

ASSIGNMENT-8.5

Name: G.Vignesh

HT.No:2303A52359

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:
 - o Username length must be between 5 and 15 characters.
 - o Must contain only alphabets and digits.
 - o Must not start with a digit.
 - o No spaces allowed. Example Assert

Test Cases:

```
assert is_valid_username("User123") == True
```

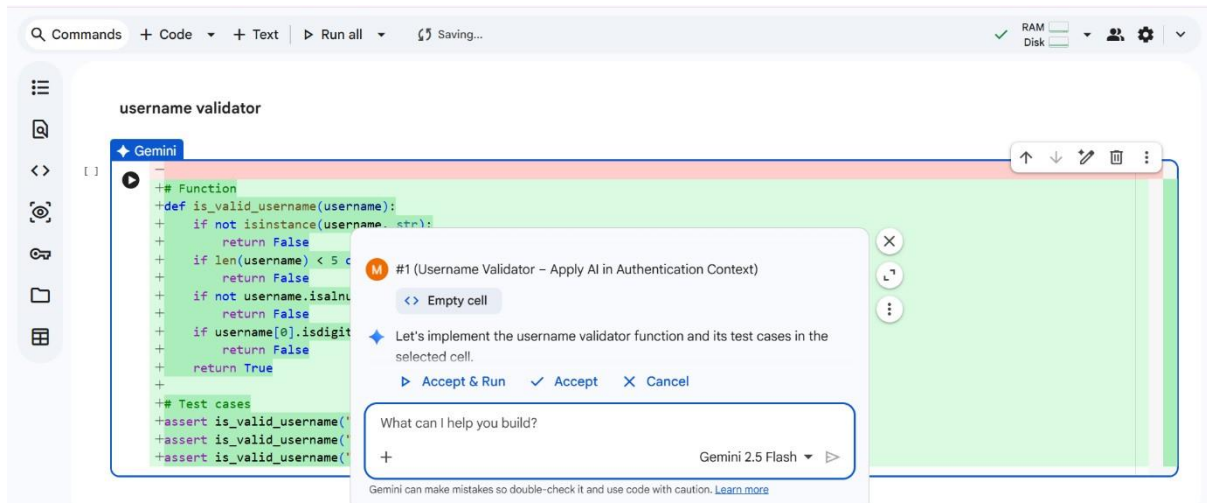
```
assert is_valid_username("12User") == False
```

```
assert is_valid_username("Us er") == False
```

Expected Output #1:

- Username validation logic successfully passing all AI-generated test cases.

Output:



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:

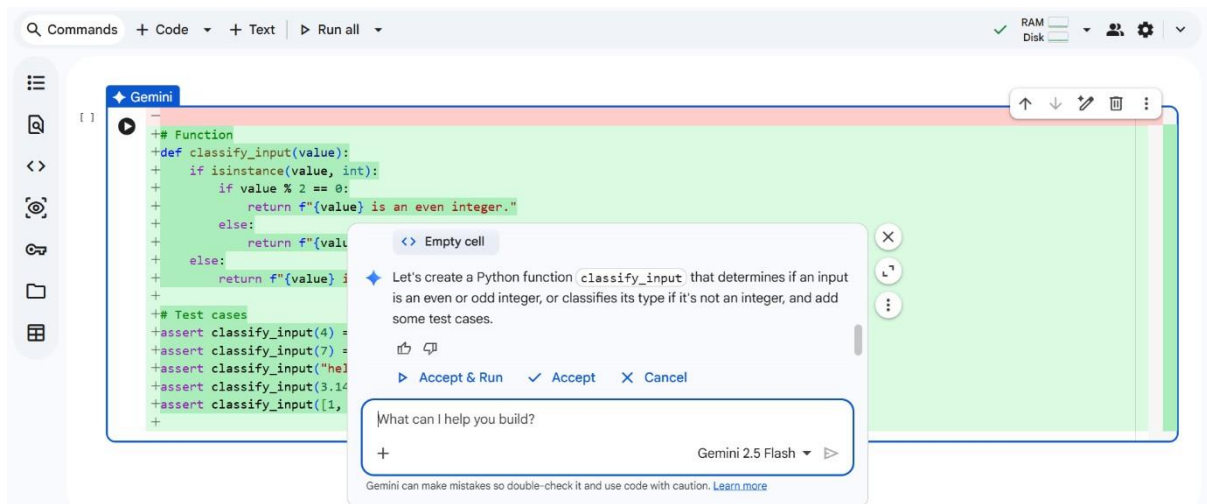
`assert classify_value(8) == "Even"` `assert`
`classify_value(7) == "Odd"` `assert`

```
classify_value("abc") == "Invalid Input"
```

Expected Output #2:

- Function correctly classifying values and passing all test cases.

