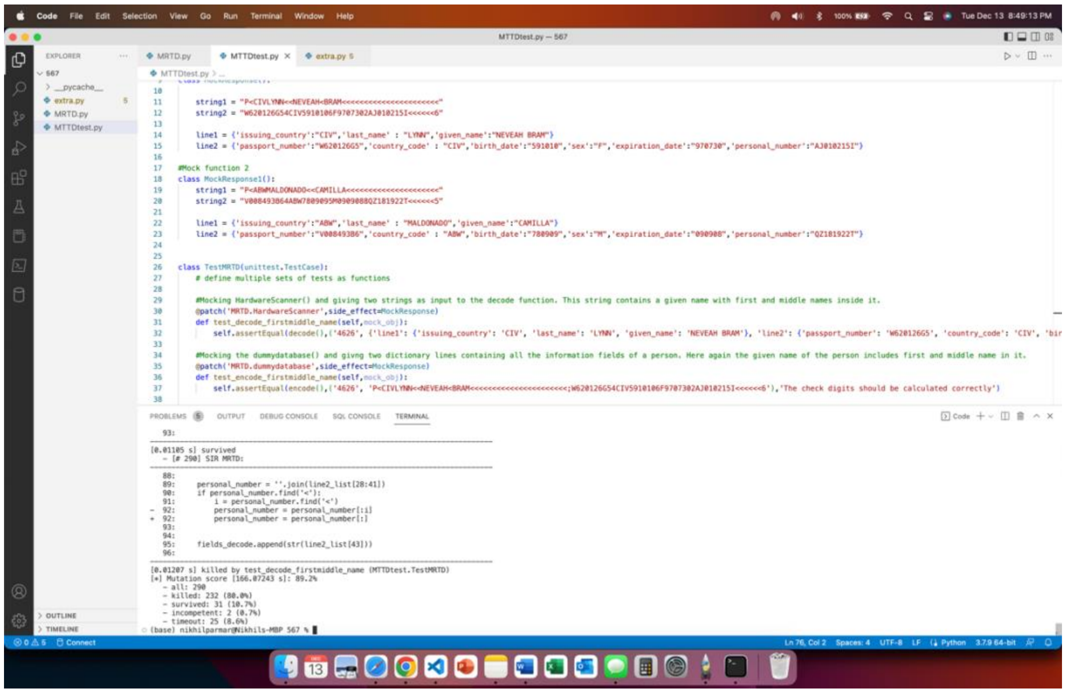
```python
    string1 = "P<CIVLYNN<<NEVEAH<BRAM<<<<<<<<<<<<<<<<<<<<<<"
    string2 = "W620126G54CIV5910106F9707302AJ010215I<<<<<<6"

    line1 = {'issuing_country':"CIV",'last_name' : "LYNN",'given_name':"NEVEAH BRAM"}
    line2 = {'passport_number':"W620126G5",'country_code' : "CIV",'birth_date':"591010",'sex':"F",'expiration_date':"970730",'personal_number':"AJ010215I"}

#Mock function 2
class MockResponse1():
    string1 = "P<ABWMALDONADO<<CAMILLA<<<<<<<<<<<<<<<<<<<<<"
    string2 = "V008493B64ABW7809095M0909088QZ181922T<<<<<<5"

    line1 = {'issuing_country':"ABW",'last_name' : "MALDONADO",'given_name':"CAMILLA"}
    line2 = {'passport_number':"V008493B6",'country_code' : "ABW",'birth_date':"780909",'sex':"M",'expiration_date':"090908",'personal_number':"QZ181922T"}


class TestMRTD(unittest.TestCase):
    # define multiple sets of tests as functions

    #Mocking HardwareScanner() and giving two strings as input to the decode function. This string contains a given name with first and middle names inside it.
    @patch('MRTD.HardwareScanner',side_effect=MockResponse)
    def test_decode_firstmiddle_name(self,mock_obj):
        self.assertEqual(decode(),('4626', {'line1': {'issuing_country': 'CIV', 'last_name': 'LYNN', 'given_name': 'NEVEAH BRAM'}, 'line2': {'passport_number': 'W620126G5', 'country_code': 'CIV', 'bir

    #Mocking the dummydatabase() and givng two dictionary lines containing all the information fields of a person. Here again the given name of the person includes first and middle name in it.
    @patch('MRTD.dummydatabase',side_effect=MockResponse)
    def test_encode_firstmiddle_name(self,mock_obj):
        self.assertEqual(encode(),('4626', 'P<CIVLYNN<<NEVEAH<BRAM<<<<<<<<<<<<<<<<<<<<<<;W620126G54CIV5910106F9707302AJ010215I<<<<<<6'),'The check digits should be calculated correctly')
```

```
   93:
--------------------------------------------------------------------------------
[0.01105 s] survived
   - [# 290] SIR MRTD:
--------------------------------------------------------------------------------
   88:
   89:     personal_number = ''.join(line2_list[28:41])
   90:     if personal_number.find('<'):
   91:         i = personal_number.find('<')
-  92:         personal_number = personal_number[:i]
+  92:         personal_number = personal_number[:]
   93:
   94:
   95:     fields_decode.append(str(line2_list[43]))
   96:
--------------------------------------------------------------------------------
[0.01207 s] killed by test_decode_firstmiddle_name (MTTDtest.TestMRTD)
[*] Mutation score [166.07243 s]: 89.2%
   - all: 290
   - killed: 232 (80.0%)
   - survived: 31 (10.7%)
   - incompetent: 2 (0.7%)
   - timeout: 25 (8.6%)
(base) nikhilparmar@Nikhils-MBP 567 %
```

