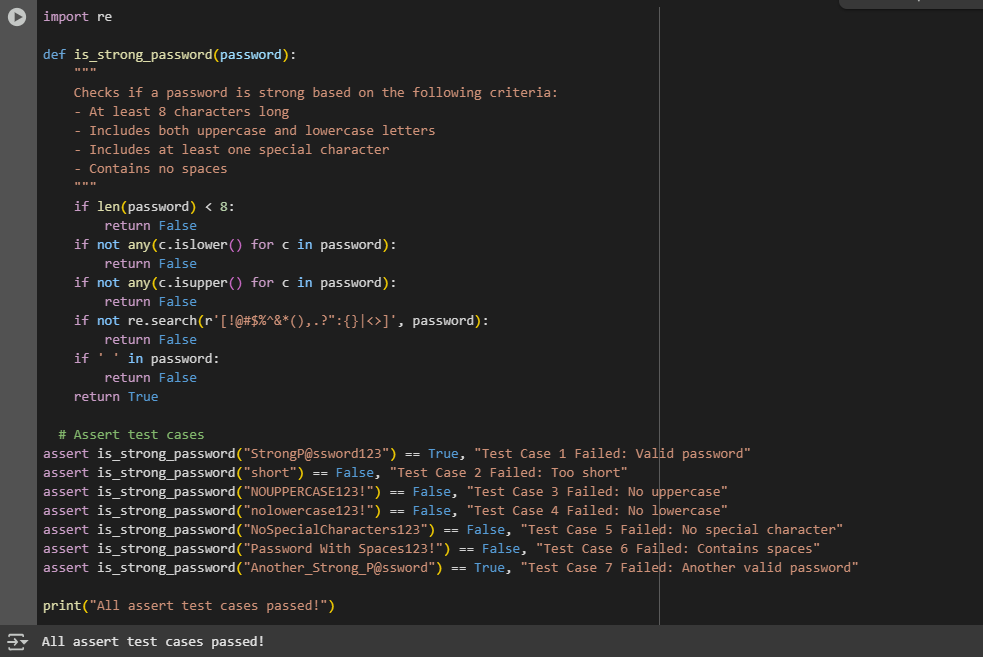
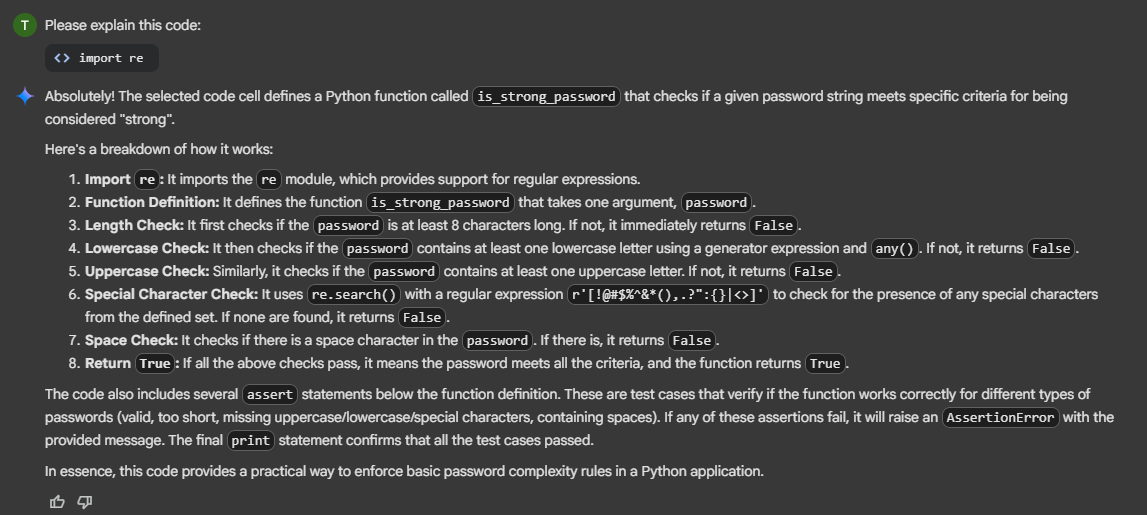
**AI ASSISTED CODING – 8**

