

# **AI ASSISTED CODING**

**NAME: B.Shivamani**

**BATCH: 39**

**ROLL NO: 2303A52079**

---

## **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:
  - Password must have at least 8 characters.
  - Must include uppercase, lowercase, digit, and special character.
  - 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.

```

16-02-2026 > Task-1.py ...
1 #Write a python program to apply AI to generate at least 3 assert test cases for is_strong_password function which checks if a password is
2 import re
3 def is_strong_password(password):
4     if len(password) < 8:
5         return False
6     if not re.search(r'[A-Z]', password):
7         return False
8     if not re.search(r'[a-z]', password):
9         return False
10    if not re.search(r'[0-9]', password):
11        return False
12    if not re.search(r'[@$!%^&?@]', password):
13        return False
14    if ' ' in password:
15        return False
16    return True
17 def _run_tests():
18     # Positive case
19     assert is_strong_password("StrongPass1@") is True, "Test case 1 failed"
20
21     # Negative cases
22     assert is_strong_password("weakpass") is False, "Test case 2 failed (too weak)"
23     assert is_strong_password("NoSpecial123") is False, "Test case 3 failed (no special char)"
24     assert is_strong_password("Short1@") is False, "Test case 4 failed (too short)"
25     assert is_strong_password("NoDigitPass@") is False, "Test case 5 failed (no digit)"
26     assert is_strong_password("WITH SPACE1@") is False, "Test case 6 failed (contains space)"
27
28 if __name__ == "__main__":
29     _run_tests()

```

All tests passed

Indexing completed.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

Ln 31, Col 1 Spaces: 4 UTF-8 () Python 3.13.7 (8) Go Live

## ANALYSIS:

- The prompt clearly defines all security rules:
- Minimum 8 characters
- Uppercase, lowercase, digit, special character
- 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:

```

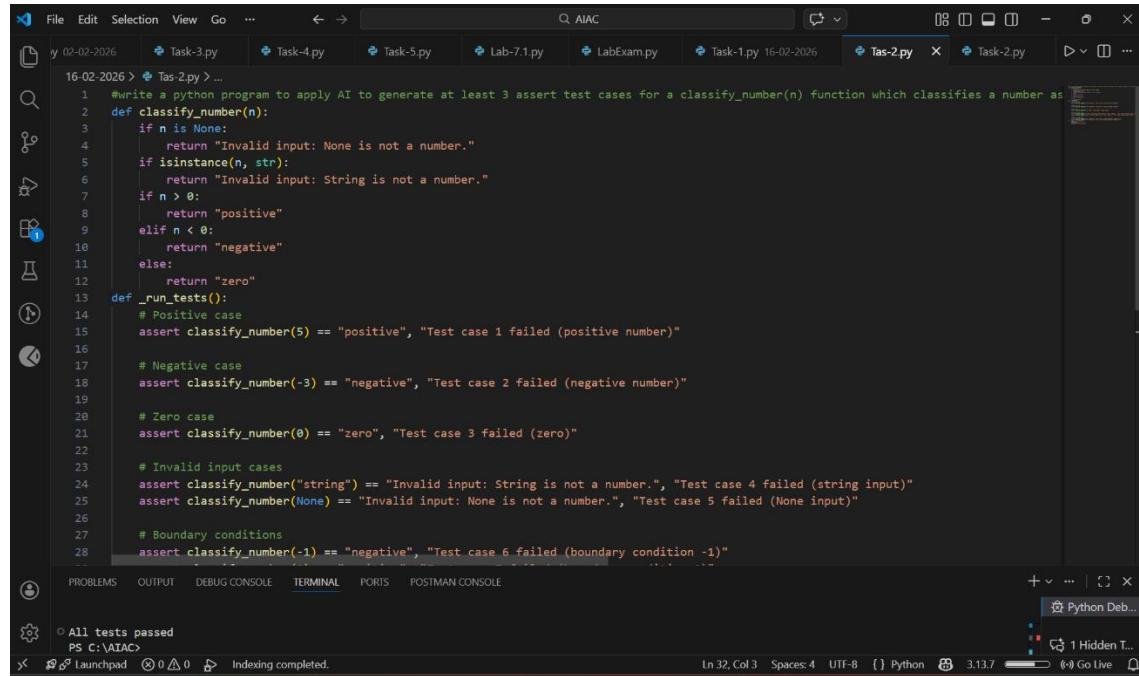
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"