- **How many mutants are generated based on your functions?**

  Based on my functions, there were a total of 290 mutants generated.

- **How many mutants are killed by your test cases; how many mutants survived your test cases? Discuss how you could improve your test cases based on results from MutPy.**



```python
    string1 = "P<CIVLYNN<<NEVEAH<BRAM<<<<<<<<<<<<<<<<<<<<<"
    string2 = "W620126G54CIV5910106F9707302AJ010215I<<<<<<6"

    line1 = {'issuing_country':"CIV",'last_name' : "LYNN",'given_name':"NEVEAH BRAM"}
    line2 = {'passport_number':"W620126G5",'country_code' : "CIV",'birth_date':"591010",'sex':"F",'expiration_date':"970730",'personal_number':"AJ010215I"}

#Mock function 2
class MockResponse1():
    string1 = "P<ABWMALDONADO<<CAMILLA<<<<<<<<<<<<<<<<<<<<<"
    string2 = "V008493B64ABW7809095M0909088QZ181922T<<<<<<5"

    line1 = {'issuing_country':"ABW",'last_name' : "MALDONADO",'given_name':"CAMILLA"}
    line2 = {'passport_number':"V008493B6",'country_code' : "ABW",'birth_date':"780909",'sex':"M",'expiration_date':"090908",'personal_number':"QZ181922T"}


class TestMRTD(unittest.TestCase):
    # define multiple sets of tests as functions

    #Mocking HardwareScanner() and giving two strings as input to the decode function. This string contains a given name with first and middle names inside it.
    @patch('MRTD.HardwareScanner',side_effect=MockResponse)
    def test_decode_firstmiddle_name(self,mock_obj):
        self.assertEqual(decode(),('4626', {'line1': {'issuing_country': 'CIV', 'last_name': 'LYNN', 'given_name': 'NEVEAH BRAM'}, 'line2': {'passport_number': 'W620126G5', 'country_code': 'CIV', 'bir

    #Mocking the dummydatabase() and givng two dictionary lines containing all the information fields of a person. Here again the given name of the person includes first and middle name in it.
    @patch('MRTD.dummydatabase',side_effect=MockResponse)
    def test_encode_firstmiddle_name(self,mock_obj):
        self.assertEqual(encode(),('4626', 'P<CIVLYNN<<NEVEAH<BRAM<<<<<<<<<<<<<<<<<<<<<;W620126G54CIV5910106F9707302AJ010215I<<<<<<6'),'The check digits should be calculated correctly')
```

```
   93:
--------------------------------------------------------------------------------
[0.01105 s] survived
   - [# 290] SIR MRTD:
--------------------------------------------------------------------------------
   88:
   89:     personal_number = ''.join(line2_list[28:41])
   90:     if personal_number.find('<'):
   91:         i = personal_number.find('<')
-  92:         personal_number = personal_number[:i]
+  92:         personal_number = personal_number[:]
   93:
   94:
   95:     fields_decode.append(str(line2_list[43]))
   96:
--------------------------------------------------------------------------------
[0.01207 s] killed by test_decode_firstmiddle_name (MTTDtest.TestMRTD)
[*] Mutation score [166.07243 s]: 89.2%
   - all: 290
   - killed: 232 (80.0%)
   - survived: 31 (10.7%)
   - incompetent: 2 (0.7%)
   - timeout: 25 (8.6%)
(base) nikhilparmar@Nikhils-MBP 567 %
```

