# AI ASSISTED CODING

# ASSIGNMENT – 8.1

# HALL TICKET :2403A51266

# BATCH :12

# QUESTION:

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

• 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)
• 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.
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.
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.
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.

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.

Deliverables (For All Tasks)

1. AI-generated prompts for code and test case generation.
2. At least 3 assert test cases for each task.
3. AI-generated initial code and execution screenshots.
4. Analysis of whether code passes all tests.
5. Improved final version with inline comments and explanation.
6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output

# TASK 1:

```
import string

def is_strong_password(password):
    if len(password) < 8:
        return False
    if ' ' in password:
        return False
    has_upper = any(c.isupper() for c in password)
    has_lower = any(c.islower() for c in password)
    has_digit = any(c.isdigit() for c in password)
    has_special = any(c in string.punctuation for c in password)
    return has_upper and has_lower and has_digit and has_specialL
```

# OUTPUT:

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

# PROMPT:

Generate 3 assert test cases for a Python function `is_strong_password(password)` that checks password strength. The rules: minimum 8 chars, at least one uppercase, one lowercase, one digit, one special character, and no spaces.

# TASK 2:

```
def classify_number(n):
    if not isinstance(n, (int, float)):
        return "Invalid input"
    if n > 0:
```

```
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"
```

# OUTPUT:

```
assert classify_number(10) == "Positive"
assert classify_number(-1) == "Negative"
assert classify_number(0) == "Zero"
assert classify_number("abc") == "Invalid input"
assert classify_number(None) == "Invalid input"
```

# PROMPT:

Generate 3 assert test cases for a `classify_number(n)` function that classifies as Positive, Negative, or Zero, and handles invalid inputs like strings and None.

# TASK :3

```
import string

def is_anagram(str1, str2):
    translator = str.maketrans('', '', string.punctuation + ' ')
    cleaned1 = str1.translate(translator).lower()
    cleaned2 = str2.translate(translator).lower()
    return sorted(cleaned1) == sorted(cleaned2)
```

# OUTPUT:

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

## PROMPT:

Generate 3 assert test cases for `is_anagram(str1, str2)` ignoring case, punctuation, and spaces.

## TASK 4:

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

    def add_item(self, name, quantity):
        self.stock[name] = self.stock.get(name, 0) + 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)
```

## OUTPUT:

```
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:

Generate 3 assert test cases for an `Inventory` class with methods: add_item(name, qty), remove_item(name, qty), get_stock(name).

# TASK 5:

```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"
```

# OUTPUT:

```
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:

Generate 3 assert test cases for a `validate_and_format_date(date_str)` function that converts "MM/DD/YYYY" to "YYYY-MM-DD", and handles invalid dates.