**TASK-1:**

(Password Strength Validator – Apply AI in  
Security Context)  
• Task: Apply AI to generate at least 3 assert test cases for  
is\_strong\_password(password) and implement the validator  
function.  
• Requirements:  
o Password must have at least 8 characters.  
o Must include uppercase, lowercase, digit, and special  
character.  
o Must not contain spaces.  
Example Assert Test Cases:  
assert is\_strong\_password("Abcd@123") == True  
assert is\_strong\_password("abcd123") == False  
assert is\_strong\_password("ABCD@1234") == True

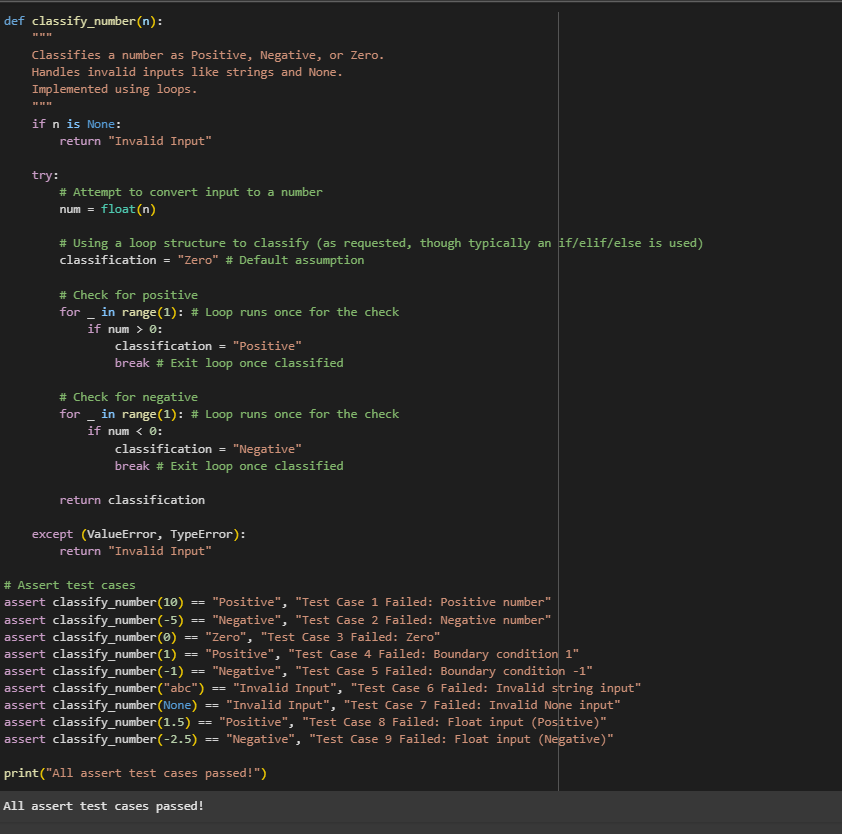
****

**EXPLANATION:**

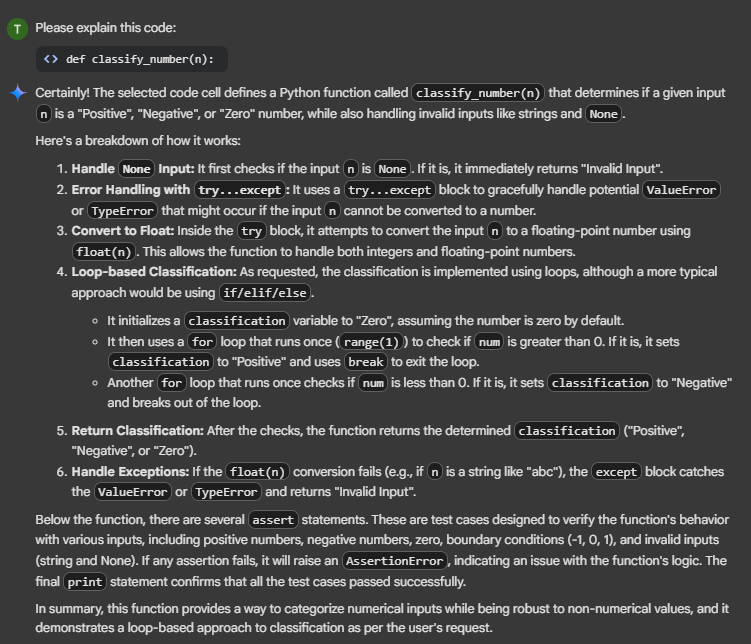
****

**TASK-2:**

(Number Classification with Loops – Apply AI for  
Edge Case Handling)  
• Task: Use AI to generate at least 3 assert test cases for a  
classify\_number(n) function. Implement using loops.  
• Requirements:  
o Classify numbers as Positive, Negative, or Zero.  
o Handle invalid inputs like strings and None o Include boundary conditions (-1, 0, 1).  
Example Assert Test Cases:  
assert classify\_number(10) == "Positive"  
assert classify\_number(-5) == "Negative"  
assert classify\_number(0) == "Zero"

