AI ASSISTED CODING

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

By- K.Sidhartha Reddy [2403A51293]

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)

• 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:

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:

• 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:

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"
```

Expected Output:

• 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:

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
```

Expected Output:

• 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.

Requirements:

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:

• 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:

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:

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

Deliverables (For All Tasks)

• AI-generated prompts for code and test case generation.

• At least 3 assert test cases for each task.

• AI-generated initial code and execution screenshots.

• Analysis of whether code passes all tests.

- Improved final version with inline comments and explanation.
- Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.