

ASSIGNMENT-8.5

Name:N.ASHOK

Ht.No:2303A52473

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
```

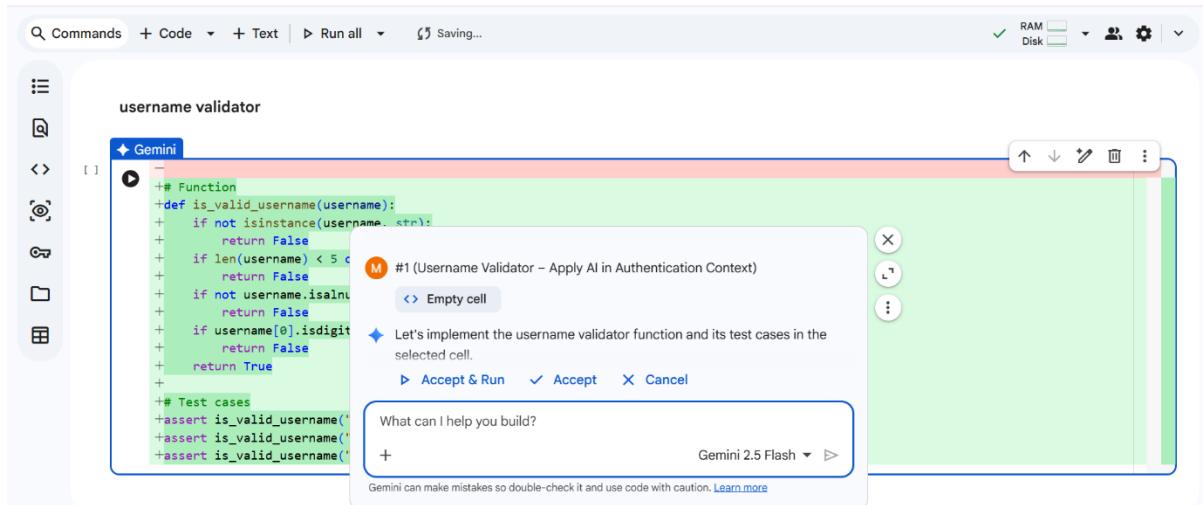
```
assert is_valid_username("12User") == False
```

```
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 a string, has a length of at least 5, and does not start with a digit. It also includes a test case block. A sidebar on the right contains an AI assistant window titled '#1 (Username Validator – Apply AI in Authentication Context)'. The assistant suggests implementing the function and its test cases. It also asks 'What can I help you build?' and provides a 'Gemini 2.5 Flash' button. The interface includes standard file, edit, and search tools at the top.

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:

```

# 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."
```

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:

- 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") ==`

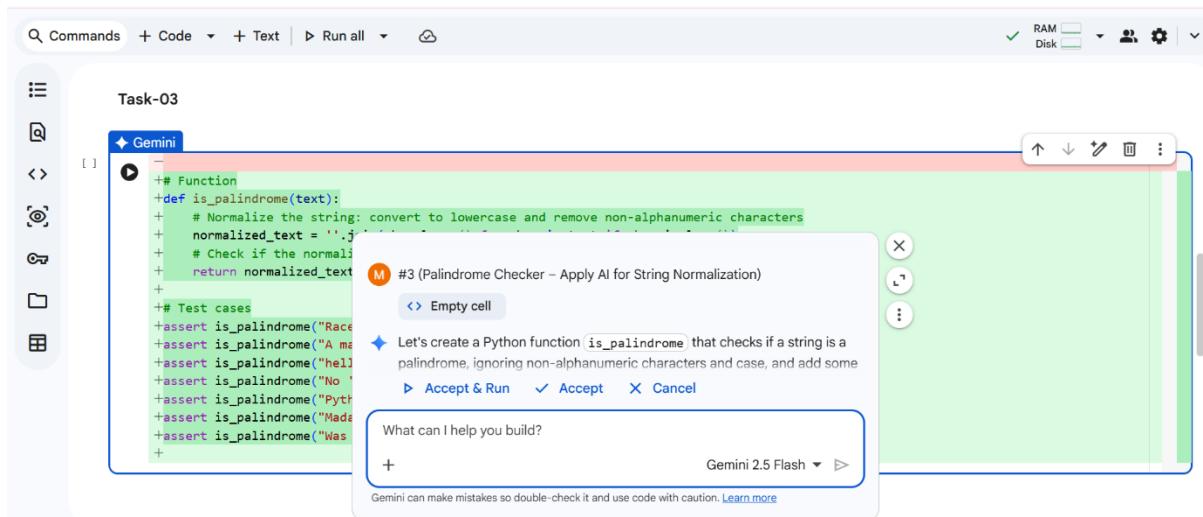
`True`

```
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 window titled "Task-03". The code in the editor is:

```
Gemini
[ ] -
  # Function
  +def is_palindrome(text):
  +    # Normalize the string: convert to lowercase and remove non-alphanumeric characters
  +    normalized_text = ''.join(c.lower() if c.isalnum() else '' for c in text)
  +    # 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" is displayed, titled "#3 (Palindrome Checker – Apply AI for String Normalization)". 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

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

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)

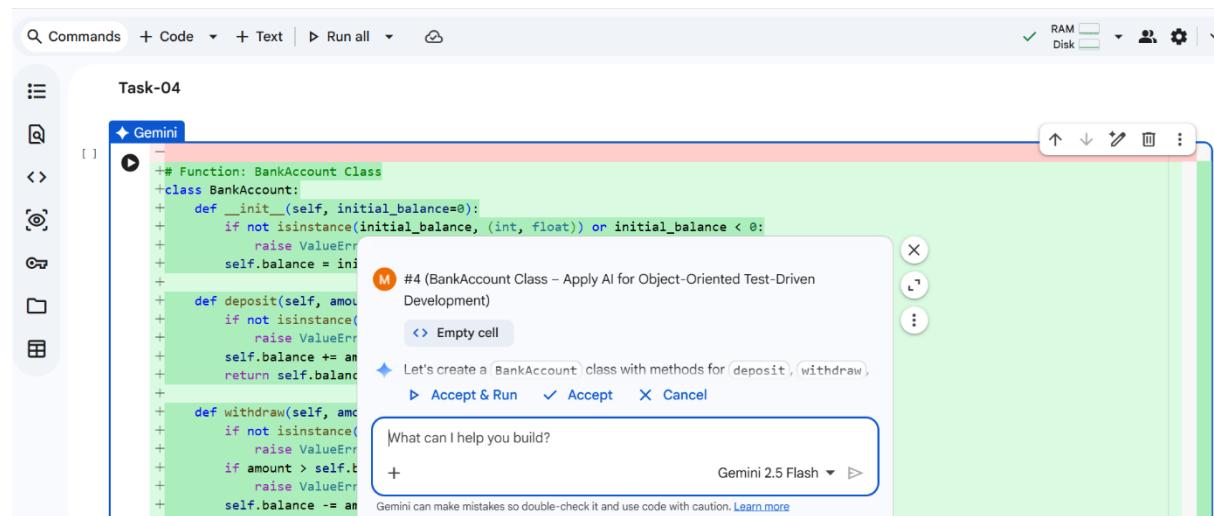
assert acc.get_balance() == 1200

```

Expected Output #4:

- Fully functional class that passes all AI-generated assertions.

Output:



The screenshot shows the Gemini AI interface with the following details:

- Code Editor:** Shows the `BankAccount` class definition and a test script below it.
- Code:**

```

# 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

```
- Toolbox:** On the left, there are icons for Commands, Code, Text, Run all, and a cloud icon.
- Header:** Shows RAM and Disk usage.
- AI Interaction:** A floating window titled "#4 (BankAccount Class – Apply AI for Object-Oriented Test-Driven Development)" contains the following text:
 - Empty cell:** A button to create a new cell.
 - Task:** "Let's create a `BankAccount` class with methods for `deposit`, `withdraw`, `get_balance`, and `__init__`".
 - Buttons:** Accept & Run, Accept, Cancel.
 - Input:** "What can I help you build?" with a text input field containing a plus sign (+).
 - Feedback:** "Gemini 2.5 Flash" and a "Learn more" link.
- Test Script:** The bottom half of the interface shows a test script with the following content:

```

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!")

```
- Output:** The test script ends with the message "... 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:

Task-05

```
◆ Gemini
[ ] #!/usr/bin/env python3
# Function
import re
+
def is_valid_email(email):
    if not isinstance(email, str):
        return False
    # Regular expression for validating an Email
    # This pattern checks
    # 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")
assert is_valid_email("john.doe@example.org")
assert is_valid_email("info@company.co.uk")
 Empty cell
```

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](#)

Variables Terminal 11:59 AM Python 3