

2303A51924

Assignment-8.4

Task 1: Developing a Utility Function Using TDD

Scenario

You are working on a small utility library for a larger software system. One of the required functions should calculate the square of a given number, and correctness is critical because other modules depend on it.

Task Description

Following the Test Driven Development (TDD) approach:

1. First, write unit test cases to verify that a function correctly returns the square of a number for multiple inputs.
2. After defining the test cases, use GitHub Copilot or Cursor AI to generate the function implementation so that all tests pass.

Ensure that the function is written only after the tests are created.

Expected Outcome

- A separate test file and implementation file
- Clearly written test cases executed before implementation
- AI-assisted function implementation that passes all tests
- Demonstration of the TDD cycle: test → fail → implement → pass

```
1 import unittest
2 def square(n):
3     """Calculates the square of a number."""
4     return n * n
5
6
7 class TestSquare(unittest.TestCase):
8
9     def test_positive(self):
10         self.assertEqual(square(10), 100)
11
12     def test_zero(self):
13         self.assertEqual(square(0), 0)
14
15     def test_negative(self):
16         # Result should be positive
17         self.assertEqual(square(-6), 36)
18
19     def test_float(self):
20         self.assertEqual(square(2.5), 6.25)
21
22
23 if __name__ == '__main__':
24     unittest.main()
```

Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher '54504' ...

Ran 4 tests in 0.001s

OK

PS C:\Users\Chinnari>

```
7 class TestSquare(unittest.TestCase):
8     self.assertEqual(square(10), 100)
9
10     def test_zero(self):
11         self.assertEqual(square(0), 0)
12
13     def test_negative(self):
14         # Result should be positive
15         self.assertEqual(square(-6), 36)
16
17     def test_float(self):
18         self.assertEqual(square(2.5), 6.25)
19
20
21 if __name__ == '__main__':
22     # Run the tests
23     unittest.main()
```

Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher '54504' ...

Ran 4 tests in 0.001s

OK

PS C:\Users\Chinnari>

Task 2: Email Validation for a User Registration System

Scenario

You are developing the backend of a user registration system. One requirement is to validate user email addresses before storing them in the database.

Task Description

Apply Test Driven Development by:

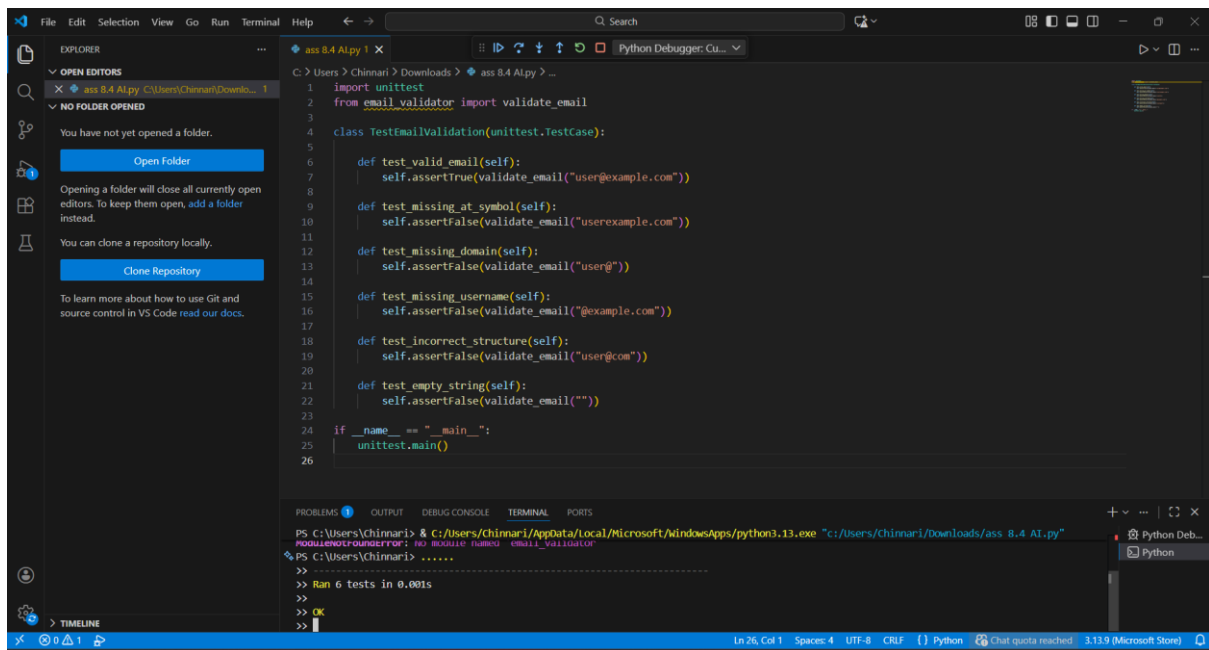
1. Writing unit test cases that define valid and invalid email formats (e.g., missing @, missing domain, incorrect structure).

