

ASSIGNMENT-8.5

Name: E.Tharun

Ht.No:2303A52352

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 development environment. 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 a string, has at least 5 characters, and starts with a digit. It also includes test cases for 'a' and '1'. A sidebar on the right contains a 'Gemini' interface with a message about applying AI to authentication contexts, a 'Gemini 2.5 Flash' dropdown, and a 'What can I help you build?' input field.

```
username validator

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('a') == False
assert is_valid_username('1') == False
assert is_valid_username('abc') == True
```

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. The code defines a function `classify_input` that checks if a value is an even integer, odd integer, or not an integer. It also includes some test cases. A tooltip from Gemini provides instructions on how to use the AI for generating test cases. Below the code editor, a text input field asks "What can I help you build?".

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

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.

Accept & Run Accept Cancel

What can I help you build?

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:

```

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 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

```

#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 test cases.

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 a code editor window titled "Task-04". At the top, there's a toolbar with icons for Commands, Code, Text, Run all, and a file icon. On the right, there are status indicators for RAM and Disk. The main area has a sidebar with icons for file operations. A central panel displays Python code for a `BankAccount` class with methods `deposit` and `withdraw`. A tooltip from Gemini provides instructions for applying AI to OODT. Below the code, a text input field asks "What can I help you build?" with a "Gemini 2.5 Flash" dropdown. The bottom half of the window contains a test script with several test cases for the `BankAccount` class, including checks for negative initial balance, float amounts, and withdrawal limits. The output shows the tests passing.

```
+# 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
```

```
[5] ✓ Os
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:
 - 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 main pane contains the following Python code:

```

# Gemini
def is_valid_email(email):
    if not isinstance(email, str):
        return False
    # Regular expression
    # 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("info@company.co")

```

A modal dialog box is open in the center of the screen, titled "#5 (Email ID Validation – Apply AI for Data Validation)". It contains the following text:

<> 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

The status bar at the bottom shows "11:59 AM" and "Python 3".