LAB ASSIGNMENT-8.1

AMGOTH VIKAS NAYAK

2403A51410

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

- To introduce students to test-driven development (TDD) using AI code generation tools.

Week4 - Monday

- To enable the generation of test cases before writing code implementations.
- To reinforce the importance of testing, validation, and error handling.
- To encourage writing clean and reliable code based on AI-generated test expectations.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

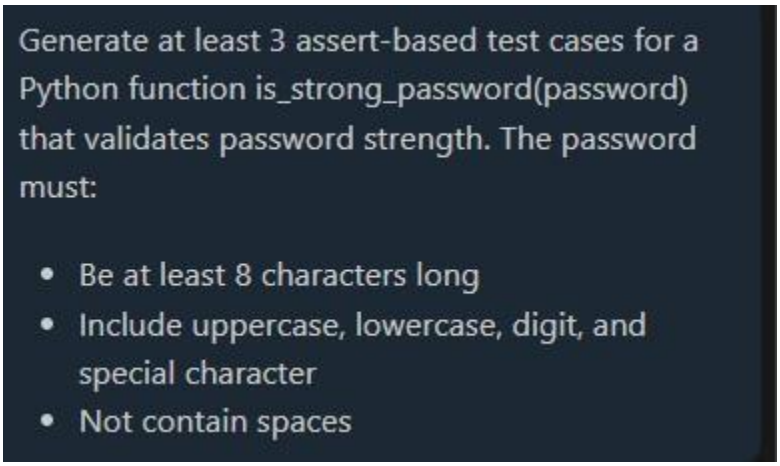
- Use AI tools to write test cases for Python functions and classes.
- Implement functions based on test cases in a test-first development style.
- Use unittest or pytest to validate code correctness.
- Analyze the completeness and coverage of AI-generated tests.
- Compare AI-generated and manually written test cases for quality and logic

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

Security Context)

PROMPT: 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.
- 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 **Prompt for test case:**



Test case:

```
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False  # No uppercase or special character
assert is_strong_password("ABCD@1234") == False  # No lowercase
assert is_strong_password("A1@ bcdef") == False  # Contains space
assert is_strong_password("Abcdefg@1") == True
```

Prompt for code:

Write a Python function is_strong_password(password) that returns True if the password meets all strength criteria and False otherwise. Use regular expressions for validation.

Code:

```
import re

def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    if not re.search(r"[A-Z]", password):
        return False
    if not re.search(r"[a-z]", password):
        return False
    if not re.search(r"\d", password):
        return False
    if not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
        return False
    return True
```

Expected Output #1:

- Password validation logic passing all AI-generated test cases.

```
All password validation tests passed.
```

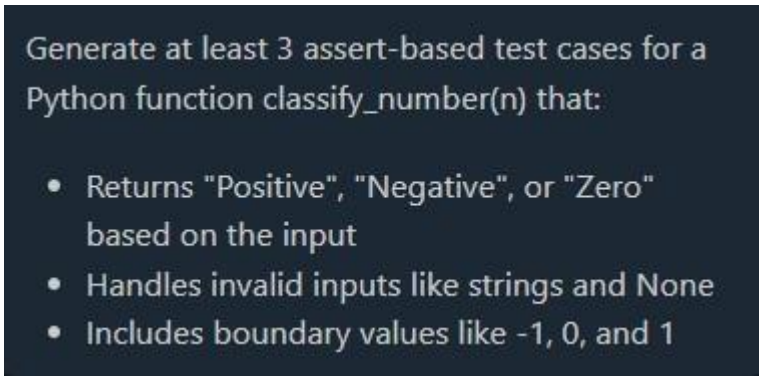
Task Description #2 (Number Classification with Loops – Apply AI for Edge Case Handling)

PROMPT:

PROMPT : 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).
- Task: Use AI to generate at least 3 assert test cases for a classify_number(n) function. Implement using loops.
- Requirements: o Classify numbers as Positive, Negative, or Zero. o Handle invalid inputs like strings and None. o 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" **prompt for test case:**



Test case:

```
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
assert classify_number("abc") == "Invalid Input"
assert classify_number(None) == "Invalid Input"
```

Prompt for code:

Write a Python function classify_number(n) using a loop that classifies the input number as "Positive", "Negative", or "Zero". Return "Invalid Input" for non-numeric values.

