ASSIGNMENT-8.5

Name: K.MANOJ KUMAR

Hall No:2303A52201

Batch:35

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.

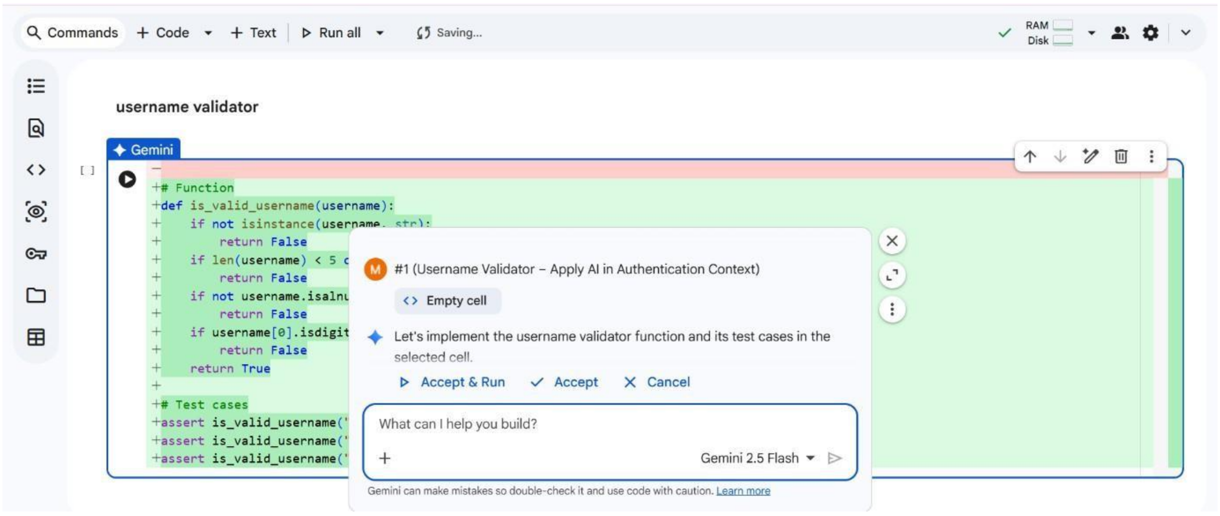
Example Assert Test Cases:

```
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:



username validator

```
-
+# Function
+def is_valid_username(username):
+    if not isinstance(username, str):
+        return False
+    if len(username) < 5 c
+        return False
+    if not username.isalnu
+        return False
+    if username[0].isdigit
+        return False
+    return True
+
+# Test cases
+assert is_valid_username('
+assert is_valid_username('
+assert is_valid_username('
```

#1 (Username Validator – Apply AI in Authentication Context)

Let's implement the username validator function and its test cases in the selected cell.

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:

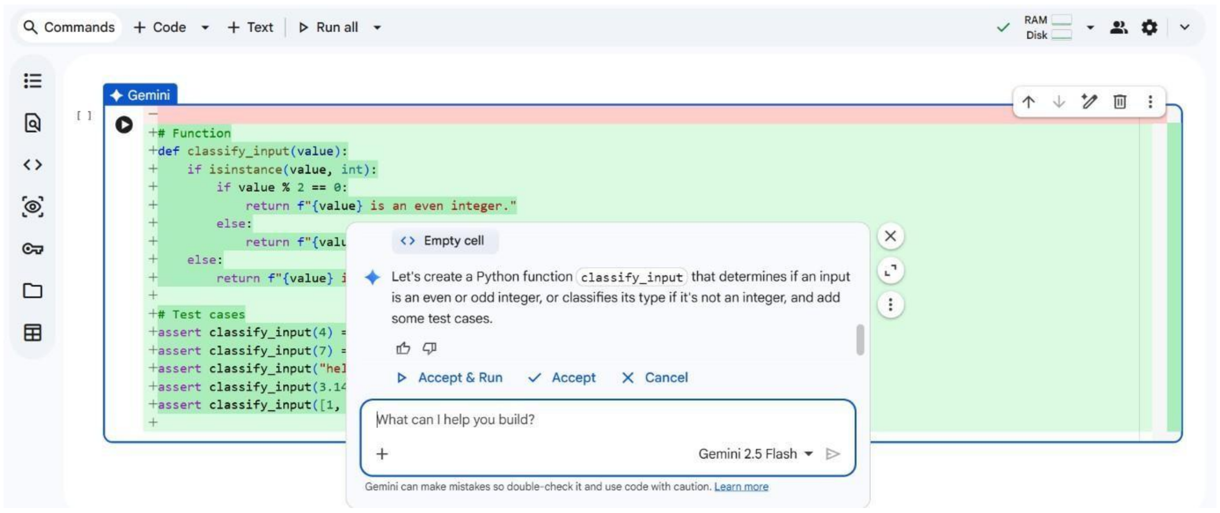
o If input is an integer, classify as "Even" or "Odd".

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:



```
-
+# Function
+def classify_input(value):
+    if isinstance(value, int):
+        if value % 2 == 0:
+            return f"{value} is an even integer."
+        else:
+            return f"{valu
+    else:
+        return f"{value} i
+
+# Test cases
+assert classify_input(4) =
+assert classify_input(7) =
+assert classify_input("hel
+assert classify_input(3.14
+assert classify_input([1,
+
```

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.

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

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

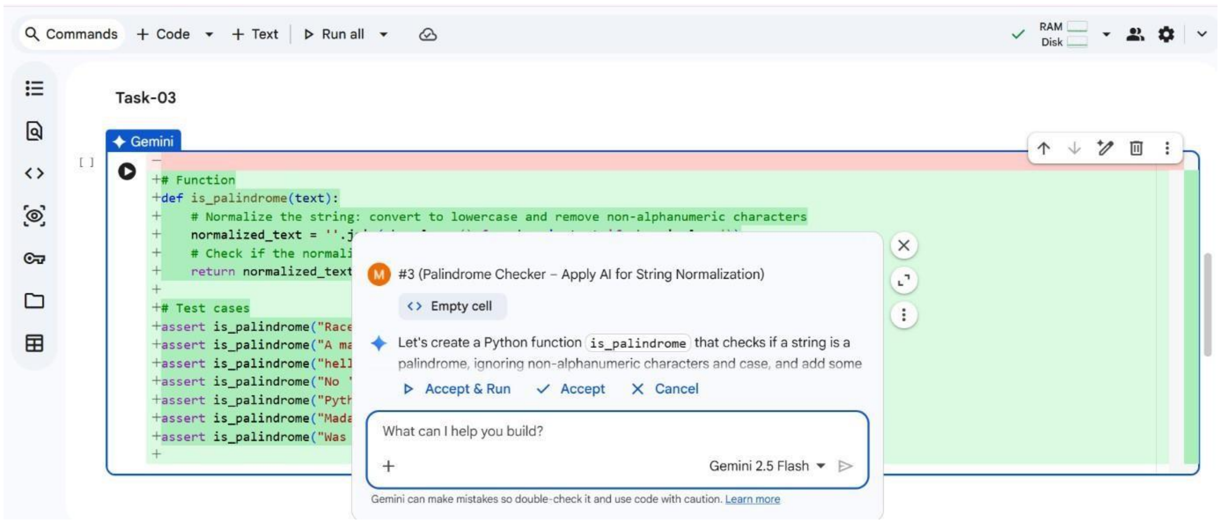
assert is_palindrome("A man a plan a canal Panama") == True

assert is_palindrome("Python") == False Expected

Output #3:

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

Output:



Task-03