```

```
assert classify_number(0) == "Zero"
```

Expected Output #2:

- Classification logic passing all assert tests.



```
16-02-2026 > Tas-2.py > ...
1   #write a python program to apply AI to generate at least 3 assert test cases for a classify_number(n) function which classifies a number as
2   def classify_number(n):
3       if n is None:
4           return "Invalid input: None is not a number."
5       if isinstance(n, str):
6           return "Invalid input: String is not a number."
7       if n > 0:
8           return "positive"
9       elif n < 0:
10          return "negative"
11     else:
12         return "zero"
13
14     def run_tests():
15         # Positive case
16         assert classify_number(5) == "positive", "Test case 1 failed (positive number)"
17
18         # Negative case
19         assert classify_number(-3) == "negative", "Test case 2 failed (negative number)"
20
21         # Zero case
22         assert classify_number(0) == "zero", "Test case 3 failed (zero)"
23
24         # Invalid input cases
25         assert classify_number("string") == "Invalid input: String is not a number.", "Test case 4 failed (string input)"
26         assert classify_number(None) == "Invalid input: None is not a number.", "Test case 5 failed (None input)"
27
28         # Boundary conditions
29         assert classify_number(-1) == "negative", "Test case 6 failed (boundary condition -1)"

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
+ | Python Deb...
All tests passed
PS C:\AIAC>
Indexing completed.
```

## ANAYLISIS:

- The task clearly states:
- Classify numbers as Positive, Negative, or Zero
- Handle invalid inputs (like string and None)
- Include boundary values (-1, 0, 1)

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

Analysis)

- Task: Use AI to generate at least 3 assert test cases for `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.

The screenshot shows a code editor window with a dark theme. The file being edited is Task-3.py, located at 16-02-2026. The code defines a function is\_anagram that takes two strings and returns True if they are anagrams. It uses regular expressions to remove punctuation and spaces from both strings and then compares their sorted versions. The code includes several assert statements to test various cases, including positive, negative, and edge cases like empty strings and identical words. A terminal tab at the bottom shows the output "All tests passed".

```
16-02-2026 > Task-3.py ...  
1 #write a python program apply AI to generate at least 3 assert test case for is_anagram(str1, str2) and implement the function which checks  
2 import re  
3 def is_anagram(str1, str2):  
4     # Remove spaces, punctuation and convert to lowercase  
5     str1_cleaned = re.sub(r'[^\w]+', '', str1.lower())  
6     str2_cleaned = re.sub(r'[^\w]+', '', str2.lower())  
7  
8     # Check if the sorted characters of both strings are the same  
9     return sorted(str1_cleaned) == sorted(str2_cleaned)  
10 def _run_tests():  
11     # Positive case  
12     assert is_anagram("Listen", "Silent") is True, "Test case 1 failed (anagram)"  
13  
14     # Negative case  
15     assert is_anagram("Hello", "World") is False, "Test case 2 failed (not anagram)"  
16  
17     # Edge cases  
18     assert is_anagram("", "") is True, "Test case 3 failed (empty strings)"  
19     assert is_anagram("Dormitory", "Dirty Room") is True, "Test case 4 failed (ignoring spaces and case)"  
20     assert is_anagram("The eyes", "They see!") is True, "Test case 5 failed (ignoring punctuation)"  
21     assert is_anagram("Same", "Same") is True, "Test case 6 failed (identical words)"  
22 if __name__ == "__main__":  
23     _run_tests()  
24     print("All tests passed")  
  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE  
All tests passed  
PS C:\AIAC>  
Launchpad 0 △ 0 Indexing completed.
```

## ANALYSIS:

- The requirements are clear: ignore case, spaces, and punctuation.
- This makes the function practical and realistic, not just basic string comparison.
- Handling empty strings and identical words improves reliability.
- It ensures the function works in all boundary situations.

## 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:

- add\_item(name, quantity)
- remove\_item(name, quantity)
- 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.