1. The number of mutants killed: 232

2. The number of mutants that survived are: 31

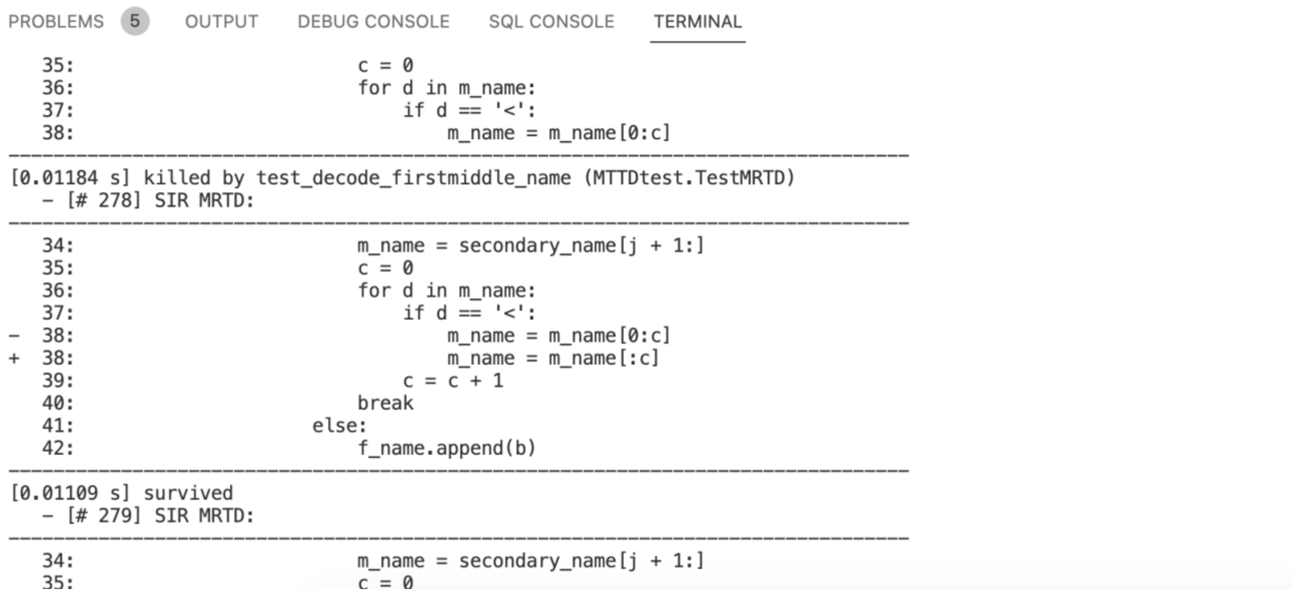
3. Based on the results from MutPy, I'm able to see where all the MutPy tool is going around in my functions and mutating (changing) parts of those functions. When those parts of the functions are changed and if my testcases don't notice it then that means that I need to create specific testcases that target those parts of the functions, so that when those parts are changed by MutPy, my new testcase will be able to detect the mutants and kill them.