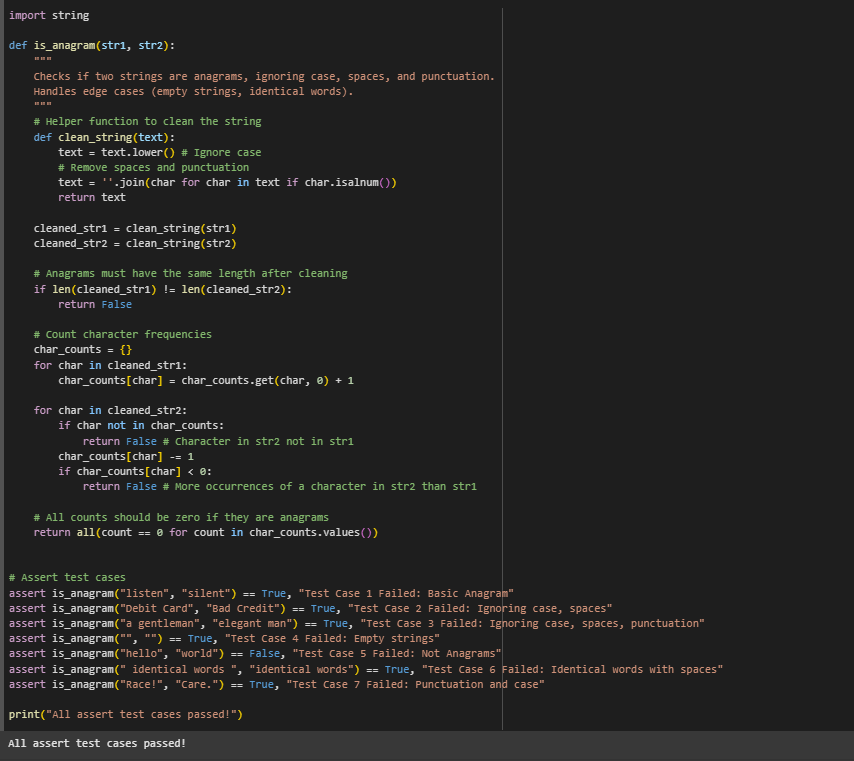
****

**EXPLANATION:**

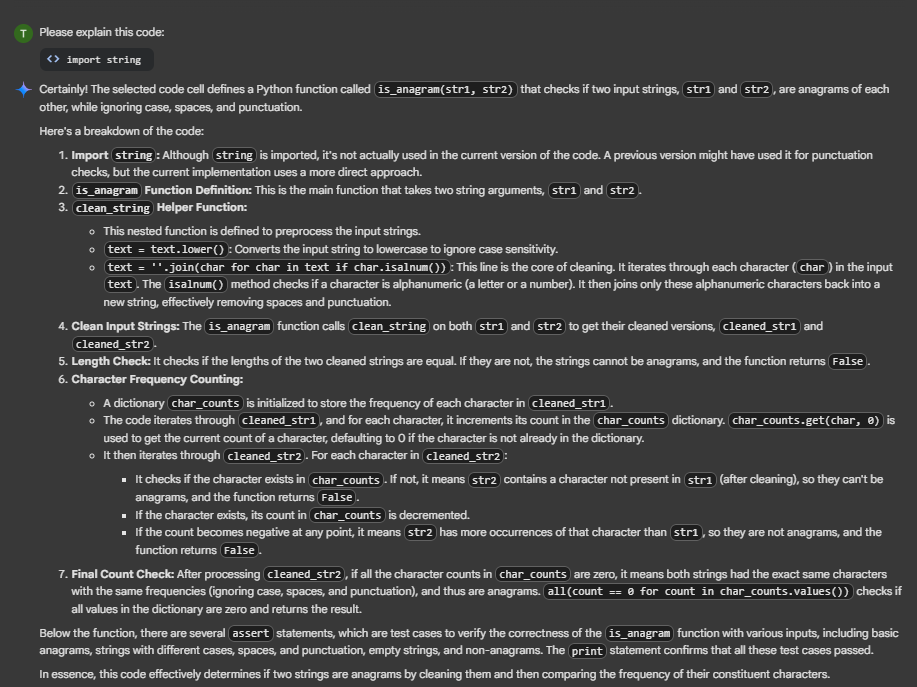
****

**TASK-3:**

(Anagram Checker – Apply AI for String Analysis)  
• Task: Use AI to generate at least 3 assert test cases for  
is\_anagram(str1, str2) and implement the function.  
• Requirements:  
o Ignore case, spaces, and punctuation.  
o Handle edge cases (empty strings, identical words).  
Example Assert Test Cases:  
assert is\_anagram("listen", "silent") == True  
assert is\_anagram("hello", "world") == False  
assert is\_anagram("Dormitory", "Dirty Room") == True

****

**EXPLANATION:**

****

**TASK-4:**

(Inventory Class – Apply AI to Simulate Real-  
World Inventory System)  
• Task: Ask AI to generate at least 3 assert-based tests for an  
Inventory class with stock management.  
• Methods:  
o add\_item(name, quantity)  
o remove\_item(name, quantity)  
o get\_stock(name)  
Example Assert Test Cases:  
inv = Inventory()  
inv.add\_item("Pen", 10)  
assert inv.get\_stock("Pen") == 10  
inv.remove\_item("Pen", 5)  
assert inv.get\_stock("Pen") == 5  
inv.add\_item("Book", 3)  
assert inv.get\_stock("Book") == 3

class Inventory:

    """

    A simple Inventory class for managing stock of items.

    """

    def \_\_init\_\_(self):

        self.stock = {}

    def add\_item(self, name, quantity):

        """

        Adds a specified quantity of an item to the inventory.

        If the item already exists, its quantity is updated.

        If the item does not exist, it is added to the inventory.

        Handles cases where quantity is non-positive.

        """

        if quantity <= 0:

            print(f"Warning: Quantity must be positive to add item '{name}'.")

            return

        self.stock[name] = self.stock.get(name, 0) + quantity

        print(f"Added {quantity} of {name}. Current stock: {self.stock[name]}")

    def remove\_item(self, name, quantity):

        """

        Removes a specified quantity of an item from the inventory.

        If the item does not exist or the quantity to remove is more than available,

        it prints a warning and does not change the stock.

        Handles cases where quantity is non-positive.

