

ASSIGNMENT-8.5

Name: B. Bhargava Reddy

Hall No:2303A52476

Batch:35

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:

- Username length must be between 5 and 15 characters.
- Must contain only alphabets and digits.
- Must not start with a digit.
- 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 interface. On the left is a code editor with Python code for a 'username validator'. The code defines a function `is_valid_username` that checks if a string is a valid username based on length and character rules. Below the function is a section of test cases using `assert` statements. A sidebar on the right displays AI-generated test cases for the function. One test case is highlighted with a red border and labeled '#1 (Username Validator - Apply AI in Authentication Context)'. The sidebar also includes a message from Gemini asking what it can help build, with options to accept or cancel.

```

Gemini
-----
username validator

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('user123')
assert is_valid_username('user_123')
assert is_valid_username('user123_')

```

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 'Function' containing Python code for classifying integers. A tooltip box is overlaid on the code, providing a task description: '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 the code editor is a small AI assistant window asking 'What can I help you build?'. At the bottom right of the screen, there's a status bar with 'Gemini 2.5 Flash'.

```
+# Function
+def classify_input(value):
+    if isinstance(value, int):
+        if value % 2 == 0:
+            return f"{value} is an even integer."
+        else:
+            return f"{value} is an odd integer."
+    else:
+        return f"{value} is not an integer."
+
+# Test cases
+assert classify_input(4) ==
+assert classify_input(7) ==
+assert classify_input("hello") ==
+assert classify_input(3.14) ==
+assert classify_input([1, 2]) ==
```

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 a code editor interface with a sidebar on the left containing icons for file, code, text, run, and help. The main area is titled "Task-03" and contains a code cell labeled "Gemini". The code cell displays the following Python code:+# Function
+def is_palindrome(text):
+ # Normalize the string: convert to lowercase and remove non-alphanumeric characters
+ normalized_text = ''.join([char.lower() for char in text if char.isalnum()])
+ # Check if the normalized text is equal to its reverse
+ return normalized_text == normalized_text[::-1]
+
+# Test cases
+assert is_palindrome("Race") == False
+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("Madam") == True
+assert is_palindrome("Was") == TrueA tooltip from Gemini provides instructions: "#3 (Palindrome Checker – Apply AI for String Normalization)" followed by "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". Below the tooltip are buttons for "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 AI-generated assertions.

Output:

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

- Code Completion:** A tooltip for the `BankAccount` class definition provides documentation and suggestions:
 - #4 (BankAccount Class – Apply AI for Object-Oriented Test-Driven Development)
 - <> Empty cell
 - Let's create a `BankAccount` class with methods for `deposit`, `withdraw`,
 - Accept & Run ✓ Accept ✖ Cancel
- Test Script:** Below the completion, a test script is shown with several failed assertions:

```
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:** Shows 5 tests, 0 errors, and a green checkmark icon.

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:

