AI ASSIGNMENT 8.1

HTNO:2403A51284

BATCH:12

Task Description #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
Expected Output #1:
• Password validation logic passing all AI-generated test cases

PROMPT:

Implement the function is_strong_password(password) to validate password strength with the following requirements:

At least 8 characters long

Includes at least one uppercase letter, one lowercase letter, one digit, and one special character

Must not contain spaces

Use AI to generate at least 3 assert test cases for the function.

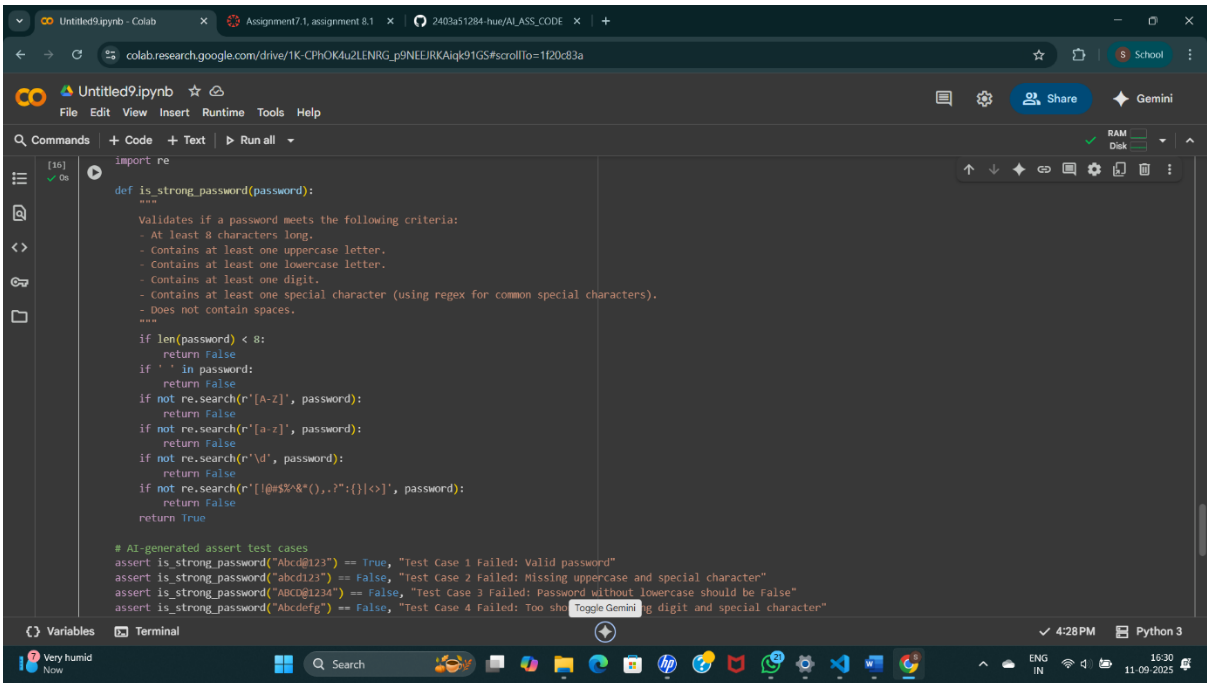
Implement the password validator function.

Ensure the function passes all AI-generated test cases.

