

ASSIGNMENT-8.5

Name: G.Vignesh

HT.No:2303A52359

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 interface with a Python script titled "username validator". The script contains a function `is_valid_username` with several conditional checks for length, type, and first character. Below the function is a section of test cases using `assert` statements. A tooltip from Gemini suggests implementing the function and its test cases. The interface includes standard code editor features like a sidebar with icons, a toolbar at the top, and a status bar at the bottom.

```

username validator

Gemini
+ 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('user2')
+assert is_valid_username('user3')

```

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. In the main workspace, there is a code editor window titled 'Gemini' containing the following Python code:

```
+# 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("he
+assert classify_input(3.14)
+assert classify_input([1,
+
```

A tooltip box is open over the code, asking '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.' It includes buttons for 'Accept & Run', 'Accept', and 'Cancel'. Below the code editor, a text input field says 'What can I help you build?' and a status bar at the bottom right says 'Gemini 2.5 Flash'.

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:

The screenshot shows a Jupyter Notebook interface with a tab titled "Task-03". In the code cell, there is Python code for a palindrome checker function and some test cases. A tooltip from "Gemini" provides instructions for applying AI to string normalization. A user interaction window from "Gemini 2.5 Flash" asks what can be built, with options to accept or cancel.

```
+# Function
+def is_palindrome(text):
+    # Normalize the string: convert to lowercase and remove non-alphanumeric characters
+    normalized_text = ''.join(c.lower() if c.isalnum() else '' for c in text)
+
+    # 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
```

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 Panel (Top):** Shows the `BankAccount` class definition with AI-generated documentation and a help box asking "What can I help you build?".
- Test Results Panel (Bottom):** Displays a series of test cases (try-except blocks) for the `BankAccount` class, showing expected behavior for various operations like depositing and withdrawing.
- Status Bar:** Shows "5s" and "0s" indicating the time taken for the tests.
- Header:** Includes search, code, text, run all, and disk status indicators.

```
+# 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!")
```

... 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:

Commands + Code + Text ▶ Run all ⌂

Task-05

Gemini

```
# Function
import re

def is_valid_email(email):
    if not isinstance(email, str):
        return False
    # Regular expression
    # This pattern checks
    # It allows letters, numbers, and symbols
    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.org")
assert is_valid_email("info@company.co.uk")
assert is_valid_email("user.name+alias@domain.net")
assert is_valid_email("user@domain.com")
```

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

Empty cell

Let's create a Python function `is_valid_email` that validates email addresses based on common patterns, and include test cases for various

Accept & Run ✓ Accept X Cancel

What can I help you build?

+ Gemini 2.5 Flash ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Variables Terminal

✓ 11:59 AM Python 3