```
-
+# Function
+def is_palindrome(text):
+    # Normalize the string: convert to lowercase and remove non-alphanumeric characters
+    normalized_text = ''.j
+    # Check if the normali
+    return normalized_text
+
+# Test cases
+assert is_palindrome("Race
+assert is_palindrome("A ma
+assert is_palindrome("hell
+assert is_palindrome("No '
+assert is_palindrome("Pyth
+assert is_palindrome("Mada
+assert is_palindrome("Was
+
```

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

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

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)

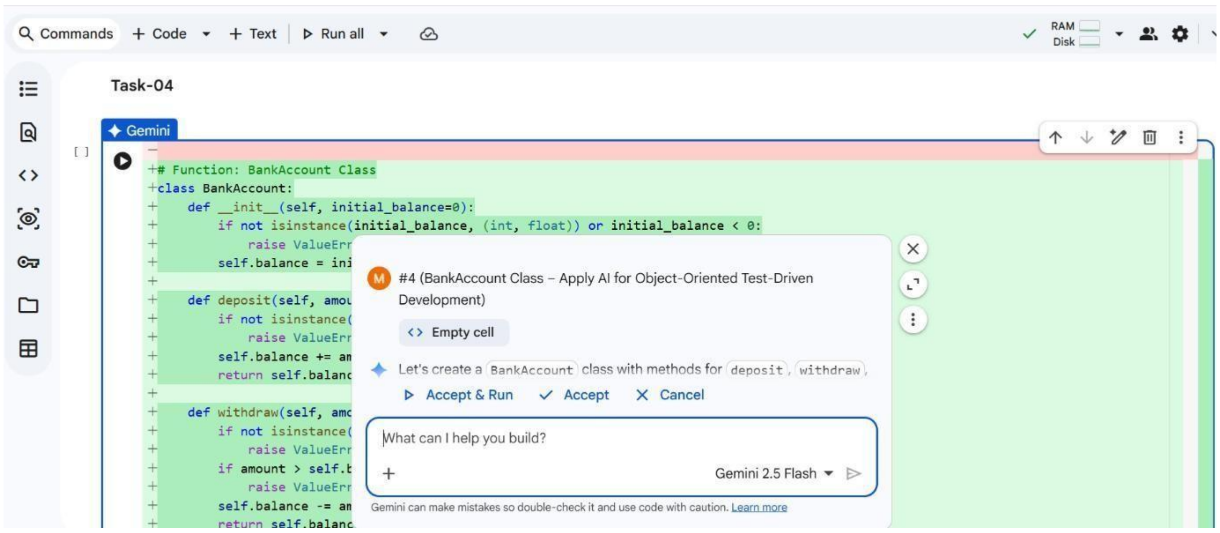
assert acc.get_balance() == 1500 acc.withdraw(300)

assert acc.get_balance() == 1200 Expected

Output #4:

- Fully functional class that passes all AIgenerated assertions.

Output:



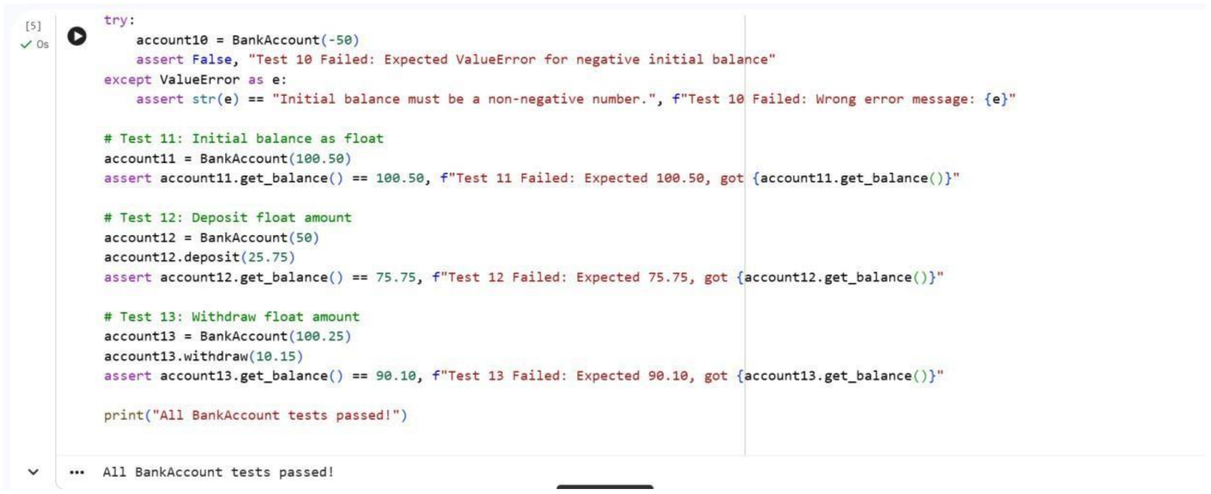
```
-
+# Function: BankAccount Class
+class BankAccount:
+    def __init__(self, initial_balance=0):
+        if not isinstance(initial_balance, (int, float)) or initial_balance < 0:
+            raise ValueErr
+        self.balance = ini
+
+    def deposit(self, amou
+        if not isinstance(
+            raise ValueErr
+        self.balance += am
+        return self.balanc
+
+    def withdraw(self, amc
+        if not isinstance(
+            raise ValueErr
+        if amount > self.b
+            raise ValueErr
+        self.balance -= am
+        return self.balanc
```

#4 (BankAccount Class – Apply AI for Object-Oriented Test-Driven Development)

Empty cell

Let's create a `BankAccount` class with methods for `deposit`, `withdraw`,

Gemini can make mistakes so double-check it and use code with caution. Learn more



