

## AI ASSIGNMENT-8.5

**Name:** R.Akshitha

**Ht.No:**2303A52464

**Batch:**36

**Task Description #1** (Username Validator – Apply AI in Authentication Context)

- Task: Use AI to generate at least 3 assert test cases for a function `is_valid_username(username)` and then implement the function using Test-Driven Development principles.

- Requirements:

- o Username length must be between 5 and 15 characters.
- o Must contain only alphabets and digits.
- o Must not start with a digit.
- o No spaces allowed.

Example Assert Test Cases:

```
assert is_valid_username("User123") == True  
assert is_valid_username("12User") == False  
assert is_valid_username("Us er") == False
```

Expected Output #1:

- Username validation logic successfully passing all AI-generated test cases.

Output:

The screenshot shows the Gemini AI development environment. The main area displays a Python script named 'username validator' with the following code:

```

# Function
def is_valid_username(username):
    if not isinstance(username, str):
        return False
    if len(username) < 5:
        return False
    if not username.isalnum():
        return False
    if username[0].isdigit():
        return False
    return True

# Test cases
assert is_valid_username('user1')
assert is_valid_username('user_1')
assert is_valid_username('user1_')

```

An AI assistant window is open over the code, titled '#1 (Username Validator – Apply AI in Authentication Context)'. It contains the following text:

- <> Empty cell
- Let's implement the username validator function and its test cases in the selected cell.

At the bottom of the AI window, there are buttons for 'Accept & Run', 'Accept', and 'Cancel'. A message box says 'What can I help you build?' with a '+' button. The status bar at the bottom right says 'Gemini 2.5 Flash ▾'.

## Task Description #2 (Even–Odd & Type Classification – Apply)

AI for Robust Input Handling)

- Task: Use AI to generate at least 3 assert test cases for a function `classify_value(x)` and implement it using conditional logic and loops.

• Requirements:

- If input is an integer, classify as "Even" or "Odd".
- If input is 0, return "Zero".
- If input is non-numeric, return "Invalid Input".

Example Assert Test Cases:

```

assert classify_value(8) == "Even"
assert classify_value(7) == "Odd"
assert classify_value("abc") == "Invalid Input"

```

Expected Output #2:

- Function correctly classifying values and passing all test cases.

Output:

The screenshot shows the Gemini AI interface. On the left, there's a code editor window titled 'Gemini' containing Python code for a 'classify\_input' function and some test cases. A modal dialog box is open in the center, titled 'Empty cell'. It contains the instruction: 'Let's create a Python function classify\_input that determines if an input is an even or odd integer, or classifies its type if it's not an integer, and add some test cases.' Below this is a button bar with 'Accept & Run', 'Accept', and 'Cancel'. At the bottom of the dialog is a text input field asking 'What can I help you build?' and a note: 'Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)'.

### Task Description #3 (Palindrome Checker – Apply AI for

#### String Normalization)

- Task: Use AI to generate at least 3 assert test cases for a function `is_palindrome(text)` and implement the function.

- Requirements:

- o Ignore case, spaces, and punctuation.
  - o Handle edge cases such as empty strings and single characters.

Example Assert Test Cases:

```
assert is_palindrome("Madam") == True  
assert is_palindrome("A man a plan a canal Panama") ==  
True  
assert is_palindrome("Python") == False
```

Expected Output #3:

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

Output:

```

# Function
def is_palindrome(text):
    # Normalize the string: convert to lowercase and remove non-alphanumeric characters
    normalized_text = ''.join([c.lower() for c in text if c.isalnum()])
    # Check if the normalized text is equal to its reverse
    return normalized_text == normalized_text[::-1]

# Test cases
assert is_palindrome("Racecar") == True
assert is_palindrome("A man a plan a canal Panama") == True
assert is_palindrome("Hello World") == False
assert is_palindrome("No lemon no melon") == True
assert is_palindrome("Python") == False
assert is_palindrome("Madam") == True
assert is_palindrome("Was it a car or a cat I saw") == True

```

**M #3 (Palindrome Checker – Apply AI for String Normalization)**

Let's create a Python function `is_palindrome` that checks if a string is a palindrome, ignoring non-alphanumeric characters and case, and add some test cases.

**Accept & Run**   **Accept**   **Cancel**

What can I help you build?

Gemini 2.5 Flash

## Task Description #4 (BankAccount Class – Apply AI for

Object-Oriented Test-Driven Development)

- Task: Ask AI to generate at least 3 assert-based test cases for a BankAccount class and then implement the class.

