Project: Unit Tests

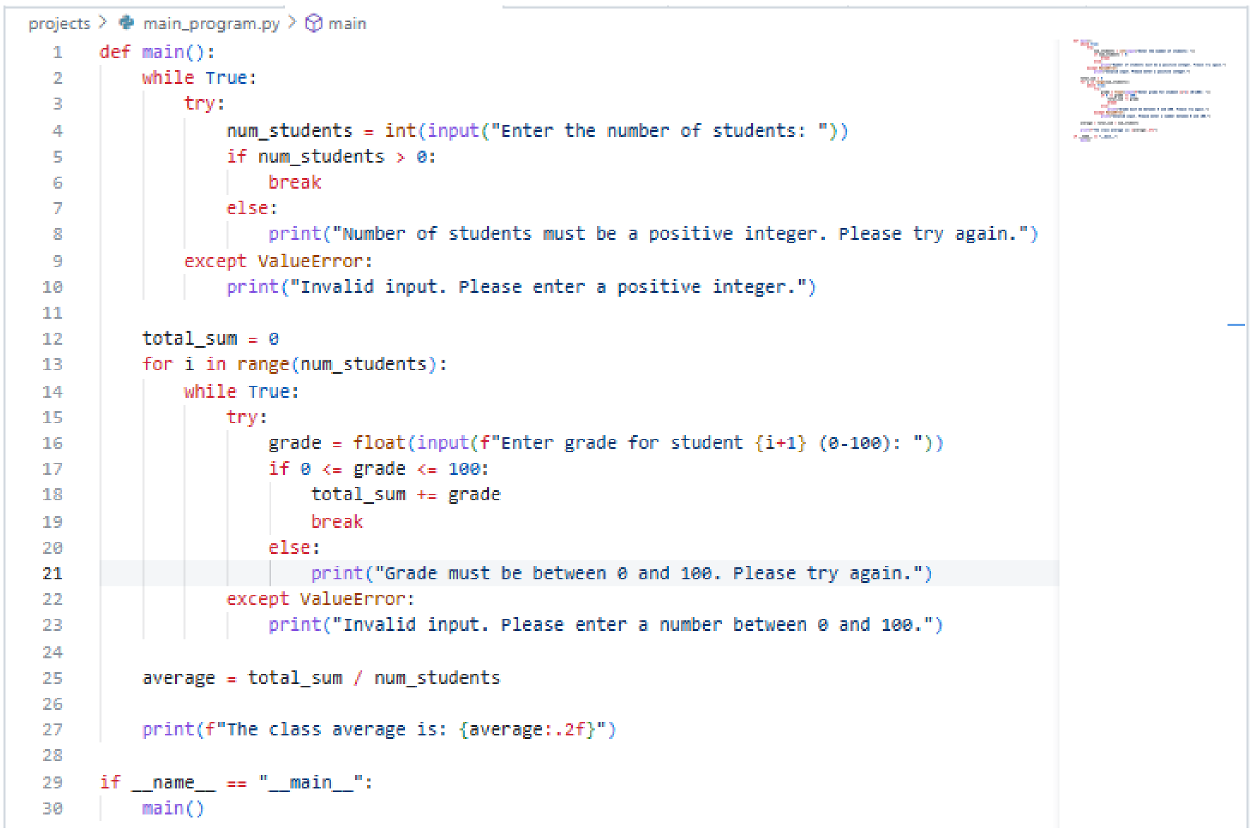
Name: Jhonatan Parada

Course: ET574

Objective: Add three different test cases to the module *test_main_program* to extra test how *main_program.py* handles different types of inputs for the program.