Code:

```
def classify_number(n):
    for _ in range(1):  # Loop used as per requirement
        if not isinstance(n, (int, float)):
            return "Invalid Input"
        if n > 0:
            return "Positive"
        elif n < 0:
            return "Negative"
        else:
            return "Zero"
```

Expected Output #2:

• Classification logic passing all assert tests.

```
All number classification tests passed.
```

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

PROMPT : 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).

• Function correctly identifying anagrams and passing all AI-generated tests.

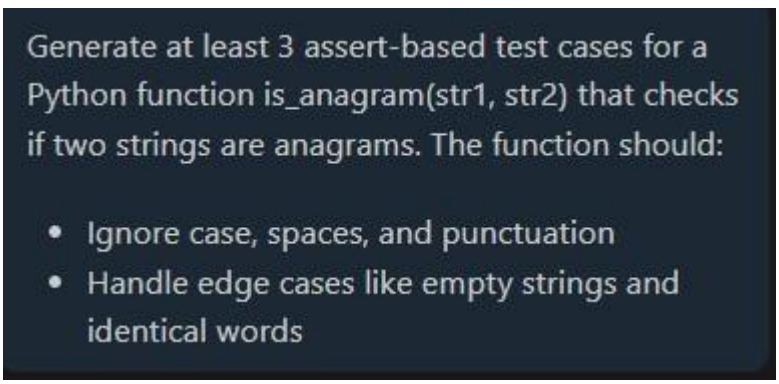
• Task: Use AI to generate at least 3 assert test cases for is_anagram(str1, str2) and implement the function.

• Requirements:

o Ignore case, spaces, and punctuation. o 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

Prompt for test case:



Test case:

```
assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True
assert is_anagram("", "") == True
assert is_anagram("School master", "The classroom") == True
```

Prompt for code:

Write a Python function is_anagram(str1, str2) that returns True if the strings are anagrams, ignoring case, spaces, and punctuation. Use character filtering and sorting.

Code:

```
import string

def is_anagram(str1, str2):
    def clean(s):
        return sorted(c.lower() for c in s if c.isalnum())
    return clean(str1) == clean(str2)
```

Expected Output #3:

- Function correctly identifying anagrams and passing all AI-generated tests.

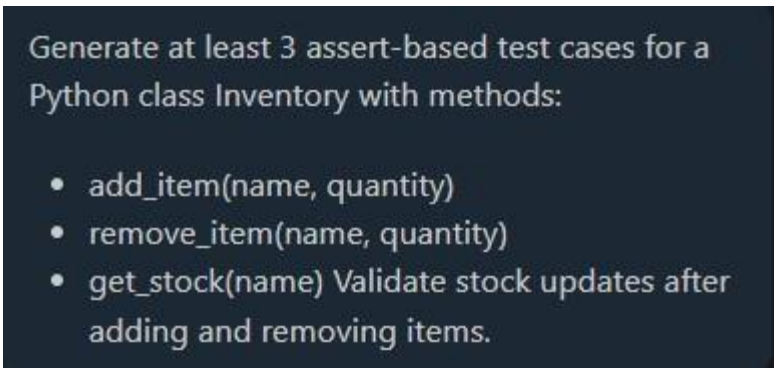
```
All anagram tests passed.
```

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

PROMPT: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)
- 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 **Prompt for test case:**



Test case:

```
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
```

Prompt for code:

```
Write a Python class Inventory that manages
stock levels. Implement methods to add items,
remove items (without going negative), and
retrieve current stock.
```