- Methods:

- o `deposit(amount)`

- o `withdraw(amount)`

- o `get_balance()`

Example Assert Test Cases:

```
acc = BankAccount(1000)
```

```
acc.deposit(500)
```

```
assert acc.get_balance() == 1500
```

```
acc.withdraw(300)
```

```
assert acc.get_balance() == 1200
```

Expected Output #4:

- Fully functional class that passes all AI-generated assertions.

Output:

The screenshot shows the Gemini AI interface. At the top, there's a toolbar with 'Commands', 'Code', 'Text', 'Run all', and other icons. On the right, there are status indicators for RAM and Disk. The main area is titled 'Task-04' and contains a code editor for a 'BankAccount' class. The code includes methods for initialization, deposit, withdraw, and balance retrieval, with validation logic for non-negative initial balance and amounts. A tooltip provides context for the current task: '#4 (BankAccount Class – Apply AI for Object-Oriented Test-Driven Development)'. Below the code editor is a message input field asking 'What can I help you build?' and a button labeled '+'. To the right of the message input is a note: 'Gemini can make mistakes so double-check it and use code with caution.' A 'Gemini 2.5 Flash' dropdown is also present. Below the code editor, a terminal window shows a Python script with several assert statements for testing the BankAccount class. The output of the script indicates that all tests passed.

```

# Function: BankAccount Class
class BankAccount:
    def __init__(self, initial_balance=0):
        if not isinstance(initial_balance, (int, float)) or initial_balance < 0:
            raise ValueError
        self.balance = initial_balance

    def deposit(self, amount):
        if not isinstance(amount, (int, float)):
            raise ValueError
        self.balance += amount
        return self.balance

    def withdraw(self, amount):
        if not isinstance(amount, (int, float)):
            raise ValueError
        if amount > self.balance:
            raise ValueError
        self.balance -= amount
        return self.balance

```

```

try:
    account10 = BankAccount(-50)
    assert False, "Test 10 Failed: Expected ValueError for negative initial balance"
except ValueError as e:
    assert str(e) == "Initial balance must be a non-negative number.", f"Test 10 Failed: Wrong error message: {e}"

# Test 11: Initial balance as float
account11 = BankAccount(100.50)
assert account11.get_balance() == 100.50, f"Test 11 Failed: Expected 100.50, got {account11.get_balance()}"

# Test 12: Deposit float amount
account12 = BankAccount(50)
account12.deposit(25.75)
assert account12.get_balance() == 75.75, f"Test 12 Failed: Expected 75.75, got {account12.get_balance()}"

# Test 13: Withdraw float amount
account13 = BankAccount(100.25)
account13.withdraw(10.15)
assert account13.get_balance() == 90.10, f"Test 13 Failed: Expected 90.10, got {account13.get_balance()}"

print("All BankAccount tests passed!")

```

## Task Description #5 (Email ID Validation – Apply AI for Data Validation)

- Task: Use AI to generate at least 3 assert test cases for a function `validate_email(email)` and implement the function.

- Requirements:

- Must contain @ and .
- Must not start or end with special characters.
- Should handle invalid formats gracefully.

**Example Assert Test Cases:**

```

assert validate_email("user@example.com") == True
assert validate_email("userexample.com") == False
assert validate_email("@gmail.com") == False

```

**Expected Output #5:**

- Email validation function passing all AI-generated test cases and handling edge cases correctly.

## Output:

The screenshot shows the Gemini AI development interface. On the left is a sidebar with various icons for file operations. The main area is a code editor titled "Task-05" containing Python code for validating emails. A tooltip box is open over the code, providing instructions for creating a validation function and listing common email patterns. Below the code editor is a text input field asking "What can I help you build?" and a "Gemini 2.5 Flash" button. At the bottom, there's a status bar showing the time as 11:59 AM and the Python version as 3.

```
+# Function
+import re
+
+def is_valid_email(email):
+    if not isinstance(email, str):
+        return False
+    # Regular expression for validating an Email
+    # This pattern checks for general email patterns
+    # It allows letters, numbers, and underscores
+    # The top-level domain can be anything
+    pattern = r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.+]"
+    if re.fullmatch(pattern, email):
+        return True
+    else:
+        return False
+
+# Test cases
+assert is_valid_email("test@example.com")
+assert is_valid_email("job@company.com")
+assert is_valid_email("info@site.com")
```