```
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:

o Must contain @ and . o Must not start or end with special characters.

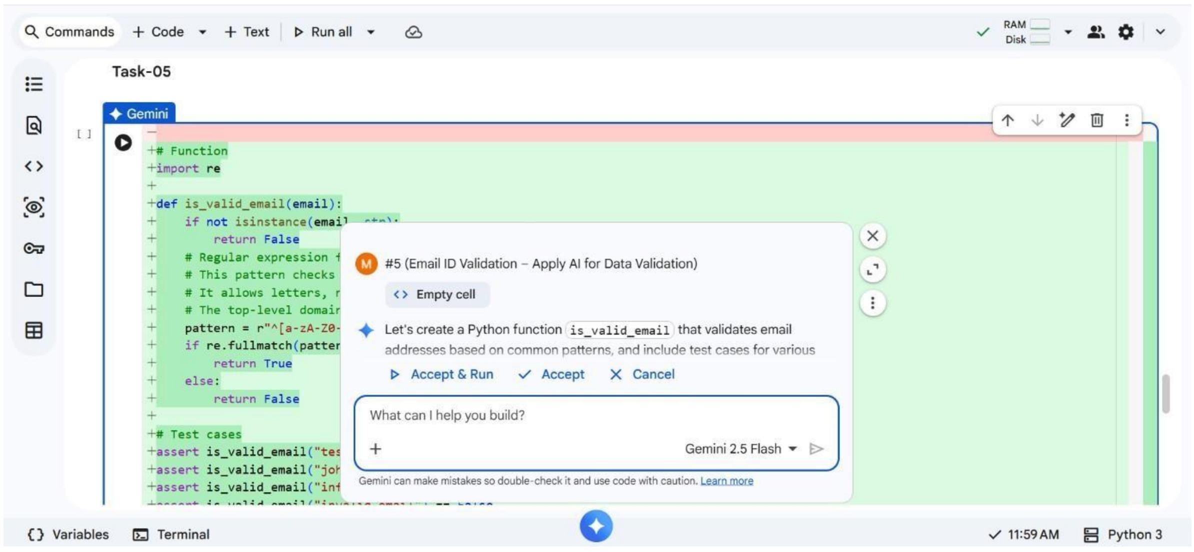
o Should handle invalid formats gracefully. Example Assert Test Cases:

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:



Task-05

```
-
+# Function
+import re
+
+def is_valid_email(email):
+    if not isinstance(email
+        return False
+    # Regular expression f
+    # This pattern checks
+    # It allows letters, r
+    # The top-level domair
+    pattern = r"^[a-zA-Z0-
+    if re.fullmatch(patter
+        return True
+    else:
+        return False
+
+# Test cases
+assert is_valid_email("tes
+assert is_valid_email("joh
+assert is_valid_email("inf
```

#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

Gemini can make mistakes so double-check it and use code with caution. Learn more