

ASSIGNMENT-8.5

Name: Mohammed Adnan Awaise

Hall No:2303A52244

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:
 - 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 username is valid based on length and character type. It also includes a section for test cases using `assert` statements. A sidebar on the right contains an AI-generated task description: '#1 (Username Validator – Apply AI in Authentication Context)'. It suggests implementing the function and its test cases. Below this is a 'What can I help you build?' input field and a 'Gemini 2.5 Flash' dropdown.

```

username validator

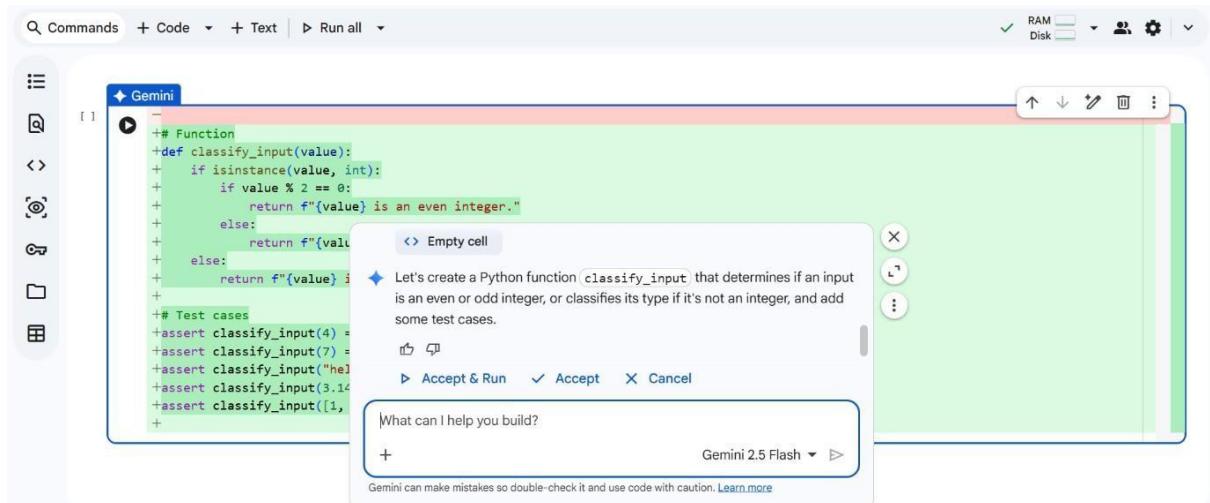
Gemini
+ Function
+def is_valid_username(username):
+    if not isinstance(username, str):
+        return False
+    if len(username) < 5:
+        return False
+    if not username.isalnum():
+        return False
+    if username[0].isdigit():
+        return False
+    return True
+
+ Test cases
+assert is_valid_username('abc')
+assert is_valid_username('12345')
+assert is_valid_username('a1b2c3')

```

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"
- Function correctly classifying values and passing all test cases.

Output:



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:

```

Gemini
[ ] -
+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 the original text
+    return normalized_text == text
+
## 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 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
+

```

A tooltip from Gemini provides instructions: "#3 (Palindrome Checker – Apply AI for String Normalization)". It says: "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." Below the tooltip are buttons for "Accept & Run", "Accept", and "Cancel". A message at the bottom of the tooltip 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:

```

# 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

```

M #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

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](#)

```

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:

The screenshot shows a code editor interface with the following details:

- Title Bar:** Task-05
- Code Area:**

```
+# Function
+import re
+
+def is_valid_email(email):
+    if not isinstance(email, str):
+        return False
+    # Regular expression to check email format
+    # It allows letters, numbers, and the 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") == True
+assert is_valid_email("john.doe@example.com") == True
+assert is_valid_email("info@example.com") == True
+assert is_valid_email("invalid_email@.com") == False
+assert is_valid_email("user@example..com") == False
```
- AI Assistant Panel:**
 - A message from Gemini: "#5 (Email ID Validation – Apply AI for Data Validation)"
 - A button: "Empty cell"
 - A note: "Let's create a Python function `is_valid_email` that validates email addresses based on common patterns, and include test cases for various"
 - Buttons: "Accept & Run", "Accept", and "Cancel"
 - A text input field: "What can I help you build?"
 - Information: "Gemini can make mistakes so double-check it and use code with caution. Learn more"
- Bottom Status Bar:**
 - Variables, Terminal
 - Gemini 2.5 Flash
 - 11:59 AM, Python 3