

**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

The screenshot shows two instances of the VS Code interface, both displaying the same Python code in a file named `test (1).csv`. The code defines a `square` function and a `TestSquare` class with five test methods: `test_positive`, `test_zero`, `test_negative`, `test_float`, and `main`.

```

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

```

The left instance shows the `CALL STACK` tab selected, indicating no module named `'math.util'`. The right instance shows the `TERMINAL` tab selected, displaying the output of the test run:

```

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

A screenshot of Microsoft Visual Studio Code. The left sidebar shows the Explorer, Git, and Timeline panes. The main editor pane contains Python code for testing email validation. The terminal pane at the bottom shows the output of running the test script, indicating 6 tests ran in 0.001 seconds with a result of OK.

```
C:\> Users > Chinnari > Downloads > ass 8.4 Alpy > ...
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 Alpy.py"  
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

```
File Edit Selection View Go Run Terminal Help < > Search
EXPLORER OPEN EDITORS ass_8_4 Al.py C:\Users\Chinnari\Downloads NO FOLDER OPENED
Import unittest
1 def find_max_of_three(a, b, c):
2     if a >= b and a >= c:
3         return a
4     elif b >= a and b >= c:
5         return b
6     else:
7         return c
8
9 class TestMaximumLogic(unittest.TestCase):
10
11     def test_different_positions(self):
12         """Tests that the function finds the max regardless of its position."""
13         self.assertEqual(find_max_of_three(10, 5, 2), 10) # Max is first
14         self.assertEqual(find_max_of_three(3, 15, 7), 15) # Max is middle
15         self.assertEqual(find_max_of_three(1, 4, 9), 9) # Max is last
16
17     def test_negative_numbers(self):
18         """Tests that the logic holds for negative values."""
19         self.assertEqual(find_max_of_three(-1, -10, -5), -1)
20
21     def test_all_equal(self):
22         """Tests the edge case where all inputs are identical."""
23         self.assertEqual(find_max_of_three(7, 7, 7), 7)
24
25     def test_two_equal(self):
26         """Tests when two numbers are the same and represent the maximum."""
27         self.assertEqual(find_max_of_three(12, 12, 4), 12)
28         self.assertEqual(find_max_of_three(5, 20, 20), 20)
29
30
31 if __name__ == '__main__':
32     unittest.main(argv=['first-arg-is-ignored'], exit=False)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Chinnari\Downloads> & "c:\Users\Chinnari\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "c:\Users\Chinnari\.vscodeextensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher" "50865" -- "C:\Users\Chinnari\Downloads\ass_8_4 Al.py"
ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher" "50865" -- "C:\Users\Chinnari\Downloads\ass_8_4 Al.py"
Ran 4 tests in 0.000s
OK
PS C:\Users\Chinnari\Downloads>
```

## 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

- Unit tests defining expected shopping cart behavior
- AI-generated class implementation
- All tests passing successfully
- Clear demonstration of TDD applied to a class-based design

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows a folder named "ass 8.4 Alpy" containing files "ass 8.4 Alpy.py" and "test\_ass\_8\_4\_Alpy.py".
- Code Editor:** Displays Python code for a string validation module and its corresponding unit tests.
- Terminal:** Shows the command "python -m unittest test\_ass\_8\_4\_Alpy.py" being run, resulting in "Ran 4 tests in 0.002s" and "OK".
- Status Bar:** Shows the path "C:\Users\Chinnari\Downloads", file "ass 8.4 Alpy.py", and the status "Python Debugging 2025.8.0-win32-x64/handled llns/debugpy\_launcher '61176' ... C:\Users\Chinnari\Downloads\ass 8.4 Alpy.py".

```

# ass 8.4 Alpy.py
import re

def validate_email(email):
    """Validate email address format"""
    pattern = r'^[a-zA-Z0-9.-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if re.match(pattern, email):
        return True
    return False

class TestUserRegistrationEmail(unittest.TestCase):

    def test_valid_email(self):
        """Valid format should pass"""
        self.assertTrue(validate_email("new_user@registration.com"))

    def test_missing_at_symbol(self):
        """Invalid: No @ symbol"""
        self.assertFalse(validate_email("userdomain.com"))

    def test_missing_domain(self):
        """Invalid: Nothing after @"""
        self.assertFalse(validate_email("user@"))

    def test_missing_username(self):
        """Invalid: Nothing before @"""
        self.assertFalse(validate_email("@registration.com"))

    def test_incorrect_structure(self):
        """Invalid: Multiple @ symbols or missing dot"""
        self.assertFalse(validate_email("user@domain.com"))
        self.assertFalse(validate_email("user@domaincom"))

if __name__ == '__main__':
    # '-exit=false' allows the script to finish and show output in VS Code terminal
    unittest.main(argv=['first-arg-is-ignored'], exit=False)

```

## Task 5: String Validation Module Using TDD

### Scenario

You are working on a text-processing module where a function is required to identify whether a given string is a palindrome. The function must handle different cases and inputs reliably.

### Task Description

#### Using Test Driven Development:

1. Write test cases for a palindrome checker covering:
  - o Simple palindromes
  - o Non-palindromes
  - o Case variations
2. Use GitHub Copilot or Cursor AI to generate the `is_palindrome()` function based on the test case expectations.

The function should be implemented only after tests are written.

### Expected Outcome

- 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

The screenshot shows the Visual Studio Code interface with the following details:

- File Structure:** The project tree shows a folder named "ass 8.4 AI" containing a file "is\_palindrome.py".
- Code Editor:** The main editor window displays a Python script named "is\_palindrome.py". The code defines a class `TestPalindromeChecker` that inherits from `unittest.TestCase`. It contains several test methods: `test\_simple\_palindrome`, `test\_non\_palindrome`, `test\_case\_variation\_palindrome`, `test\_single\_character`, and `test\_empty\_string`. Each test method uses `self.assertTrue` or `self.assertFalse` to check if the `is\_palindrome` function returns the expected result for a given input string.
- Terminal:** The bottom terminal tab shows the command-line output of the tests. It starts with "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.python\2025.18.0-win32-x64\bundled\libs\debug\launcher' '61176' '--' 'C:\Users\Chinnari\Downloads\ass 8.4 AI.py'" followed by "OK".
- Bottom Status Bar:** The status bar indicates the current file is "is\_palindrome.py" at line 22, column 52, with 3139 characters in total.