        """

        if quantity <= 0:

            print(f"Warning: Quantity must be positive to remove item '{name}'.")

            return

        if name not in self.stock:

            print(f"Warning: Item '{name}' not found in inventory.")

            return

        if self.stock[name] < quantity:

            print(f"Warning: Not enough stock of '{name}' to remove {quantity}. Available: {self.stock[name]}")

            return

        self.stock[name] -= quantity

        print(f"Removed {quantity} of {name}. Current stock: {self.stock[name]}")

        if self.stock[name] == 0:

            del self.stock[name]

            print(f"Item '{name}' stock reached zero and removed from inventory.")

    def get\_stock(self, name):

        """

        Returns the current stock quantity of a specific item.

        Returns 0 if the item is not found in the inventory.

        """

        return self.stock.get(name, 0)

# Assert test cases

# Test Case 1: Adding items and checking stock

inventory = Inventory()

inventory.add\_item("apple", 10)

assert inventory.get\_stock("apple") == 10, "Test Case 1 Failed: Add item and check stock"

# Test Case 2: Adding more of an existing item

inventory.add\_item("apple", 5)

assert inventory.get\_stock("apple") == 15, "Test Case 2 Failed: Add more of existing item"

# Test Case 3: Adding a new item

inventory.add\_item("banana", 20)

assert inventory.get\_stock("banana") == 20, "Test Case 3 Failed: Add new item"

# Test Case 4: Removing some stock

inventory.remove\_item("apple", 7)

assert inventory.get\_stock("apple") == 8, "Test Case 4 Failed: Remove some stock"

# Test Case 5: Removing all stock of an item

inventory.remove\_item("apple", 8)

assert inventory.get\_stock("apple") == 0, "Test Case 5 Failed: Remove all stock"

assert "apple" not in inventory.stock, "Test Case 5 Failed: Item not removed from stock dictionary"

# Test Case 6: Attempting to remove more stock than available

inventory.remove\_item("banana", 25) # Should print a warning, stock should remain 20

assert inventory.get\_stock("banana") == 20, "Test Case 6 Failed: Remove more than available"

# Test Case 7: Attempting to remove a non-existent item

inventory.remove\_item("grape", 5) # Should print a warning, stock should remain 0

assert inventory.get\_stock("grape") == 0, "Test Case 7 Failed: Remove non-existent item"

# Test Case 8: Attempting to add with non-positive quantity

inventory.add\_item("orange", 0) # Should print a warning, stock should remain 0

assert inventory.get\_stock("orange") == 0, "Test Case 8 Failed: Add with non-positive quantity"

# Test Case 9: Attempting to remove with non-positive quantity

inventory.remove\_item("banana", 0) # Should print a warning, stock should remain 20

assert inventory.get\_stock("banana") == 20, "Test Case 9 Failed: Remove with non-positive quantity"

print("All assert test cases passed!")

Added 10 of apple. Current stock: 10

Added 5 of apple. Current stock: 15

Added 20 of banana. Current stock: 20

Removed 7 of apple. Current stock: 8

Removed 8 of apple. Current stock: 0

Item 'apple' stock reached zero and removed from inventory.

Warning: Not enough stock of 'banana' to remove 25. Available: 20

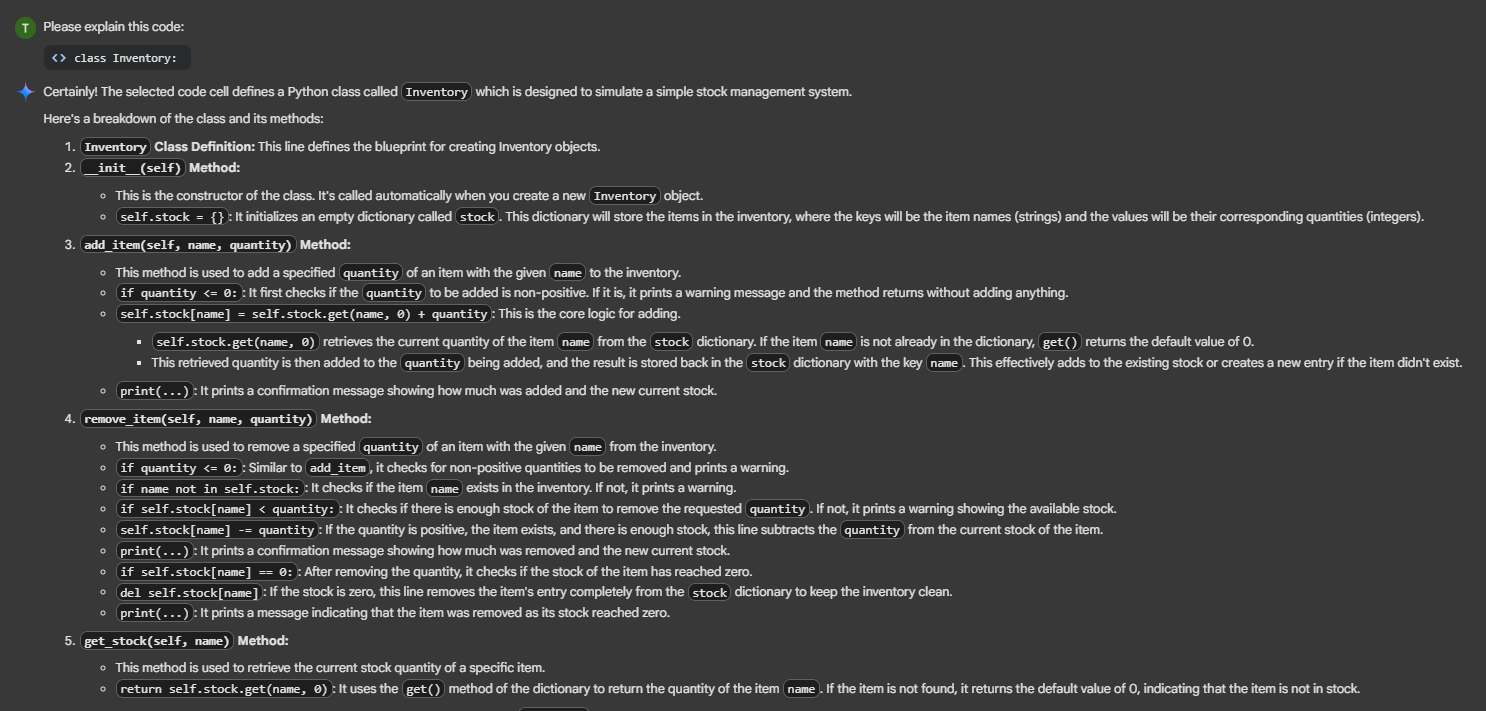
Warning: Item 'grape' not found in inventory.

Warning: Quantity must be positive to add item 'orange'.

Warning: Quantity must be positive to remove item 'banana'.

All assert test cases passed!

**EXPLANATION:**

****

**TASK-5:**

(Date Validation & Formatting – Apply AI for  
Data Validation)  
• Task: Use AI to generate at least 3 assert test cases for  
validate\_and\_format\_date(date\_str) to check and convert dates.  
• Requirements:  
o Validate "MM/DD/YYYY" format.  
o Handle invalid dates.  
o Convert valid dates to "YYYY-MM-DD".  
Example Assert Test Cases:  
assert validate\_and\_format\_date("10/15/2023") == "2023-10-15"  
assert validate\_and\_format\_date("02/30/2023") == "Invalid Date"  
assert validate\_and\_format\_date("01/01/2024") == "2024-01-01"

from datetime import datetime

def validate\_and\_format\_date(date\_str):

    """

    Validates a date string in "MM/DD/YYYY" format and converts it to "YYYY-MM-DD".

    Handles invalid date formats or invalid dates.