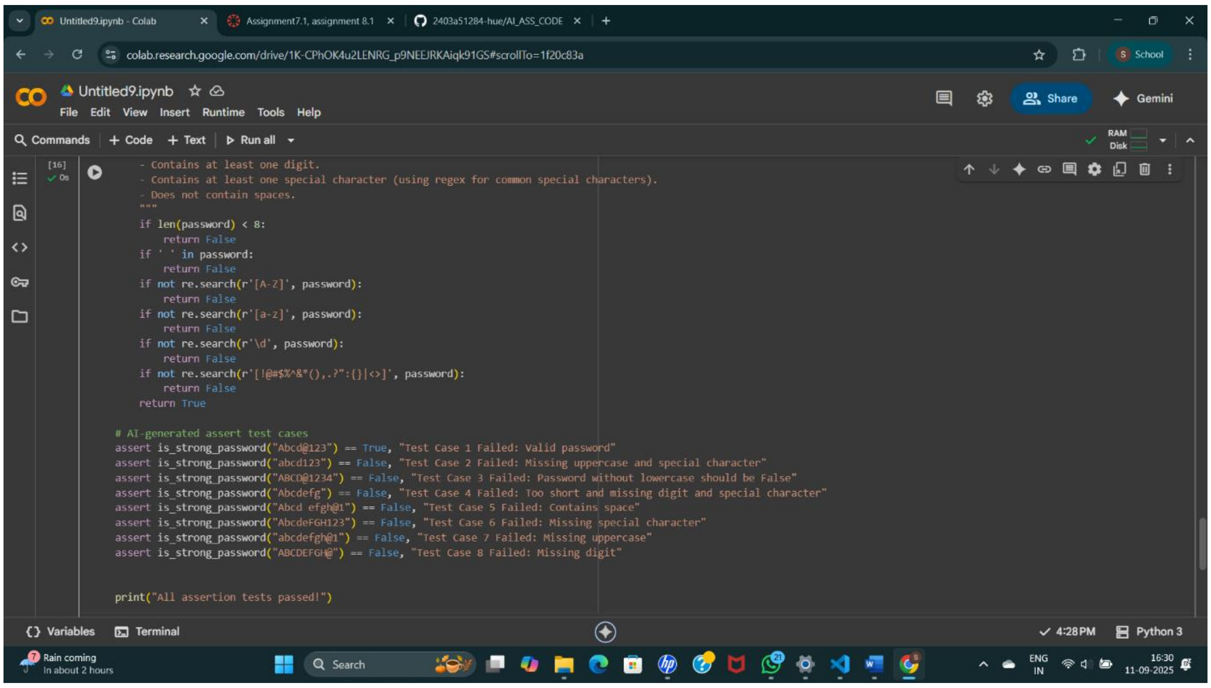
Use loops if needed for input validation or processing.

Use AI to generate at least 3 assert test cases.

CODE:

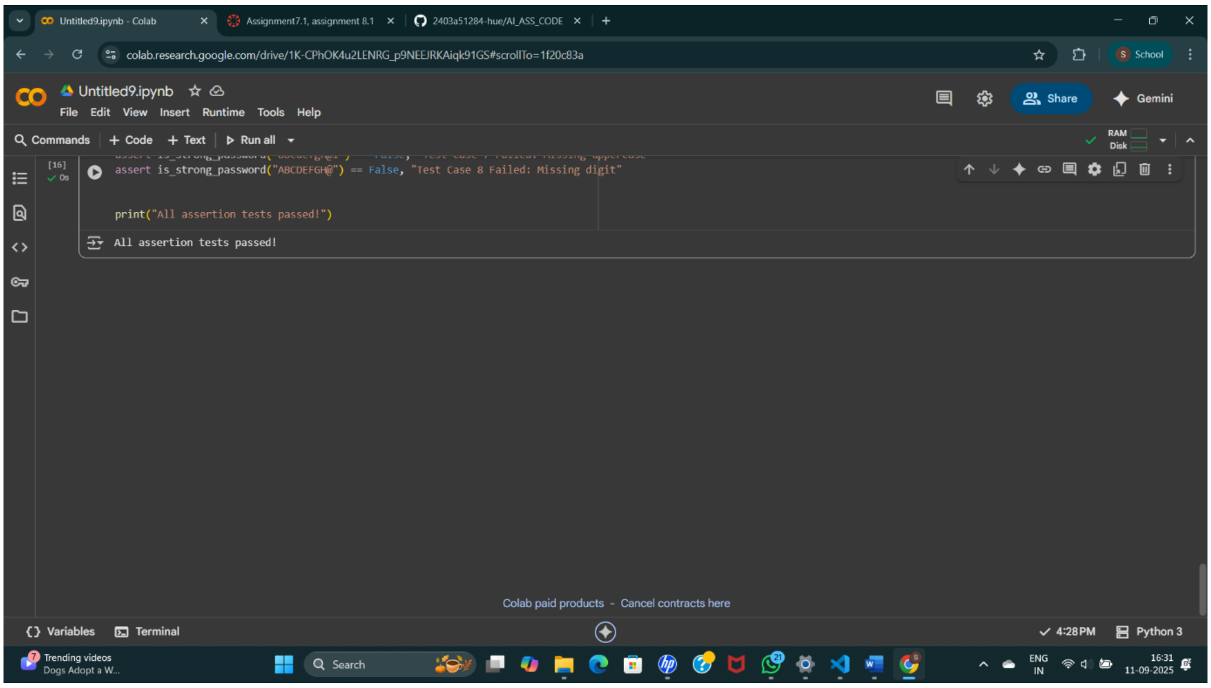


```python
import re

def is_strong_password(password):
    """
    Validates if a password meets the following criteria:
    - At least 8 characters long.
    - Contains at least one uppercase letter.
    - Contains at least one lowercase letter.
    - Contains at least one digit.
    - Contains at least one special character (using regex for common special characters).
    - Does not contain spaces.
    """
    if len(password) < 8:
        return False
    if ' ' in password:
        return False
    if not re.search(r'[A-Z]', password):
        return False
    if not re.search(r'[a-z]', password):
        return False
    if not re.search(r'\d', password):
        return False
    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password):
        return False
    return True

# AI-generated assert test cases
assert is_strong_password("Abcd@123") == True, "Test Case 1 Failed: Valid password"
assert is_strong_password("abcd123") == False, "Test Case 2 Failed: Missing uppercase and special character"
assert is_strong_password("ABCD@1234") == False, "Test Case 3 Failed: Password without lowercase should be False"
assert is_strong_password("Abcdefg") == False, "Test Case 4 Failed: Too sho Toggle Gemini g digit and special character"
```



```python
    - Contains at least one digit.
    - Contains at least one special character (using regex for common special characters).
    - Does not contain spaces.
    """
    if len(password) < 8:
        return False
    if ' ' in password:
        return False
    if not re.search(r'[A-Z]', password):
        return False
    if not re.search(r'[a-z]', password):
        return False
    if not re.search(r'\d', password):
        return False
    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password):
        return False
    return True

# AI-generated assert test cases
assert is_strong_password("Abcd@123") == True, "Test Case 1 Failed: Valid password"
assert is_strong_password("abcd123") == False, "Test Case 2 Failed: Missing uppercase and special character"
assert is_strong_password("ABCD@1234") == False, "Test Case 3 Failed: Password without lowercase should be False"
assert is_strong_password("Abcdefg") == False, "Test Case 4 Failed: Too short and missing digit and special character"
assert is_strong_password("Abcd efgh@1") == False, "Test Case 5 Failed: Contains space"
assert is_strong_password("AbcdeFGH123") == False, "Test Case 6 Failed: Missing special character"
assert is_strong_password("abcdefgh@1") == False, "Test Case 7 Failed: Missing uppercase"
assert is_strong_password("ABCDEFGH@") == False, "Test Case 8 Failed: Missing digit"


print("All assertion tests passed!")
```