- **Please create additional test cases to kill the mutants. And list the names of the additional test cases.**

ADDITIONAL TESTCASES:

The different places where I wrote the testcases to kill the mutants are as follows:

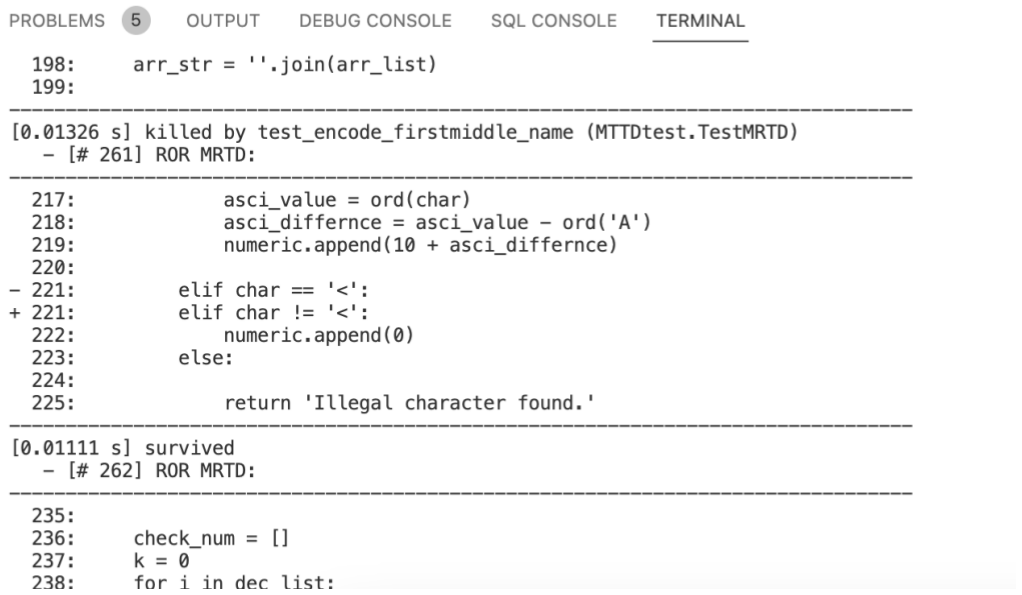
1. test_getCheckCode_arrow()



```
   35:                          c = 0
   36:                          for d in m_name:
   37:                              if d == '<':
   38:                                  m_name = m_name[0:c]
--------------------------------------------------------------------------------
[0.01184 s] killed by test_decode_firstmiddle_name (MTTDtest.TestMRTD)
   - [# 278] SIR MRTD:
--------------------------------------------------------------------------------
   34:                          m_name = secondary_name[j + 1:]
   35:                          c = 0
   36:                          for d in m_name:
   37:                              if d == '<':
-  38:                                  m_name = m_name[0:c]
+  38:                                  m_name = m_name[:c]
   39:                              c = c + 1
   40:                          break
   41:                      else:
   42:                          f_name.append(b)
--------------------------------------------------------------------------------
[0.01109 s] survived
   - [# 279] SIR MRTD:
--------------------------------------------------------------------------------
   34:                          m_name = secondary_name[j + 1:]
   35:                          c = 0
```

This mutant survived since I didn't have any testcase checking for '<' arrow input in the getCheckCode(). So, I wrote a testcase with name: "test_getCheckCode_arrow() " and gave "<" as an input to activate that part of the function and kill the mutant.