    """

    try:

        # Attempt to parse the date string in the expected format

        date\_obj = datetime.strptime(date\_str, "%m/%d/%Y")

        # Format the date object into the desired output format

        return date\_obj.strftime("%Y-%m-%d")

    except (ValueError, TypeError):

        # If parsing fails (invalid format or invalid date), return an error message

        return "Invalid Date"

# Assert test cases

assert validate\_and\_format\_date("12/25/2023") == "2023-12-25", "Test Case 1 Failed: Valid date"

assert validate\_and\_format\_date("01/01/2024") == "2024-01-01", "Test Case 2 Failed: Valid date"

assert validate\_and\_format\_date("13/01/2023") == "Invalid Date", "Test Case 3 Failed: Invalid month"

assert validate\_and\_format\_date("12/32/2023") == "Invalid Date", "Test Case 4 Failed: Invalid day"

assert validate\_and\_format\_date("12/25/23") == "Invalid Date", "Test Case 5 Failed: Invalid year format"

assert validate\_and\_format\_date("2023/12/25") == "Invalid Date", "Test Case 6 Failed: Invalid format"

assert validate\_and\_format\_date("abc") == "Invalid Date", "Test Case 7 Failed: Non-date string"

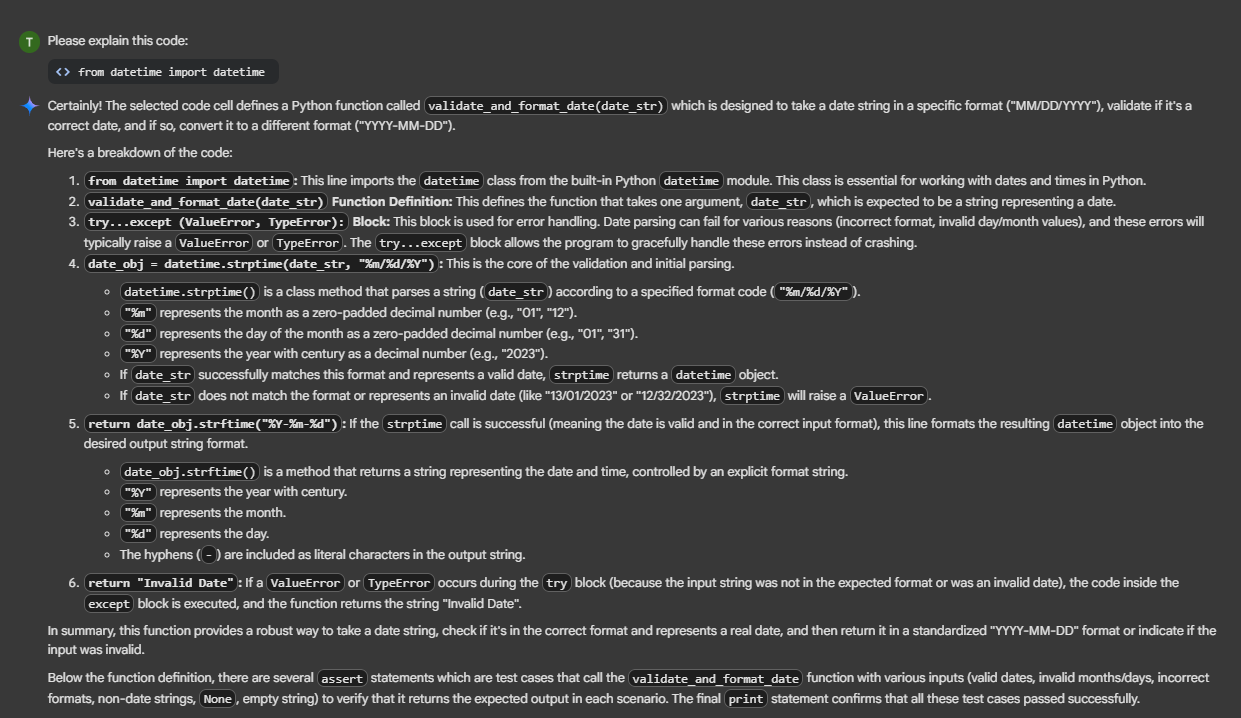
assert validate\_and\_format\_date(None) == "Invalid Date", "Test Case 8 Failed: None input"

assert validate\_and\_format\_date("") == "Invalid Date", "Test Case 9 Failed: Empty string"

print("All assert test cases passed!")

All assert test cases passed!

**EXPLANATION:**

****