

Lab Assignment # 08

Course Title : AI Assistant Coding
Name of Student : B.Sai Varsha
Enrollment No. : 2303A54065
Batch No. : 48

Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases

Task Description #1 (Password Strength Validator – Apply AI in Security Context)

- Task: Apply AI to generate at least 3 assert test cases for `is_strong_password(password)` and implement the validator function.

• Requirements:

- o Password must have at least 8 characters.
- o Must include uppercase, lowercase, digit, and special character.
- o Must not contain spaces.

Example Assert Test Cases:

```
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == True
```

Expected Output #1: • Password validation logic passing all AI-generated test cases.

Output Screenshot:

```

task1.py task2.py task3.py task4.py task5.py
LAB8.1.PY > task1.py > ...
1 # Task 1: Password Strength Validator -| TDD Style
2 # Step 1: Test Cases (TDD Style)
3
4 import re
5
6 def is_strong_password(password):
7     if len(password) < 8:
8         return False
9     if " " in password:
10        return False
11     if not re.search(r"[A-Z]", password):
12         return False
13     if not re.search(r"[a-z]", password):
14         return False
15     if not re.search(r"[0-9]", password):
16         return False
17     if not re.search(r"[@#$%^&(*),.?\" :{}|<>]", password):
18         return False
19     return True
20
21
22 # Step 2: Test Cases (TDD Style)
23
24 def test_is_strong_password():
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Password validation logic passing all AI-generated test cases.
PS C:\Users\saiva\OneDrive\Desktop\AI assisted coding> python -m doctest -v LAB8.1.PY/task1.py
Password validation logic passing all AI-generated test cases.
3 items had no tests:
    task1
    task1.is_strong_password
    task1.test_is_strong_password
0 tests in 3 items.
0 passed.
Test passed.

```

Explanation: This checks if a password is truly “strong” by making sure it has the right mix of characters and no spaces.

Task Description #2 (Number Classification with Loops – Apply AI for Edge Case Handling)

- **Task:** Use AI to generate at least 3 assert test cases for a `classify_number(n)` function. Implement using loops.

- **Requirements:**

- Classify numbers as Positive, Negative, or Zero.
- Handle invalid inputs like strings and None.
- Include boundary conditions (-1, 0, 1).

Example Assert Test Cases:

```

assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"

```

Expected Output #2: • Classification logic passing all assert tests.

Output Screenshot:

```

task1.py  task2.py  X  task3.py  task4.py  task5.py
LAB8.1.PY > task2.py > ...
1 # Task 2: Number Classification
2 # Step 2: Implementation (Using Loop)
3
4 def classify_number(n):
5     if not isinstance(n, (int, float)):
6         return "Invalid Input"
7
8     if n > 0:
9         return "Positive"
10    elif n < 0:
11        return "Negative"
12    else:
13        return "Zero"
14
15
16 # Step 1: Test Cases (TDD Style)
17
18 def test_classify_number():
19     # Positive number
20     assert classify_number(10) == "Positive"
21
22     # Negative number
23     assert classify_number(-5) == "Negative"
24

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

classification logic passing all AI-generated test cases.
PS C:\Users\saiva\OneDrive\Desktop\AI assisted coding> python -m doctest -v LAB8.1.PY/task2.py
Classification logic passing all AI-generated test cases.
3 items had no tests:
    task2
    task2.classify_number
    task2.test_classify_number
0 tests in 3 items.
0 passed.
Test passed.

```

Explanation: This sorts numbers into positive, negative, or zero, while politely rejecting anything that isn't a number.

Task Description #3 (Anagram Checker – Apply AI for String Analysis)

- **Task:** Use AI to generate at least 3 assert test cases form `is_anagram(str1, str2)` and implement the function.

- **Requirements:**

- Ignore case, spaces, and punctuation.
- Handle edge cases (empty strings, identical words).

Example Assert Test Cases:

```

assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True

```

Expected Output #3: • Function correctly identifying anagrams and passing all AI-generated tests.

Output Screenshot:

The screenshot shows a code editor with multiple tabs: task1.py, task2.py, task3.py, task4.py (active), and task5.py. The code in task4.py is a TDD-style test for an Inventory class, covering addition, removal, and checking stock levels. Below the code editor is a terminal window showing the command `python -m doctest -v LAB8.1.PY/task4.py` running, which outputs that the inventory system is passing all AI-generated test cases. It lists 7 items with no tests, and 0 tests in 7 items, with 0 passed.

```
LAB8.1.PY > task4.py > ...
1 # Task 4: Inventory Class
2 # Step 1: Test Cases (TDD Style)
3
4 def test_inventory():
5     inv = Inventory()
6
7     # Add item
8     inv.add_item("Pen", 10)
9     assert inv.get_stock("Pen") == 10
10
11    # Remove item
12    inv.remove_item("Pen", 5)
13    assert inv.get_stock("Pen") == 5
14
15    # Add another item
16    inv.add_item("Book", 3)
17    assert inv.get_stock("Book") == 3
18
19    # Removing more than available should not go below 0
20    inv.remove_item("Pen", 10)
21    assert inv.get_stock("Pen") == 0
22
23    # Non-existing item should return 0
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\saiva\OneDrive\Desktop\AI assisted coding> python -m doctest -v LAB8.1.PY/task4.py
Inventory system passing all AI-generated test cases.
7 items had no tests:
task4
task4.Inventory
task4.Inventory.__init__
task4.Inventory.add_item
task4.Inventory.get_stock
task4.Inventory.remove_item
task4.test_inventory
0 tests in 7 items.
0 passed.
```

Explanation: This spots whether two words or phrases are made of the same letters, ignoring case, spaces, and punctuation.