2. Using AI assistance to implement the `validate_email()` function based strictly on the behavior described by the test cases.

The implementation should be driven entirely by the test expectations.

Expected Outcome

- Well-defined unit tests using `unittest` or `pytest`
- An AI-generated email validation function
- All test cases passing successfully
- Clear alignment between test cases and function behavior



```
1 import unittest
2 from email_validator import validate_email
3
4 class TestEmailValidation(unittest.TestCase):
5
6     def test_valid_email(self):
7         self.assertTrue(validate_email("user@example.com"))
8
9     def test_missing_at_symbol(self):
10        self.assertFalse(validate_email("userexample.com"))
11
12    def test_missing_domain(self):
13        self.assertFalse(validate_email("user@"))
14
15    def test_missing_username(self):
16        self.assertFalse(validate_email("@example.com"))
17
18    def test_incorrect_structure(self):
19        self.assertFalse(validate_email("user@com"))
20
21    def test_empty_string(self):
22        self.assertFalse(validate_email(""))
23
24    if __name__ == "__main__":
25        unittest.main()
26
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Chinnari> & C:/Users/Chinnari/AppData/Local/Microsoft/WindowsApps/python3.13.exe "C:/Users/Chinnari/Downloads/ass 8.4 AT.py"
ModuleNotFoundError: No module named 'email_validator'
PS C:\Users\Chinnari> .....
>> Ran 6 tests in 0.001s
>> OK
```

Task 3: Decision Logic Development Using TDD

Scenario

In a grading or evaluation module, a function is required to determine the maximum value among three inputs. Accuracy is essential, as incorrect results could affect downstream decision logic.

Task Description

Using the TDD methodology:

1. Write test cases that describe the expected output for different combinations of three numbers.
2. Prompt GitHub Copilot or Cursor AI to implement the function logic based on the written tests.

Avoid writing any logic before test cases are completed.

Expected Outcome

- Comprehensive test cases covering normal and edge cases
- AI-generated function implementation
- Passing test results demonstrating correctness
- Evidence that logic was derived from tests, not assumptions

```
1 import unittest
2
3 def find_max_of_three(a, b, c):
4     if a >= b and a >= c:
5         return a
6     elif b >= a and b >= c:
7         return b
8     else:
9         return c
10
11 class TestMaxLogic(unittest.TestCase):
12
13     def test_different_positions(self):
14         """Tests that the function finds the max regardless of its position."""
15         self.assertEqual(find_max_of_three(10, 5, 2), 10) # Max is first
16         self.assertEqual(find_max_of_three(3, 15, 7), 15) # Max is middle
17         self.assertEqual(find_max_of_three(1, 4, 9), 9) # Max is last
18
19     def test_negative_numbers(self):
20         """Tests that the logic holds for negative values."""
21         self.assertEqual(find_max_of_three(-1, -10, -5), -1)
22
23     def test_all_equal(self):
24         """Tests the edge case where all inputs are identical."""
25         self.assertEqual(find_max_of_three(7, 7, 7), 7)
26
27     def test_two_equal(self):
28         """Tests when two numbers are the same and represent the maximum."""
29         self.assertEqual(find_max_of_three(12, 12, 4), 12)
30         self.assertEqual(find_max_of_three(5, 20, 20), 20)
31
32 if __name__ == '__main__':
33     unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Chinnari\Downloads> python -m unittest
Ran 4 tests in 0.002s

Task 4: Shopping Cart Development with AI-Assisted TDD

Scenario

You are building a simple shopping cart module for an e-commerce application. The cart must support adding items, removing items, and calculating the total price accurately.

Task Description

Follow a test-driven approach:

1. Write unit tests for each required behavior:

- o Adding an item
- o Removing an item
- o Calculating the total price

2. After defining all tests, use AI tools to generate the ShoppingCart class and its methods so that the tests pass.

Focus on behavior-driven testing rather than implementation details.

Expected Outcome

- The image shows a Visual Studio Code (VS Code) editor window with a Python file named `ass_8_4_AI.py` open. The file contains a Python script for email validation using regular expressions and the `unittest` module.