OUTPUT:



```
assert is_strong_password("ABCDEFGH@") == False, "Test Case 8 Failed: Missing digit"


print("All assertion tests passed!")
```

```
All assertion tests passed!
```

Task Description #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"
Expected Output #2:
• Classification logic passing all assert tests

PROMPT:

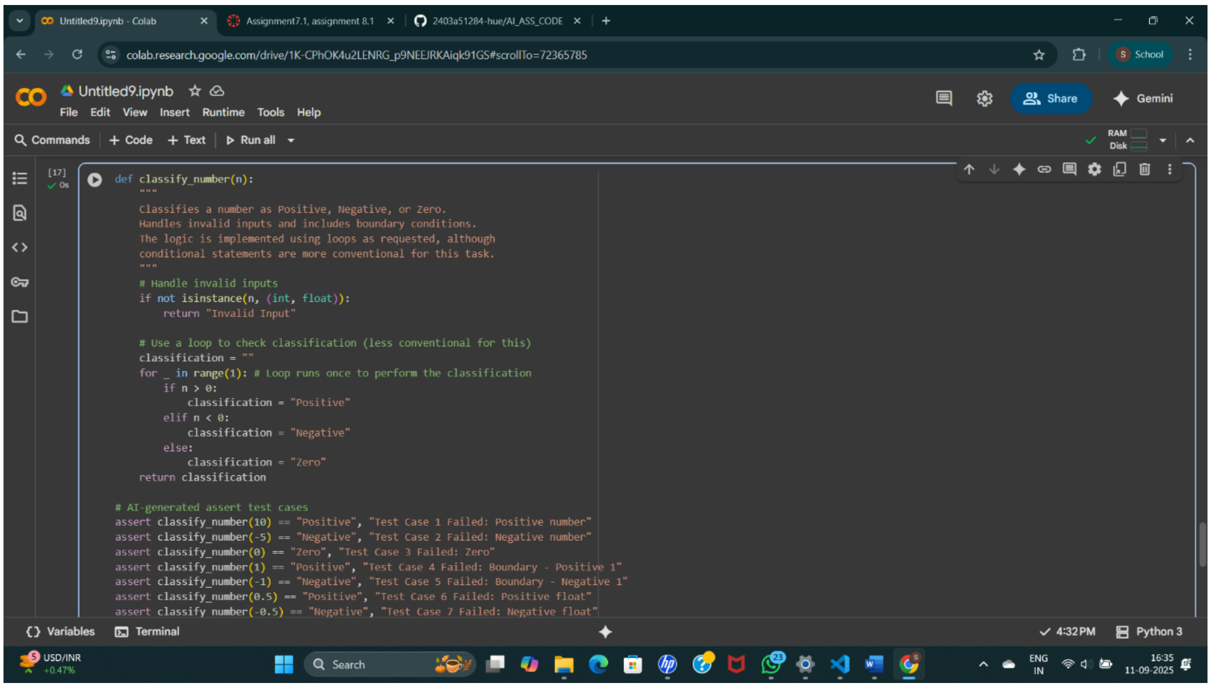
Implement the function classify number(n) to classify numbers as "Positive", "Negative", or "Zero".

