

ASSIGNMENT-8.5

Name: T.Ravali

Hall No:2303A52408

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, there's a sidebar with icons for Commands, Code, Text, Run all, Saving..., RAM, Disk, and other settings. The main area has a title "username validator". Below it is a code editor window titled "Gemini". The code shown is:

```


# 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')


```

A tooltip from Gemini says "#1 (Username Validator – Apply AI in Authentication Context)". A modal dialog box is open, asking "Let's implement the username validator function and its test cases in the selected cell." It has buttons for "Accept & Run", "Accept", and "Cancel". Below the modal is a text input field with "What can I help you build?" and a plus sign. At the bottom of the interface, it says "Gemini can make mistakes so double-check it and use code with caution. Learn more".

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 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 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?'. The overall interface has a light green background.

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 operations like New, Open, Save, and Run. The main area is titled "Task-03" and contains a code block labeled "Gemini". The code is as follows:

```
+# 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 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") == True
+assert is_palindrome("Python") == False
+assert is_palindrome("Was it a car or a cat I saw") == True
```

A tooltip from Gemini says "#3 (Palindrome Checker – Apply AI for String Normalization)". Below the code, there's a message from Gemini: "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". There are three buttons: "Empty cell", "Accept & Run", "Accept", and "Cancel". A text input field at the bottom asks "What can I help you build?". The status bar at the bottom right says "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:

```
+# 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 ▶

```
[5] 0s
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("user@example.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 Python code editor interface with a sidebar containing icons for search, commands, code, text, run all, and cloud storage. The main area has a title "Task-05" and a tab labeled "Gemini". The code editor displays the following Python script:

```
+# 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 top-level domains
+    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")
+assert is_valid_email("user.name+alias@domain.net")
```

A tooltip from Gemini provides instructions for the task:

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 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