

ASSIGNMENT-8.5

Name: T. Bhavana Sri

Hall No:2303A52428

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('a1bc')
+assert is_valid_username('12345')

```

#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 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:
 - o If input is an integer, classify as "Even" or "Odd".
 - o If input is 0, return "Zero".
 - o 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 tooltip box is overlaid on the code, providing instructions: '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, a text input field says 'What can I help you build?'. At the bottom right, it says '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) == "4 is an even integer."
+assert classify_input(7) == "7 is an odd integer."
+assert classify_input("hello") == "hello is not an integer."
+assert classify_input(3.14) == "3.14 is not an integer."
+assert classify_input([1, 2]) == "[1, 2] is not an integer."
```

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
+# 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

+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
```

A tooltip window titled "#3 (Palindrome Checker – Apply AI for String Normalization)" is open over the first few lines of code. It contains the following text:

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.

Accept & Run Accept Cancel

At the bottom of the tooltip, there is a message: "What can I help you build?".

At the very bottom of the editor, a footer bar says "Gemini 2.5 Flash" with a dropdown arrow.

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 a code editor interface with the following details:

- Top Bar:** Search, Commands, Code, Text, Run all, Disk, RAM.
- Title Bar:** Task-04.
- Gemini Panel:** A floating window titled "Gemini" containing:
 - A code snippet of the `BankAccount` class.
 - A message: "#4 (BankAccount Class – Apply AI for Object-Oriented Test-Driven Development)
 - A button: <> Empty cell
 - A note: Let's create a `BankAccount` class with methods for `deposit`, `withdraw`.
 - Buttons: Accept & Run, Accept, Cancel.
 - A text input field: What can I help you build?
 - A button: Gemini 2.5 Flash.
- Code Editor:** A large panel showing a `try:` block with several test cases for the `BankAccount` class. The tests include:
 - # Test 10: Negative initial balance
 - # Test 11: Initial balance as float
 - # Test 12: Deposit float amount
 - # Test 13: Withdraw float amount
- Status Bar:** Shows "15 0s" and "... 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 Gemini AI interface with the title "Task-05". The code editor contains the following Python code:

```
+# 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-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("invalid_email@.com")
```

A tooltip from Gemini provides the following information:

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

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

The interface includes standard Python development tools like a variable browser, terminal, and file manager.