The screenshot shows the Visual Studio Code interface. The code editor displays a file named `Task-4.py` containing the following Python code:

```
16-02-2026 > Task-4.py > ...
1   #write a python program to aplly AI to generate at least 3 assert test cases for an Inventory class with stock management features. The cla
2   class Inventory:
3       def __init__(self):
4           self.stock = {}
5
6       def add_item(self, item_name, quantity):
7           if quantity < 0:
8               return "Cannot add negative quantity."
9           if item_name in self.stock:
10               self.stock[item_name] += quantity
11           else:
12               self.stock[item_name] = quantity
13           return f"Added {quantity} of {item_name}. Current stock: {self.stock[item_name]}"
14
15       def remove_item(self, item_name, quantity):
16           if item_name not in self.stock:
17               return "Item not found in inventory."
18           if quantity < 0:
19               return "Cannot remove negative quantity."
20           if self.stock[item_name] < quantity:
21               return "Not enough stock to remove."
22           self.stock[item_name] -= quantity
23           return f"Removed {quantity} of {item_name}. Current stock: {self.stock[item_name]}"
24
25       def check_stock(self, item_name):
26           return self.stock.get(item_name, "Item not found in inventory.")
27
28   def run_tests():
29       pass
```

The terminal window at the bottom shows the command line output:

```
PS C:\AIAC> c; cd 'c:\AIAC'; & 'c:\Users\chandana\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\chandana\.vscode\extensions\ms-python.python.debug-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '5415' '--' 'C:\AIAC\16-02-2026\Task-4.py'
All tests passed
PS C:\AIAC> []
```

The status bar at the bottom indicates the code has 49 lines, 30 spaces, and is in UTF-8 encoding, with Python selected as the language.

## **ANALYSIS:**

- This task simulates a real inventory system (like a shop or warehouse).
- It is more practical compared to simple functions.
- It improves understanding of object-oriented programming (OOP).
- Using a class with methods (add\_item, remove\_item, get\_stock) promotes modular design.
- It teaches how to manage data properly inside objects instead of using global variables.

## **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:
  - Validate "MM/DD/YYYY" format.
  - Handle invalid dates.
  - 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.

The screenshot shows a VS Code interface with several tabs open at the top: m.py, Task-1.py, Task-2.py, Task-5.py (which is the active tab), Task-3.py, Task-4.py, and Task-2.py. The code in Task-5.py is as follows:

```
16-02-2026 > Task-5.py > run_tests
1 #write a python program to apply AI to generate at least 3 assert test cases for validate_and_format_date(date_str) to check if a date str
2 from datetime import datetime
3 def validate_and_format_date(date_str):
4     try:
5         # Try to parse the date string
6         date_obj = datetime.strptime(date_str, "%Y-%m-%d")
7         # If successful, return the formatted date
8         return date_obj.strftime("%B %d, %Y")
9     except ValueError:
10        return "Invalid date format or non-existent date."
11 def _run_tests():
12     # Valid date case
13     assert validate_and_format_date("2023-02-16") == "February 16, 2023", "Test case 1 failed (valid date)"
14     # Invalid format case
15     assert validate_and_format_date("16-02-2023") == "Invalid date format or non-existent date.", "Test case 2 failed (invalid format)"
16     # Non-existent date case
17     assert validate_and_format_date("2023-02-30") == "Invalid date format or non-existent date.", "Test case 3 failed (non-existent date)"
18     # Leap year case
19     assert validate_and_format_date("2020-02-29") == "February 29, 2020", "Test case 4 failed (leap year)"
20     # Invalid leap year case
21     assert validate_and_format_date("2019-02-29") == "Invalid date format or non-existent date.", "Test case 5 failed (invalid leap year)"
22 if __name__ == "__main__":
23     _run_tests()
24     print("All tests passed")
```

The terminal below shows the output of running the script:

```
\Users\chandana\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '50353' '--' 'C:\AIAC\16-02-2026\Task-5.py'
PS C:\AIAC> [1]
Hidden T...
```

The status bar at the bottom indicates the file is indexed.

## ANALYSIS:

- Unlike simple string tasks, this requires:
- Checking correct format structure
- Verifying month range (1–12)
- Verifying valid days per month
- Handling leap years (e.g., Feb 29)
- The prompt includes invalid dates like "02/30/2023", which tests logical validation, not just string formatting.
- This ensures the function is logically correct, not superficially correct