Source Files: main_program.py, test_main_program.py

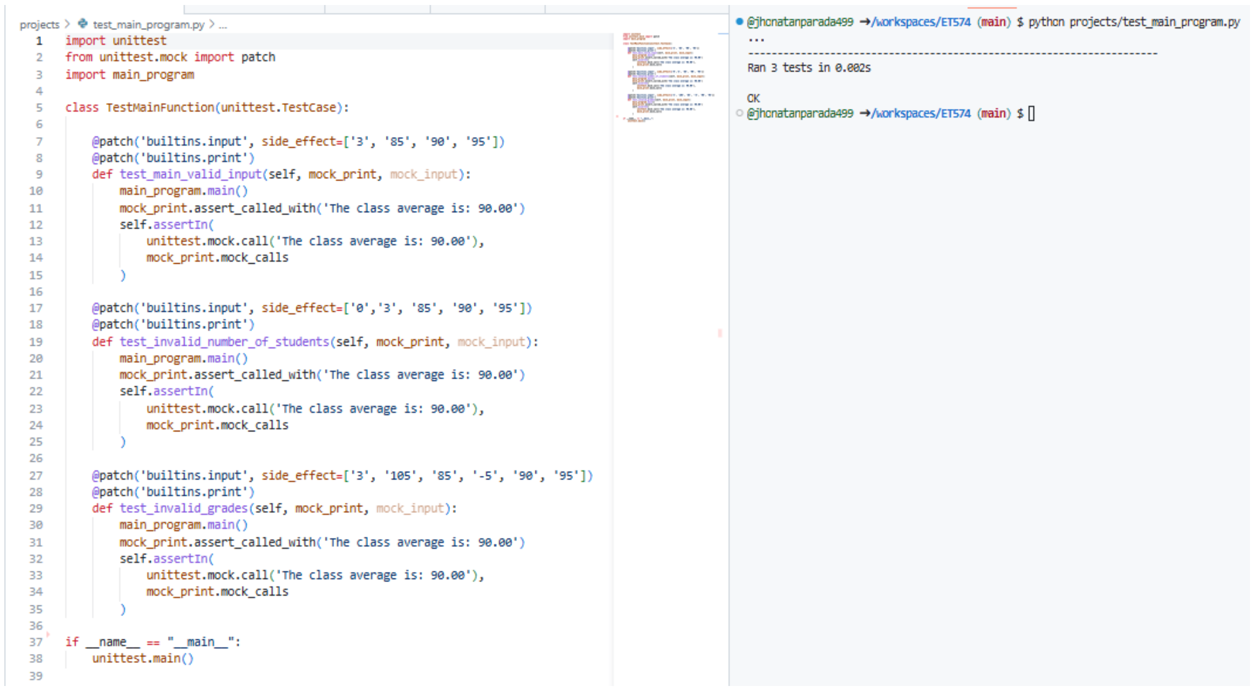
main_program.py:



projects > main_program.py > main

```python
def main():
    while True:
        try:
            num_students = int(input("Enter the number of students: "))
            if num_students > 0:
                break
            else:
                print("Number of students must be a positive integer. Please try again.")
        except ValueError:
            print("Invalid input. Please enter a positive integer.")

    total_sum = 0
    for i in range(num_students):
        while True:
            try:
                grade = float(input(f"Enter grade for student {i+1} (0-100): "))
                if 0 <= grade <= 100:
                    total_sum += grade
                    break
                else:
                    print("Grade must be between 0 and 100. Please try again.")
            except ValueError:
                print("Invalid input. Please enter a number between 0 and 100.")

    average = total_sum / num_students

    print(f"The class average is: {average:.2f}")

if __name__ == "__main__":
    main()
```

test_main_program.py:



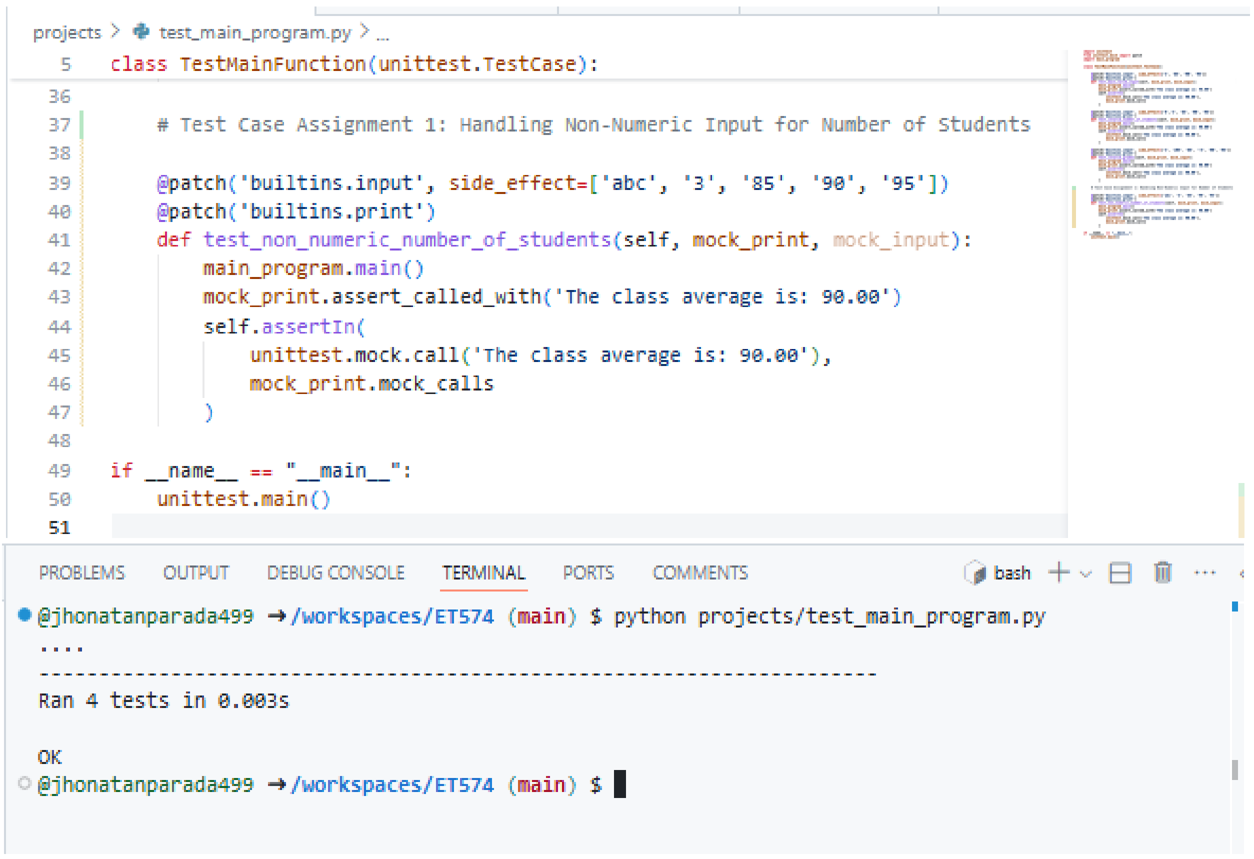
projects > test_main_program.py > ...

```python
import unittest
from unittest.mock import patch
import main_program

class TestMainFunction(unittest.TestCase):

    @patch('builtins.input', side_effect=['3', '85', '90', '95'])
    @patch('builtins.print')
    def test_main_valid_input(self, mock_print, mock_input):
        main_program.main()
        mock_print.assert_called_with('The class average is: 90.00')
        self.assertIn(
            unittest.mock.call('The class average is: 90.00'),
            mock_print.mock_calls
        )

    @patch('builtins.input', side_effect=['0','3', '85', '90', '95'])
    @patch('builtins.print')
    def test_invalid_number_of_students(self, mock_print, mock_input):
        main_program.main()
        mock_print.assert_called_with('The class average is: 90.00')
        self.assertIn(
            unittest.mock.call('The class average is: 90.00'),
            mock_print.mock_calls
        )

    @patch('builtins.input', side_effect=['3', '105', '85', '-5', '90', '95'])
    @patch('builtins.print')
    def test_invalid_grades(self, mock_print, mock_input):
        main_program.main()
        mock_print.assert_called_with('The class average is: 90.00')
        self.assertIn(
            unittest.mock.call('The class average is: 90.00'),
            mock_print.mock_calls
        )

if __name__ == "__main__":
    unittest.main()

