

# **AI - Assisted Coding**

## **Lab Assignment - 8.1**

**NAME:** B.Rohith

**HT.No:** 2303A51882

**Batch No:** 02

### **Lab 8: 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.
- 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:**

- 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.

### Copilot Suggested Code :

```
task_1.py > ...
1  # Generate at least 3 assert test cases for is_strong_password(password) and implement the validator function.
2  # Example Assert Test Cases:
3  # assert is_strong_password("Abcd@123") == True
4  # assert is_strong_password("abcd123") == False
5  # assert is_strong_password("ABCD@1234") == True
6
7 def is_strong_password(password):
8     if len(password) < 8:
9         return False
10    if ' ' in password:
11        return False
12    has_upper = any(c.isupper() for c in password)
13    has_lower = any(c.islower() for c in password)
14    has_digit = any(c.isdigit() for c in password)
15    has_special = any(not c.isalnum() for c in password)
16
17    return has_upper and has_lower and has_digit and has_special
18 # AI-generated test cases
19 assert is_strong_password("Abcd@123") == True
20 assert is_strong_password("abcd123") == False
21 assert is_strong_password("ABCD@1234") == False
```

## Output :

```
● PS C:\Btech\3-2\Ai AC\Assignment_8.1> & C:\Python314\python.exe "c:/Btech/3-2/Ai AC/Assignment_8.1/task_1.py"
● PS C:\Btech\3-2\Ai AC\Assignment_8.1> python -m doctest task_1.py
❖ PS C:\Btech\3-2\Ai AC\Assignment_8.1> █
```

## 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 #2:

- Classification logic passing all assert tests.

## Copilot Suggested Code :

```
task_2.py > ...
1  # Generate at least 3 assert test cases for a classify_number(n) function. Implement using loops.
2  # Example Assert Test Cases:
3  # assert classify_number(10) == "Positive"
4  # assert classify_number(-5) == "Negative"
5  # assert classify_number(0) == "Zero"
6
7  def classify_number(n):
8      if n is None or isinstance(n, str):
9          return "Invalid input"
10
11     if n > 0:
12         return "Positive"
13     elif n < 0:
14         return "Negative"
15     else:
16         return "Zero"
17
18 # AI-generated test cases
19 assert classify_number(10) == "Positive"
20 assert classify_number(-5) == "Negative"
21 assert classify_number(0) == "Zero"
```

## Output :

```
● PS C:\Btech\3-2\Ai AC\Assignment_8.1> & C:\Python314\python.exe "c:/Btech/3-2/Ai AC/Assignment_8.1/task_2.py"
● PS C:\Btech\3-2\Ai AC\Assignment_8.1> python -m doctest task_2.py
❖ PS C:\Btech\3-2\Ai AC\Assignment_8.1> █
```

## 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:**

- 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 #3:

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

## Copilot Suggested Code :

```
task_3.py > ...
1  # Generate at least 3 assert test cases for is_anagram(str1, str2) and implement the function.
2  # Example Assert Test Cases:
3  # assert is_anagram("listen", "silent") == True
4  # assert is_anagram("hello", "world") == False
5  # assert is_anagram("Dormitory", "Dirty Room") == True
6
7 def is_anagram(str1, str2):
8     # Remove spaces and punctuation, and convert to lowercase
9     def clean_string(s):
10         return ''.join(e for e in s if e.isalnum()).lower()
11
12     cleaned_str1 = clean_string(str1)
13     cleaned_str2 = clean_string(str2)
14
15     # Sort the characters of both strings and compare
16     return sorted(cleaned_str1) == sorted(cleaned_str2)
17 # AI-generated test cases
18 assert is_anagram("listen", "silent") == True
19 assert is_anagram("hello", "world") == False
20 assert is_anagram("Dormitory", "Dirty Room") == True
```

## **Output :**

```
● PS C:\Btech\3-2\Ai AC\Assignment_8.1> & C:\Python314\python.exe "c:/Btech/3-2/Ai AC/Assignment_8.1/task_3.py"
● PS C:\Btech\3-2\Ai AC\Assignment_8.1> python -m doctest task_3.py
❖ PS C:\Btech\3-2\Ai AC\Assignment_8.1>
```