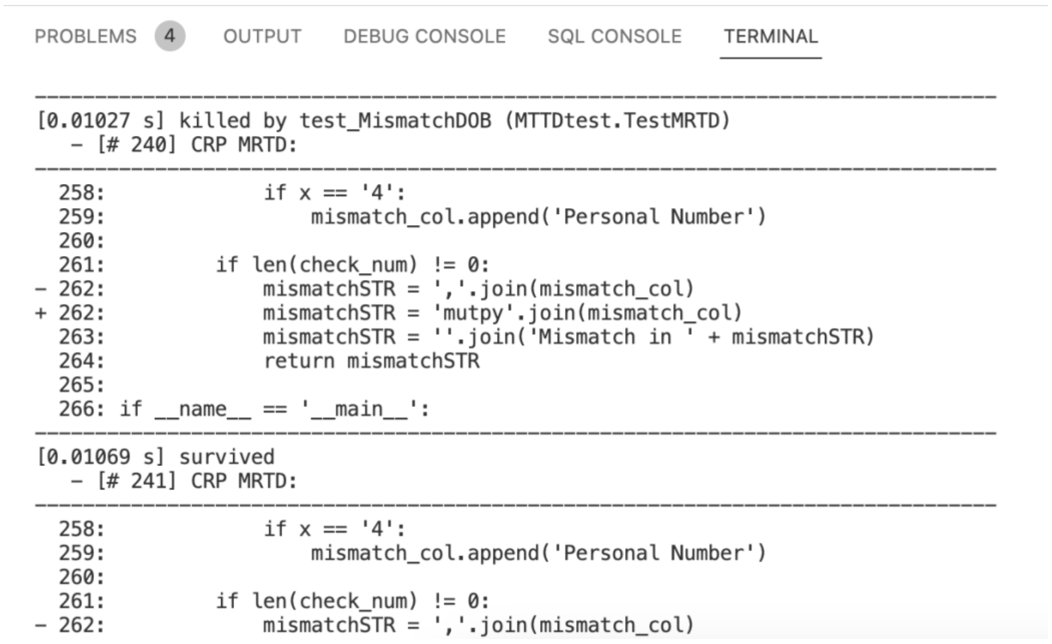
2. test_getCheckCode_illegalcharacter()



```
  198:     arr_str = ''.join(arr_list)
  199:
--------------------------------------------------------------------------------
[0.01326 s] killed by test_encode_firstmiddle_name (MTTDtest.TestMRTD)
   - [# 261] ROR MRTD:
--------------------------------------------------------------------------------
  217:             asci_value = ord(char)
  218:             asci_differnce = asci_value - ord('A')
  219:             numeric.append(10 + asci_differnce)
  220:
- 221:         elif char == '<':
+ 221:         elif char != '<':
  222:             numeric.append(0)
  223:         else:
  224:
  225:             return 'Illegal character found.'
--------------------------------------------------------------------------------
[0.01111 s] survived
   - [# 262] ROR MRTD:
--------------------------------------------------------------------------------
  235:
  236:     check_num = []
  237:     k = 0
  238:     for i in dec_list:
```

This mutant survived since I didn't have any testcase checking for 'Illegal character' return input in the getCheckCode(). So, I wrote a testcase with name: "test_getCheckCode_illegalcharacter() " and gave "$" as an input to activate that part of the function and kill the mutant.

3. test_MismatchPersNoDoe()