```

```
● @jhonatanparada499 →/workspaces/ET574 (main) $ python projects/test_main_program.py
  ...
  ----------------------------------------------------------------------
  Ran 3 tests in 0.002s

  OK
○ @jhonatanparada499 →/workspaces/ET574 (main) $ ▯
```

Step 1:

Description: We want to test how main_program.py will handle ‘abc’ (non-numeric) as input for number of students. If the main_program’s code is solid, it will reject that input and request again a valid one. If that happens, the next mocked input value in ‘side_effects’ which is 3, would be then passed as number of students and finally taking the last 3 valid grades for the 3 students.



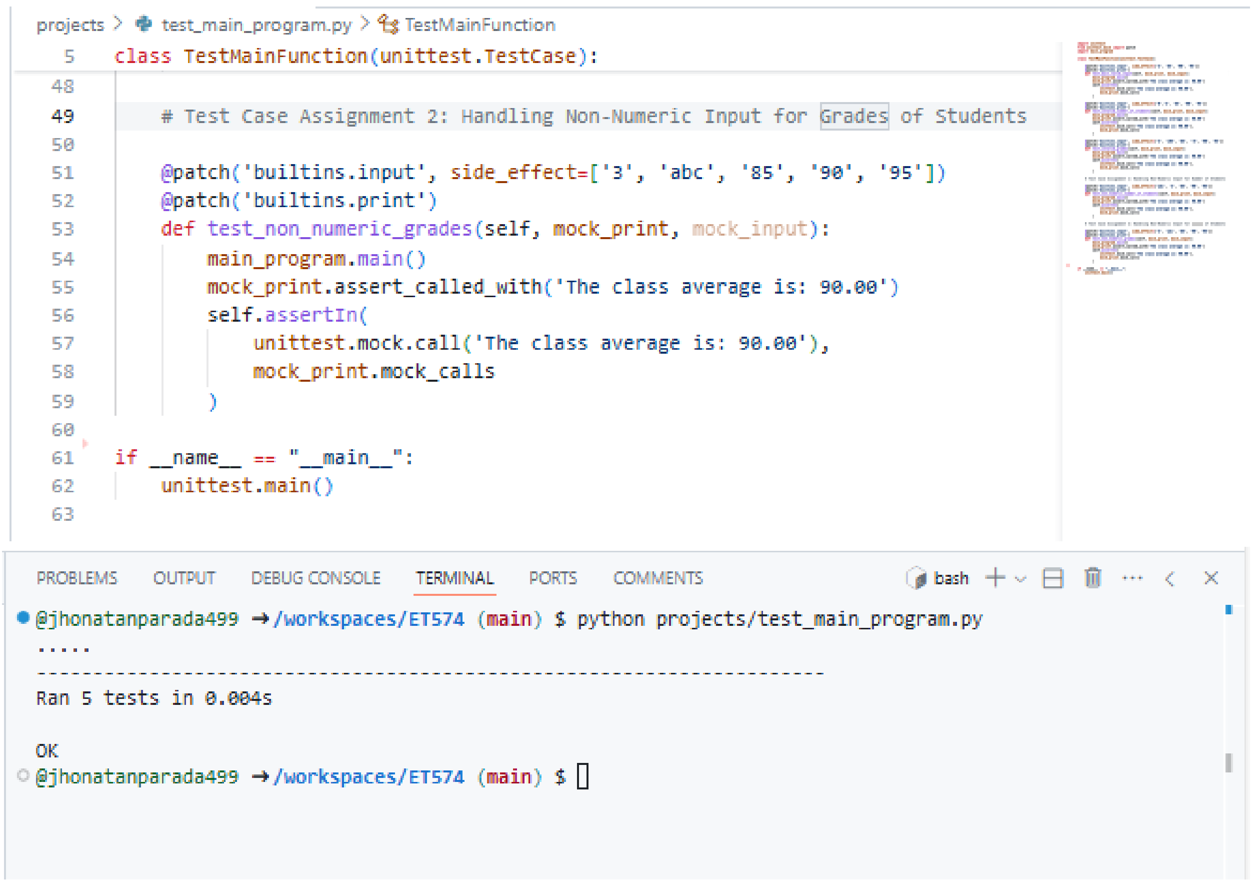
projects > test_main_program.py > ...

```python
class TestMainFunction(unittest.TestCase):

    # Test Case Assignment 1: Handling Non-Numeric Input for Number of Students

    @patch('builtins.input', side_effect=['abc', '3', '85', '90', '95'])
    @patch('builtins.print')
    def test_non_numeric_number_of_students(self, mock_print, mock_input):
        main_program.main()
        mock_print.assert_called_with('The class average is: 90.00')
        self.assertIn(
            unittest.mock.call('The class average is: 90.00'),
            mock_print.mock_calls
        )

if __name__ == "__main__":
    unittest.main()
```

```
@jhonatanparada499 ➜ /workspaces/ET574 (main) $ python projects/test_main_program.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.003s

OK
@jhonatanparada499 ➜ /workspaces/ET574 (main) $
```