```
1 import re
2
3 def validate_email(email):
4     # \.[a-zA-Z]{2,5}$ : Dot followed by 2+ char extension (e.g., .com)
5     pattern = r'^[a-zA-Z0-9_-]+@[a-zA-Z0-9-]+\.[a-zA-Z]{2,5}$'
6
7     if re.match(pattern, email):
8         return True
9     return False
10
11 class TestUserRegistrationEmail(unittest.TestCase):
12
13     def test_valid_email(self):
14         """Valid format should pass"""
15         self.assertTrue(validate_email("new_user@registration.com"))
16
17     def test_missing_at_symbol(self):
18         """Invalid: No @ symbol"""
19         self.assertFalse(validate_email("userdomain.com"))
20
21     def test_missing_domain(self):
22         """Invalid: Nothing after @"""
23         self.assertFalse(validate_email("user@"))
24
25     def test_missing_username(self):
26         """Invalid: Nothing before @"""
27         self.assertFalse(validate_email("@registration.com"))
28
29     def test_incorrect_structure(self):
30         """Invalid: Multiple @ symbols or missing dot"""
31         self.assertFalse(validate_email("user@domain.com"))
32         self.assertFalse(validate_email("user@domaincom"))
33
34     if __name__ == '__main__':
35         # exit=False allows the script to finish and show output in VS Code terminal
36         unittest.main(argv=['first-arg-is-ignored'], exit=False)
37
38 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

The terminal at the bottom shows the command to run the script and its output:

```
PS C:\Users\Chinnari\Downloads> cd 'c:\Users\Chinnari\Downloads'; & 'c:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\vscode\extensions\python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61176' '-x' 'c:\Users\Chinnari\Downloads\ass_8_4_AI.py'
-----
Ran 4 tests in 0.002s
OK
PS C:\Users\Chinnari\Downloads>
```

- Clearly written test cases defining expected behavior

- AI-assisted implementation of the palindrome checker
- All test cases passing successfully
- Evidence of TDD methodology applied correctly

```

1 import unittest
2
3 class TestPalindromeChecker(unittest.TestCase):
4
5     def test_simple_palindrome(self):
6         self.assertTrue(is_palindrome("madam"))
7
8     def test_non_palindrome(self):
9         self.assertFalse(is_palindrome("hello"))
10
11     def test_case_variation_palindrome(self):
12         self.assertTrue(is_palindrome("Madam"))
13
14     def test_single_character(self):
15         self.assertTrue(is_palindrome("a"))
16
17     def test_empty_string(self):
18         self.assertTrue(is_palindrome(""))
19
20 def is_palindrome(text):
21     normalized_text = text.lower()
22     return normalized_text == normalized_text[::-1]
23
24 if __name__ == "__main__":
25     unittest.main()
26

```

```

PS C:\Users\Chinnari\Downloads> cd 'c:\Users\Chinnari\Downloads'; & 'c:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '61176' '-c' 'C:\Users\Chinnari\Downloads\ass_8.4_AI.py'
OK
PS C:\Users\Chinnari\Downloads> cd 'c:\Users\Chinnari'; & 'c:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Chinnari\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '58112' '-c' 'C:\Users\Chinnari\py'

```