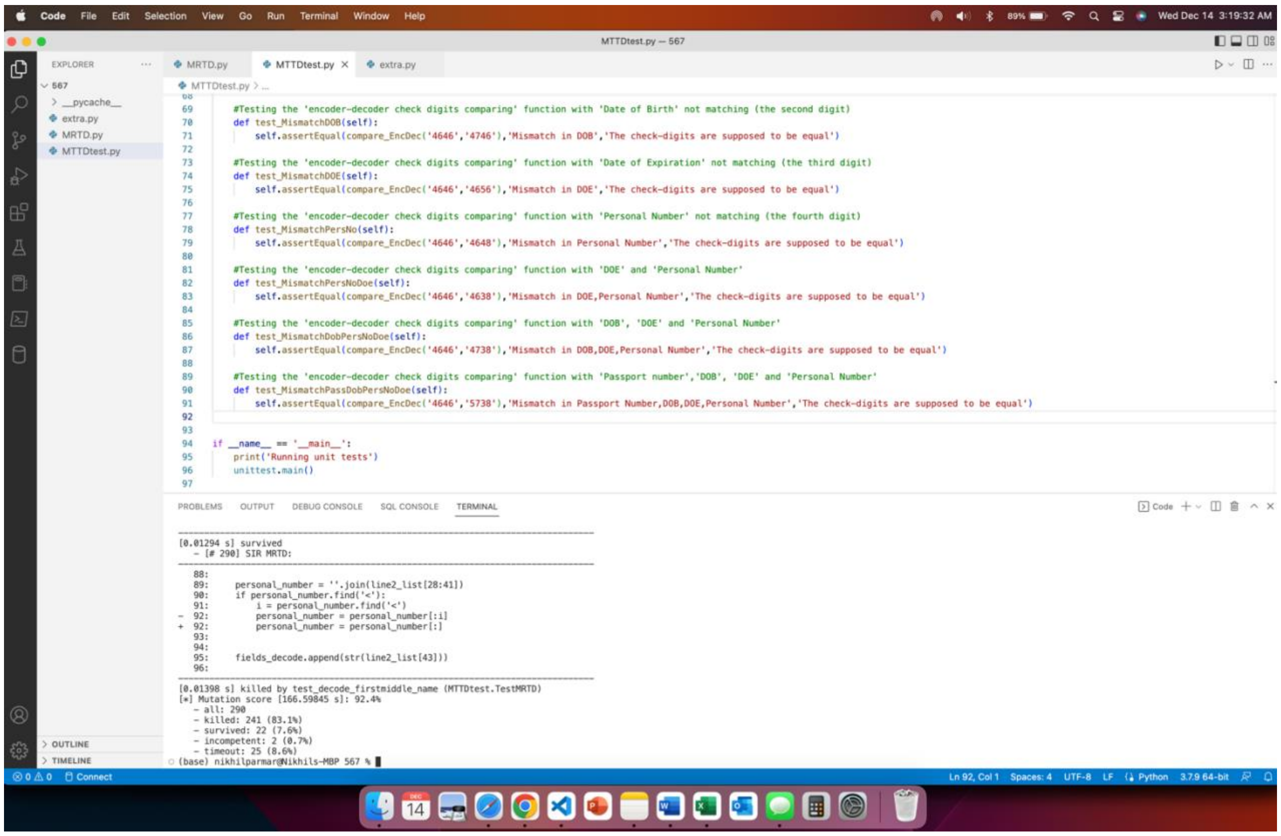
```
--------------------------------------------------------------------------------
[0.01027 s] killed by test_MismatchDOB (MTTDtest.TestMRTD)
   - [# 240] CRP MRTD:
--------------------------------------------------------------------------------
  258:             if x == '4':
  259:                 mismatch_col.append('Personal Number')
  260:
  261:         if len(check_num) != 0:
- 262:             mismatchSTR = ','.join(mismatch_col)
+ 262:             mismatchSTR = 'mutpy'.join(mismatch_col)
  263:             mismatchSTR = ''.join('Mismatch in ' + mismatchSTR)
  264:             return mismatchSTR
  265:
  266: if __name__ == '__main__':
--------------------------------------------------------------------------------
[0.01069 s] survived
   - [# 241] CRP MRTD:
--------------------------------------------------------------------------------
  258:             if x == '4':
  259:                 mismatch_col.append('Personal Number')
  260:
  261:         if len(check_num) != 0:
- 262:             mismatchSTR = ','.join(mismatch_col)
```

This mutant survived since I didn't have any testcase returning a multiple fields mismatch which would have involved a 'comma' to represent them. So, I wrote a testcase with name: "test_MismatchPersNoDoe()" which will involve a return with   mismatch in personal number, date of expiration.

Likewise, here are other additional testcases written to kill more mutants.

4. test_MismatchDobPersNoDoe()
5. test_MismatchPassDobPersNoDoe()

UPDATED MutPy Report:



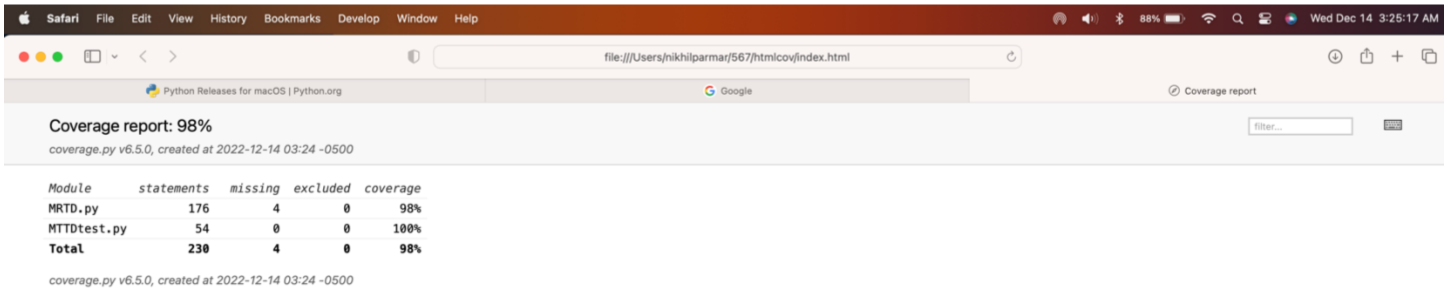
```python
    #Testing the 'encoder-decoder check digits comparing' function with 'Date of Birth' not matching (the second digit)
    def test_MismatchDOB(self):
        self.assertEqual(compare_EncDec('4646','4746'),'Mismatch in DOB','The check-digits are supposed to be equal')

    #Testing the 'encoder-decoder check digits comparing' function with 'Date of Expiration' not matching (the third digit)
    def test_MismatchDOE(self):
        self.assertEqual(compare_EncDec('4646','4656'),'Mismatch in DOE','The check-digits are supposed to be equal')

    #Testing the 'encoder-decoder check digits comparing' function with 'Personal Number' not matching (the fourth digit)
    def test_MismatchPersNo(self):
        self.assertEqual(compare_EncDec('4646','4648'),'Mismatch in Personal Number','The check-digits are supposed to be equal')

    #Testing the 'encoder-decoder check digits comparing' function with 'DOE' and 'Personal Number'
    def test_MismatchPersNoDoe(self):
        self.assertEqual(compare_EncDec('4646','4638'),'Mismatch in DOE,Personal Number','The check-digits are supposed to be equal')

    #Testing the 'encoder-decoder check digits comparing' function with 'DOB', 'DOE' and 'Personal Number'
    def test_MismatchDobPersNoDoe(self):
        self.assertEqual(compare_EncDec('4646','4738'),'Mismatch in DOB,DOE,Personal Number','The check-digits are supposed to be equal')

    #Testing the 'encoder-decoder check digits comparing' function with 'Passport number','DOB', 'DOE' and 'Personal Number'
    def test_MismatchPassDobPersNoDoe(self):
        self.assertEqual(compare_EncDec('4646','5738'),'Mismatch in Passport Number,DOB,DOE,Personal Number','The check-digits are supposed to be equal')


if __name__ == '__main__':
    print('Running unit tests')
    unittest.main()
```

```
--------------------------------------------------------------------------------
[0.01294 s] survived
   - [# 290] SIR MRTD:
--------------------------------------------------------------------------------
   88:
   89:     personal_number = ''.join(line2_list[28:41])
   90:     if personal_number.find('<'):
   91:         i = personal_number.find('<')
-  92:         personal_number = personal_number[:i]
+  92:         personal_number = personal_number[:]
   93:
   94:
   95:     fields_decode.append(str(line2_list[43]))
   96:
--------------------------------------------------------------------------------
[0.01398 s] killed by test_decode_firstmiddle_name (MTTDtest.TestMRTD)
[*] Mutation score [166.59845 s]: 92.4%
   - all: 290
   - killed: 241 (83.1%)
   - survived: 22 (7.6%)
   - incompetent: 2 (0.7%)
   - timeout: 25 (8.6%)
(base) nikhilparmar@Nikhils-MBP 567 %
```

After killing the mutants, the mutation score went up from **89.2%** to **92.4%.**

The number of mutants killed went up from **232** to **241.**

UPDATED COVERAGE REPORT:



Coverage report: 98%

coverage.py v6.5.0, created at 2022-12-14 03:24 -0500

Module	statements	missing	excluded	coverage
MRTD.py	176	4	0	98%
MTTDtest.py	54	0	0	100%
Total	230	4	0	98%

coverage.py v6.5.0, created at 2022-12-14 03:24 -0500