Code:

```
class Inventory:
    def __init__(self):
        self.stock = {}

    def add_item(self, name, quantity):
        if name in self.stock:
            self.stock[name] += quantity
        else:
            self.stock[name] = quantity

    def remove_item(self, name, quantity):
        if name in self.stock:
            self.stock[name] = max(0, self.stock[name] - quantity)

    def get_stock(self, name):
        return self.stock.get(name, 0)
```

Expected Output #4:

• Fully functional class passing all assertions.

```
All inventory management tests passed.
```

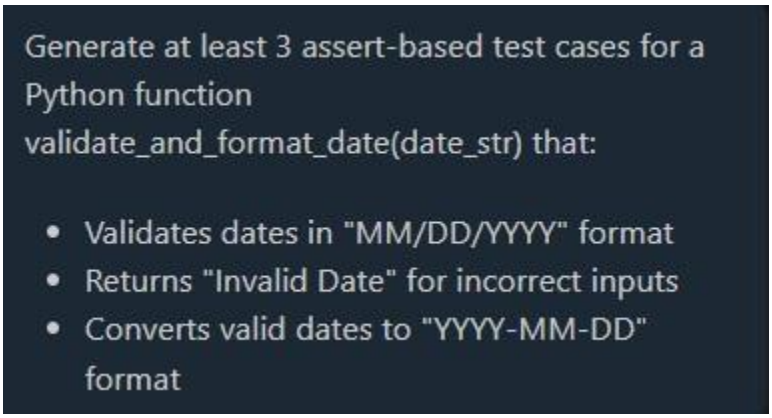
Task Description #5 (Date Validation & Formatting – Apply AI for Data Validation)

PROMPT: generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates.

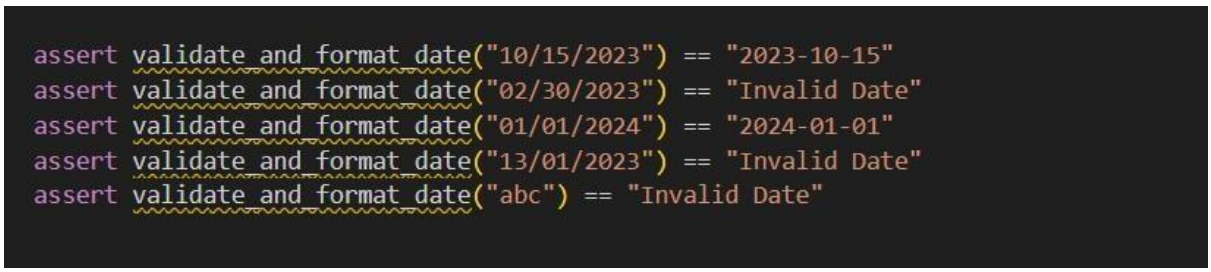
- Requirements:
  - Validate "MM/DD/YYYY" format.
  - Handle invalid dates.
  - Convert valid dates to "YYYY-MM-DD".
- Task: Use AI to generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates.
- Requirements: o Validate "MM/DD/YYYY" format. o Handle invalid 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"

Prompt for test case:



Test case:



```
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"
assert validate_and_format_date("13/01/2023") == "Invalid Date"
assert validate_and_format_date("abc") == "Invalid Date"
```

Prompt for code:

```
Write a Python function
validate_and_format_date(date_str) that checks if
the input is a valid date in "MM/DD/YYYY"
format. If valid, return it in "YYYY-MM-DD"
format; otherwise, return "Invalid Date".
```

Code:

```python
from datetime import datetime

def validate_and_format_date(date_str):
    try:
        date_obj = datetime.strptime(date_str, "%m/%d/%Y")
        return date_obj.strftime("%Y-%m-%d")
    except ValueError:
        return "Invalid Date"
```

Expected Output #5:

- Function passes all AI-generated assertions and handles edge cases.

```
All date validation and formatting tests passed.
```

Final summary:

```
✅ Task 1 Output: All password tests passed.
✅ Task 2 Output: All number classification tests passed.
✅ Task 3 Output: All anagram tests passed.
✅ Task 4 Output: All inventory tests passed.
✅ Task 5 Output: All date validation tests passed.
```