Task Description #4 (Inventory Class – Apply AI to Simulate Real- World Inventory System)

- **Task:** Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.

- **Methods:**

- o add_item(name, quantity)
- o remove_item(name, quantity)
- o get_stock(name)

Example Assert Test Cases:

```
inv = Inventory()
inv.add_item("Pen", 10)
assert inv.get_stock("Pen") == 10
inv.remove_item("Pen", 5)
assert inv.get_stock("Pen") == 5
```

```
inv.add_item("Book", 3)
assert inv.get_stock("Book") == 3
```

Expected Output #4: • Fully functional class passing all assertions.

Output Screenshot:

The screenshot shows a code editor interface with several tabs at the top: task1.py, task2.py, task3.py, task4.py (which is the active tab), and task5.py. Below the tabs is a code editor window containing a Python script named LAB8.1.PY. The script contains test cases for an Inventory class, specifically task4.py. The code includes comments and assertions for adding items, removing items, and checking stock levels. At the bottom of the code editor, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. The terminal output below the editor shows the command `python -m doctest -v LAB8.1.PY/task4.py` being run, followed by the results: "Inventory system passing all AI-generated test cases.", "7 items had no tests:", a list of 7 items (task4, task4.Inventory, task4.Inventory.__init__, task4.Inventory.add_item, task4.Inventory.get_stock, task4.Inventory.remove_item, task4.test_inventory), "0 tests in 7 items.", and "0 passed.".

```
LAB8.1.PY > task4.py > ...
1  # Task 4: Inventory Class
2  # Step 1: Test Cases (TDD Style)
3
4  def test_inventory():
5      inv = Inventory()
6
7      # Add item
8      inv.add_item("Pen", 10)
9      assert inv.get_stock("Pen") == 10
10
11     # Remove item
12     inv.remove_item("Pen", 5)
13     assert inv.get_stock("Pen") == 5
14
15     # Add another item
16     inv.add_item("Book", 3)
17     assert inv.get_stock("Book") == 3
18
19     # Removing more than available should not go below 0
20     inv.remove_item("Pen", 10)
21     assert inv.get_stock("Pen") == 0
22
23     # Non-existing item should return 0
...
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\saiva\OneDrive\Desktop\AI assisted coding> python -m doctest -v LAB8.1.PY/task4.py
Inventory system passing all AI-generated test cases.
7 items had no tests:
task4
task4.Inventory
task4.Inventory.__init__
task4.Inventory.add_item
task4.Inventory.get_stock
task4.Inventory.remove_item
task4.test_inventory
0 tests in 7 items.
0 passed.
```

Explanation: This simulates a mini store system where you can add, remove, and check stock for items.

Task Description #5 (Date Validation & Formatting – Apply AI for Data Validation)

- Task: Use AI to generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates.

- Requirements:

- o Validate "MM/DD/YYYY" format.
- o Handle invalid dates.
- o Convert valid dates to "YYYY-MM-DD".

Example Assert Test Cases:

```
assert validate_and_format_date("10/15/2023") == "2023-10-15"
```

```
assert validate_and_format_date("02/30/2023") == "Invalid Date"
assert validate_and_format_date("01/01/2024") == "2024-01-01"
```

Expected Output #5: • Function passes all AI-generated assertions and handles edge cases.

Output Screenshot:

The screenshot shows a code editor interface with a dark theme. At the top, there are tabs for 'task1.py', 'task2.py', 'task3.py', 'task4.py', and 'task5.py'. The 'task5.py' tab is active. Below the tabs, the code for 'task5.py' is displayed. The code consists of two main sections: 'Step 1: Test Cases (TDD Style)' and 'Step 2: Implementation'. The test cases include assertions for valid dates ('10/15/2023' and '01/01/2024'), an invalid date ('02/30/2023'), an invalid month ('13/01/2023'), and a wrong format/non-date string ('abc'). The implementation section imports 'datetime' and defines a function 'validate_and_format_date' that returns a formatted date string. At the bottom of the editor, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is active, showing the command 'python -m doctest -v LAB8.1.PY/task5.py' and its output. The output indicates that the date validation logic passed all AI-generated test cases, with 3 items and 0 tests, resulting in 0 passed tests and a final message 'Test passed.'

```
1 # Task 5: Date Validation & Formatting
2
3 # Step 1: Test Cases (TDD Style)
4
5 def test_validate_and_format_date():
6     # Valid dates
7     assert validate_and_format_date("10/15/2023") == "2023-10-15"
8     assert validate_and_format_date("01/01/2024") == "2024-01-01"
9
10    # Invalid date (February 30 does not exist)
11    assert validate_and_format_date("02/30/2023") == "Invalid Date"
12
13    # Invalid month
14    assert validate_and_format_date("13/01/2023") == "Invalid Date"
15
16    # Wrong format / non-date string
17    assert validate_and_format_date("abc") == "Invalid Date"
18
19
20 # Step 2: Implementation
21
22 from datetime import datetime
```

```
d coding\LAB8.1.PY\task5.py'
Date validation logic passing all AI-generated test cases.
PS C:\Users\saiva\OneDrive\Desktop\AI assisted coding> python -m doctest -v LAB8.1.PY/task5.py
Date validation logic passing all AI-generated test cases.
3 items had no tests:
    task5
    task5.test_validate_and_format_date
    task5.validate_and_format_date
0 tests in 3 items.
0 passed.
Test passed.
```

Explanation: This ensures a date is valid in “MM/DD/YYYY” format and neatly converts it into “YYYY-MM-DD”.