Step 2:

Description: To test non-numeric grades as input we would create another block of functions, changing the name of the function as well as the parameter side_effect to simulate this new scenario. The number of students will be 3, and then, as the first grade we will pass it ‘abc’. Logically, the main program rejects this input and then uses the following valid grades.

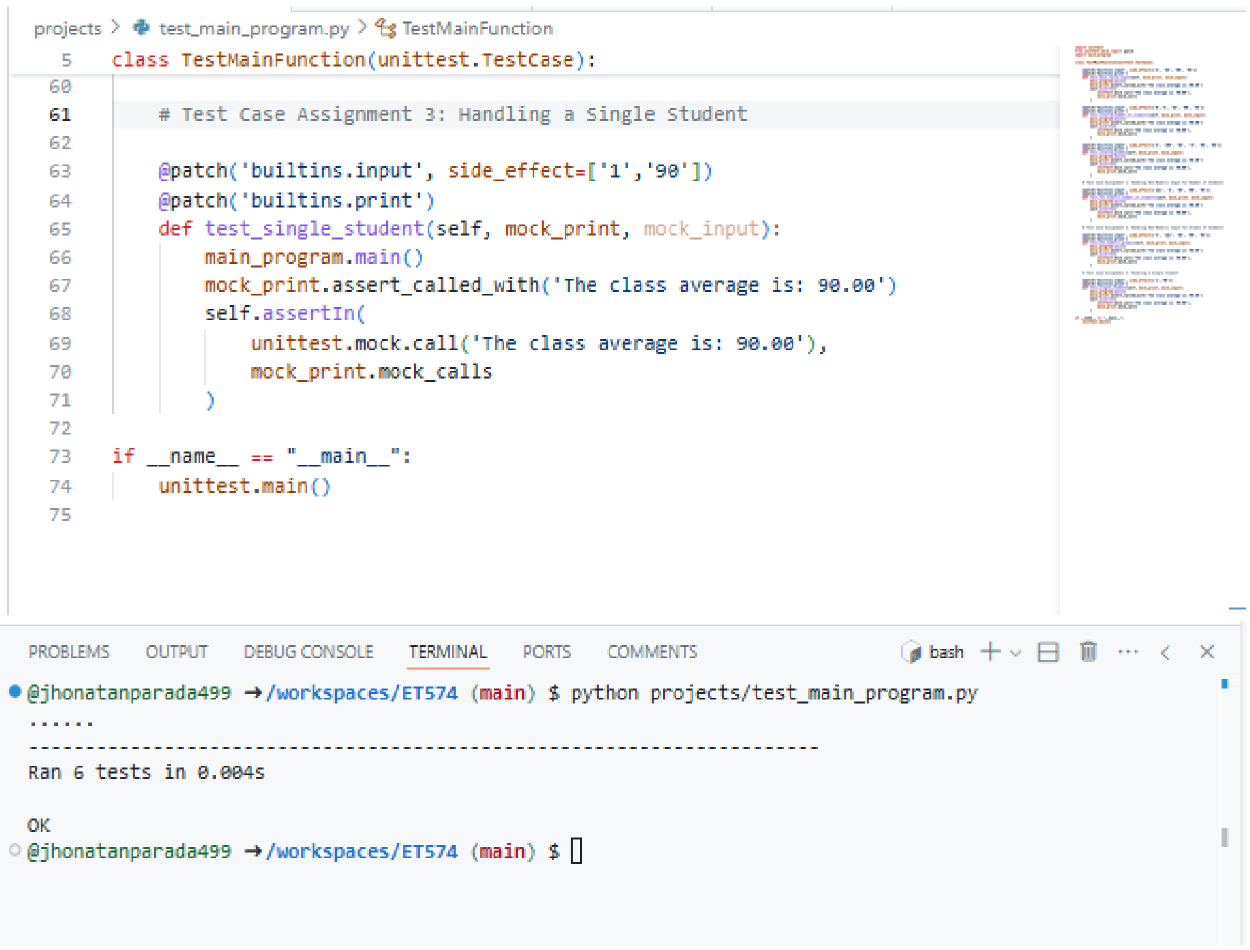

projects > test_main_program.py > TestMainFunction
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
bash

```python
class TestMainFunction(unittest.TestCase):

    # Test Case Assignment 2: Handling Non-Numeric Input for Grades of Students

    @patch('builtins.input', side_effect=['3', 'abc', '85', '90', '95'])
    @patch('builtins.print')
    def test_non_numeric_grades(self, mock_print, mock_input):
        main_program.main()
        mock_print.assert_called_with('The class average is: 90.00')
        self.assertIn(
            unittest.mock.call('The class average is: 90.00'),
            mock_print.mock_calls
        )

if __name__ == "__main__":
    unittest.main()
```

```
@jhonatanparada499 ➜/workspaces/ET574 (main) $ python projects/test_main_program.py
.....
----------------------------------------------------------------------
Ran 5 tests in 0.004s

OK
@jhonatanparada499 ➜/workspaces/ET574 (main) $ 
```

Step 3:

Description: Finally, we add the last test case to see how the main program will handle the event where the number of students is one, and to do so, we create a similar block of code like the previous ones but with different defined function name and different side_effect value. The average of 90 is 90, therefore, since no errors are displayed, we are getting the expected input, and the 6 tests are running correctly and as expected.


projects > test_main_program.py > TestMainFunction

```python
class TestMainFunction(unittest.TestCase):

    # Test Case Assignment 3: Handling a Single Student

    @patch('builtins.input', side_effect=['1','90'])
    @patch('builtins.print')
    def test_single_student(self, mock_print, mock_input):
        main_program.main()
        mock_print.assert_called_with('The class average is: 90.00')
        self.assertIn(
            unittest.mock.call('The class average is: 90.00'),
            mock_print.mock_calls
        )

if __name__ == "__main__":
    unittest.main()
```

```
@jhonatanparada499 ➜ /workspaces/ET574 (main) $ python projects/test_main_program.py
......
----------------------------------------------------------------------
Ran 6 tests in 0.004s

OK
@jhonatanparada499 ➜ /workspaces/ET574 (main) $ 
```