

ASSIGNMENT-8.5

Name: Mohammed safwan farooqi

Hall No:2303A52246

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. In the center is a code editor window titled "username validator" containing Python code. The code defines a function `is_valid_username` that checks if a username is valid based on length and character type. It also includes some test cases using `assert` statements. To the right of the code editor is a modal window titled "#1 (Username Validator – Apply AI in Authentication Context)". The modal contains a message: "Let's implement the username validator function and its test cases in the selected cell." Below this are three buttons: "Accept & Run" (with a play icon), "Accept" (with a checkmark icon), and "Cancel" (with a cross icon). At the bottom of the modal is a text input field asking "What can I help you build?" and a "Gemini 2.5 Flash" dropdown menu.

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 floating code completion dialog box is open over the code, with the title 'Empty cell'. Inside the dialog, there's a blue arrow icon followed by the text: '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, there are three buttons: 'Accept & Run' (with a checkmark), 'Accept', and 'Cancel'. At the bottom of the dialog, there's a text input field with the placeholder 'What can I help you build?' and a small 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:

- Ignore case, spaces, and punctuation.
- 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:

The screenshot shows the Gemini AI interface with a code editor window titled "Task-03". The code is as follows:

```
+# 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") == False
+assert is_palindrome("No") == True
+assert is_palindrome("Python") == False
+assert is_palindrome("Was it a car or a cat I saw") == True
+
```

A tooltip from Gemini provides instructions: "#3 (Palindrome Checker - Apply AI for String Normalization)" with options "Empty cell", "Accept & Run", "Accept", and "Cancel". A message at the bottom says "What can I help you build?".

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 Algenerated assertions.

Output:

The screenshot shows the Gemini AI interface with the following details:

- Code Completion Panel (Top):** Shows the `BankAccount` class definition:

```
+# 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
```
- Task Description (Top Right):** A modal window titled "#4 (BankAccount Class – Apply AI for Object-Oriented Test-Driven Development)" with options: "Empty cell", "Accept & Run", "Accept", and "Cancel".
- Code Editor (Bottom):** Displays a test script for `BankAccount`:

```
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!")
```
- Status Bar (Bottom Left):** Shows "[5] Os" and "... 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:
 - o Must contain @ and .
 - o Must not start or end with special characters.
 - o 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 interface with a code editor window titled "Task-05". The code being generated is:

```
+# Function
+import re
+
+def is_valid_email(email):
+    if not isinstance(email, str):
+        return False
+    # Regular expression pattern
+    # This pattern checks if the email address contains letters, numbers, and a top-level domain
+    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("john.doe@example.com")
+assert is_valid_email("info@example.com")
+assert is_valid_email("invalid_email@.com")
```

A modal window titled "#5 (Email ID Validation - Apply AI for Data Validation)" is open, providing instructions to create the function and offering options to "Accept & Run", "Accept", or "Cancel". Below the modal, a text input field asks "What can I help you build?" and a "Gemini 2.5 Flash" button is visible.