Output:



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
```

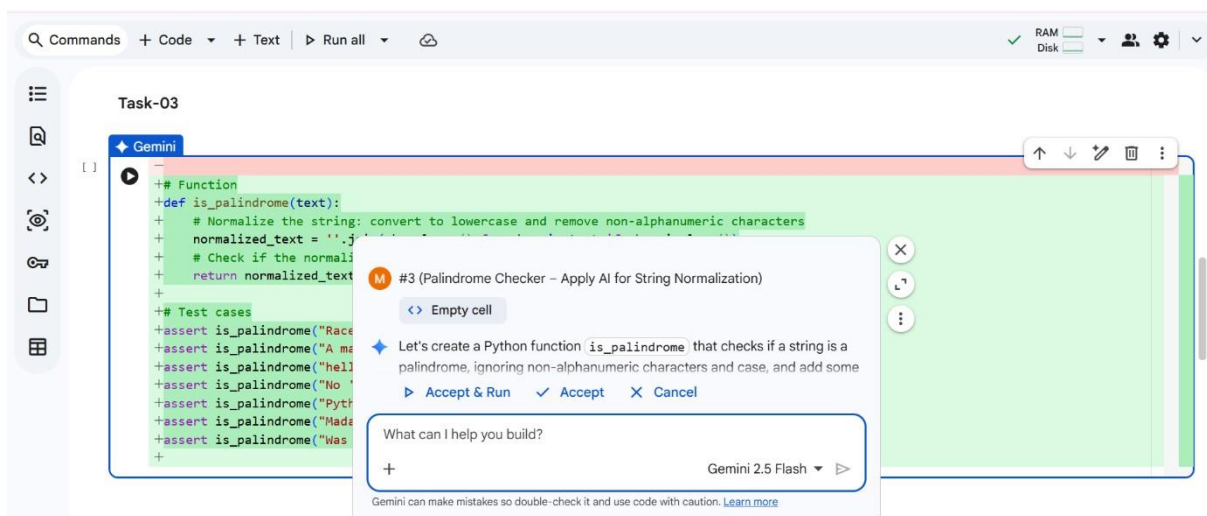
```
assert is_palindrome("A man a plan a canal Panama") ==  
True
```

```
assert is_palindrome("Python") == False Expected
```

Output #3:

- Function correctly identifying palindromes and passing all AI-generated tests.

Output:



Task 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 with a file named 'Task-04'. The code defines a `BankAccount` class with the following methods:

- `__init__(self, initial_balance=0)`: Initializes the account with a balance. It raises a `ValueError` if the initial balance is not an integer or float, or if it is less than 0.
- `deposit(self, amount)`: Adds the specified amount to the account balance. It raises a `ValueError` if the amount is not an integer or float.
- `withdraw(self, amount)`: Subtracts the specified amount from the account balance. It raises a `ValueError` if the amount is not an integer or float, or if it is greater than the current balance.

A Gemini AI chat window is open on the right side of the editor. It shows a prompt: '#4 (BankAccount Class - Apply AI for Object-Oriented Test-Driven Development)' and a response: 'Let's create a BankAccount class with methods for deposit, withdraw, ...'. The chat window also includes a search bar and a 'Gemini 2.5 Flash' dropdown menu.

The screenshot shows a terminal window with the following output:

```
[5] 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!")
```

At the bottom of the terminal, it says: '... 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:

CommandsCodeTextRun all

RAMDisk

Task-05

Gemini

```
+ # Function
+ import re
+
+ def is_valid_email(email):
+     if not isinstance(email, str):
+         return False
+     # Regular expression for email validation
+     # This pattern checks:
+     # 1. It allows letters, digits, and underscores in the local part.
+     # 2. It allows a period in the local part.
+     # 3. It allows hyphens in the local part.
+     # 4. It allows a plus sign in the local part.
+     # 5. It allows a top-level domain of 2 to 6 characters.
+     pattern = r"^[a-zA-Z0-9_+.-]+@([a-zA-Z0-9-]+\.)+[a-zA-Z]{2,6}$"
+     if re.fullmatch(pattern, email):
+         return True
+     else:
+         return False
+
+ # Test cases
+ assert is_valid_email("test@example.com") == True
+ assert is_valid_email("john.doe@company.org") == True
+ assert is_valid_email("invalid_email@") == False
+ assert is_valid_email("123@domain.co.uk") == True
+ assert is_valid_email("user@domain") == False
```

#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 & RunAcceptCancel

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](#)

VariablesTerminal

11:59 AMPython 3