

ASSIGNMENT-8.5

Name: P.Arun Reddy

Ht.No:2303A52278

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:

```
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')
```

M #1 (Username Validator – Apply AI in Authentication Context)
Empty cell
Let's implement the username validator function and its test cases in the selected cell.
Accept & Run Accept Cancel

What can I help you build?
+ Gemini 2.5 Flash ▾

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 a code editor window with a Python script named `Gemini`. The script contains a function `classify_input` and some test cases. A tooltip from the Gemini AI is displayed over the code, providing a task description and a help input field.

```
+# 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]) ==
```

What can I help you build?

Accept & Run ✓ Accept X Cancel

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:

```

Task-03
Gemini
# 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("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

```

M #3 (Palindrome Checker – Apply AI for String Normalization)

Empty cell

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

Accept & Run Accept Cancel

What can I help you build?

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:

- `deposit(amount)`

- `withdraw(amount)`

- `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:** In the main workspace, the code for a `BankAccount` class is being generated. The code includes methods for `deposit` and `withdraw`, both of which validate the input amount and raise `ValueError` if it's negative or exceeds the current balance.
- Task Description:** A tooltip provides context for the task: "#4 (BankAccount Class – Apply AI for Object-Oriented Test-Driven Development).
Let's create a `BankAccount` class with methods for `deposit`, `withdraw`, `get_balance`.
Accept & Run Accept Cancel".
- Test Script:** Below the code completion, a test script is shown with several assertions:
 - # Test 10: Negative initial balance
 - # Test 11: Initial balance as float
 - # Test 12: Deposit float amount
 - # Test 13: Withdraw float amount
- Feedback:** A message at the bottom right says "Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)".
- Status Bar:** The status bar at the bottom indicates "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:
 - Must contain @ and .
 - Must not start or end with special characters.
 - 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 window titled "Task-05". The code in the editor is:

```
+# Function
+import re
+
+def is_valid_email(email):
+    if not isinstance(email, str):
+        return False
+    # Regular expression to check email format
+    # This pattern checks
+    # It allows letters, numbers, and underscores
+    # The top-level domain
+    pattern = r"^[a-zA-Z\d_.]+\@[a-zA-Z\d]+\.\w{2,}$"
+    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("invalid_email@.com")
```

An AI interface from Gemini is overlaid on the code editor. The interface has a pink header bar with the text "#5 (Email ID Validation – Apply AI for Data Validation)". Below this is a button labeled "Empty cell". A main text area contains the instruction: "Let's create a Python function `is_valid_email` that validates email addresses based on common patterns, and include test cases for various". At the bottom of this area are three buttons: "Accept & Run", "Accept", and "Cancel". Below the main text area is a smaller box asking "What can I help you build?". The bottom right corner of the interface says "Gemini 2.5 Flash".