## **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.

## Copilot Suggested Code :

```
task_4.py > ...
1  # Generate at least 3 assert-based tests for an
2  # Inventory class with stock management.
3  # Example Assert Test Cases:
4  # inv = Inventory()
5  # inv.add_item("Pen", 10)
6  # assert inv.get_stock("Pen") == 10
7  # inv.remove_item("Pen", 5)
8  # assert inv.get_stock("Pen") == 5
9  # inv.add_item("Book", 3)
10 # assert inv.get_stock("Book") == 3
11
12 class Inventory:
13     def __init__(self):
14         self.stock = {}
15
16     def add_item(self, name, quantity):
17         if name in self.stock:
18             self.stock[name] += quantity
19         else:
20             self.stock[name] = quantity
21
22     def remove_item(self, name, quantity):
23         if name in self.stock and self.stock[name] >= quantity:
24             self.stock[name] -= quantity
25         else:
26             raise ValueError("Not enough stock to remove")
27
28     def get_stock(self, name):
29         return self.stock.get(name, 0)
30 # Assert-based tests
31 inv = Inventory()
32 inv.add_item("Pen", 10)
33 assert inv.get_stock("Pen") == 10, "Test Case 1 Failed: Expected 10"
34 inv.remove_item("Pen", 5)
35 assert inv.get_stock("Pen") == 5, "Test Case 2 Failed: Expected 5"
36 inv.add_item("Book", 3)
37 assert inv.get_stock("Book") == 3, "Test Case 3 Failed: Expected 3"
38 print("All test cases passed!")
```

## **Output :**

```
● PS C:\Btech\3-2\Ai AC\Assignment_8.1> & C:\Python314\python.exe "c:/Btech/3-2/Ai AC/Assignment_8.1/task_4.py"
  All test cases passed!
● PS C:\Btech\3-2\Ai AC\Assignment_8.1> python -m doctest task_4.py
  All test cases passed!
❖ PS C:\Btech\3-2\Ai AC\Assignment_8.1> █
```

## **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.

## Copilot Suggested Code :

```
task_5.py > ...
1  # Generate at least 3 assert test cases for
2  # validate_and_format_date(date_str) to check and convert dates.
3  # Example Assert Test Cases:
4  # assert validate_and_format_date("10/15/2023") == "2023-10-15"
5  # assert validate_and_format_date("02/30/2023") == "Invalid Date"
6  # assert validate_and_format_date("01/01/2024") == "2024-01-01"
7
8  def validate_and_format_date(date_str):
9      # Split the input date string into components
10     try:
11         month, day, year = map(int, date_str.split('/'))
12     except ValueError:
13         return "Invalid Date"
14
15     # Validate the month
16     if month < 1 or month > 12:
17         return "Invalid Date"
18
19     # Validate the day based on the month and leap year
20     if month in [1, 3, 5, 7, 8, 10, 12]:
21         if day < 1 or day > 31:
22             return "Invalid Date"
23     elif month in [4, 6, 9, 11]:
24         if day < 1 or day > 30:
25             return "Invalid Date"
26     elif month == 2:
27         if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
28             if day < 1 or day > 29:
29                 return "Invalid Date"
30         else:
31             if day < 1 or day > 28:
32                 return "Invalid Date"
33
34     # If all validations pass, format the date to "YYYY-MM-DD"
35     return f"{year:04d}-{month:02d}-{day:02d}"
36 # AI-generated assert test cases
37 assert validate_and_format_date("10/15/2023") == "2023-10-15"
38 assert validate_and_format_date("02/30/2023") == "Invalid Date"
39 assert validate_and_format_date("01/01/2024") == "2024-01-01"
```

## Output :

```
● PS C:\Btech\3-2\Ai AC\Assignment_8.1> & C:\Python314\python.exe "c:/Btech/3-2/Ai AC/Assignment_8.1/task_5.py"
● PS C:\Btech\3-2\Ai AC\Assignment_8.1> python -m doctest task_5.py
❖ PS C:\Btech\3-2\Ai AC\Assignment_8.1> █
```