The function should handle invalid inputs (like strings and None) and include boundary conditions (-1, 0, 1).

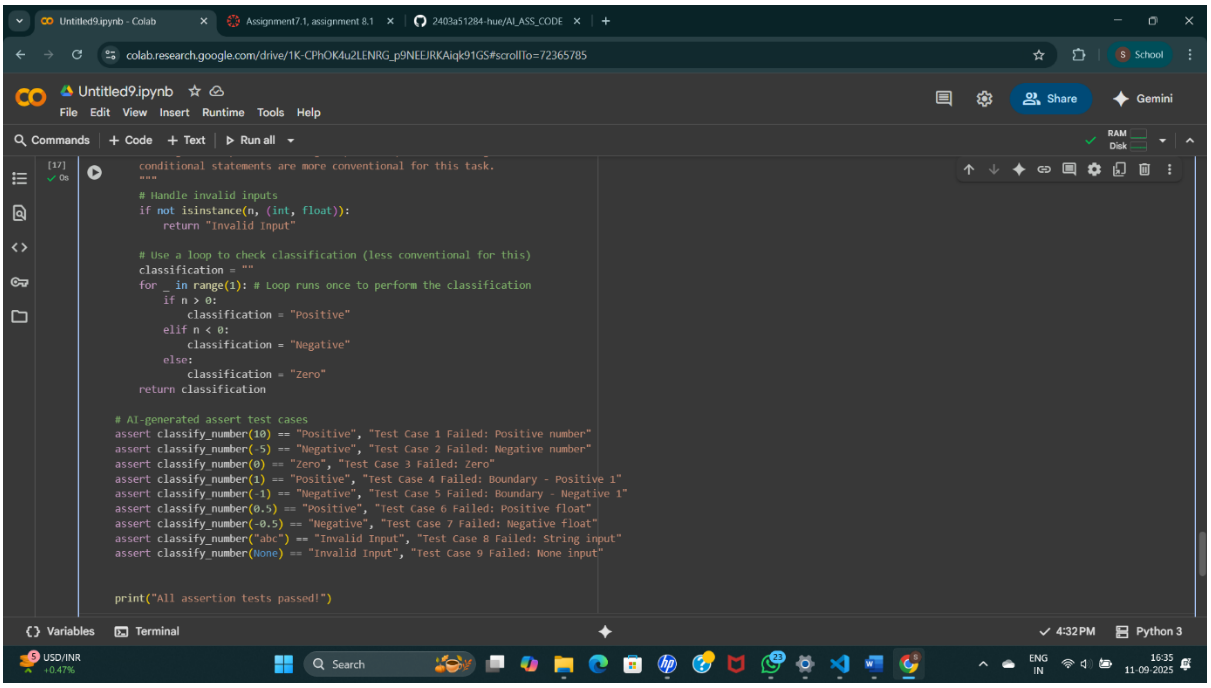
Use loops if needed for input validation or processing.

Use AI to generate at least 3 assert test cases.

CODE:

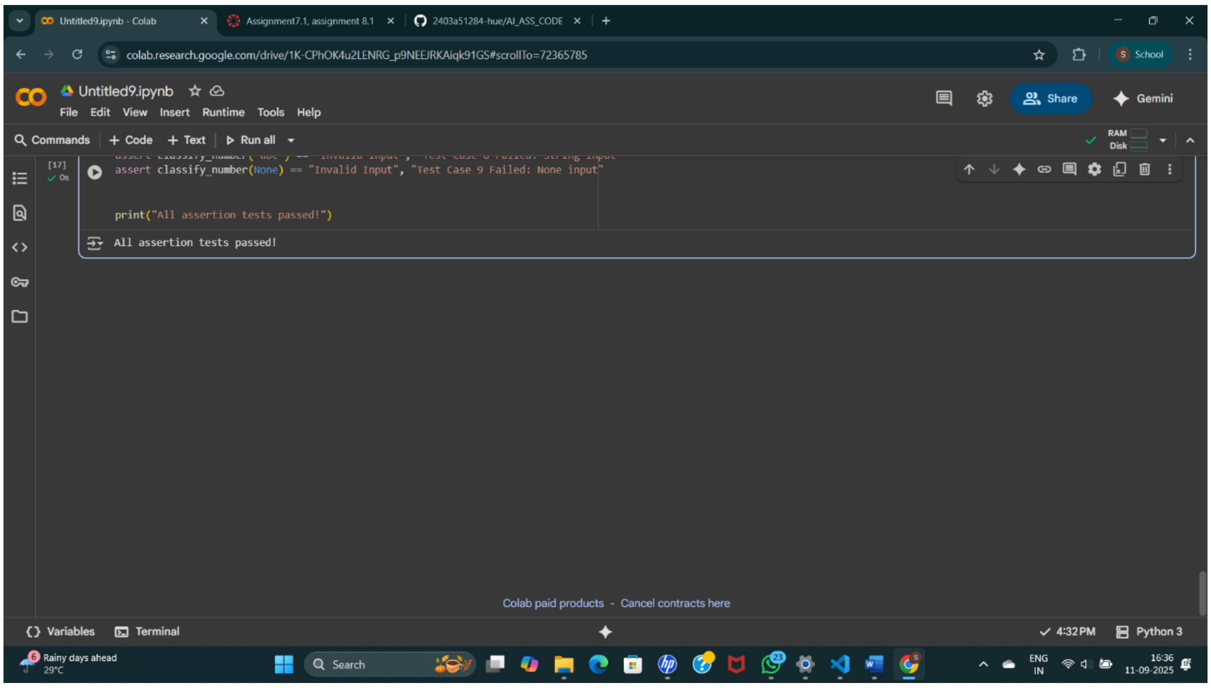


```
def classify_number(n):
    """
    Classifies a number as Positive, Negative, or Zero.
    Handles invalid inputs and includes boundary conditions.
    The logic is implemented using loops as requested, although
    conditional statements are more conventional for this task.
    """
    # Handle invalid inputs
    if not isinstance(n, (int, float)):
        return "Invalid Input"

    # Use a loop to check classification (less conventional for this)
    classification = ""
    for _ in range(1): # Loop runs once to perform the classification
        if n > 0:
            classification = "Positive"
        elif n < 0:
            classification = "Negative"
        else:
            classification = "Zero"
    return classification

# AI-generated assert test cases
assert classify_number(10) == "Positive", "Test Case 1 Failed: Positive number"
assert classify_number(-5) == "Negative", "Test Case 2 Failed: Negative number"
assert classify_number(0) == "Zero", "Test Case 3 Failed: Zero"
assert classify_number(1) == "Positive", "Test Case 4 Failed: Boundary - Positive 1"
assert classify_number(-1) == "Negative", "Test Case 5 Failed: Boundary - Negative 1"
assert classify_number(0.5) == "Positive", "Test Case 6 Failed: Positive float"
assert classify_number(-0.5) == "Negative", "Test Case 7 Failed: Negative float"
```



```
    conditional statements are more conventional for this task.
    """
    # Handle invalid inputs
    if not isinstance(n, (int, float)):
        return "Invalid Input"

    # Use a loop to check classification (less conventional for this)
    classification = ""
    for _ in range(1): # Loop runs once to perform the classification
        if n > 0:
            classification = "Positive"
        elif n < 0:
            classification = "Negative"
        else:
            classification = "Zero"
    return classification

# AI-generated assert test cases
assert classify_number(10) == "Positive", "Test Case 1 Failed: Positive number"
assert classify_number(-5) == "Negative", "Test Case 2 Failed: Negative number"
assert classify_number(0) == "Zero", "Test Case 3 Failed: Zero"
assert classify_number(1) == "Positive", "Test Case 4 Failed: Boundary - Positive 1"
assert classify_number(-1) == "Negative", "Test Case 5 Failed: Boundary - Negative 1"
assert classify_number(0.5) == "Positive", "Test Case 6 Failed: Positive float"
assert classify_number(-0.5) == "Negative", "Test Case 7 Failed: Negative float"
assert classify_number("abc") == "Invalid Input", "Test Case 8 Failed: String input"
assert classify_number(None) == "Invalid Input", "Test Case 9 Failed: None input"


print("All assertion tests passed!")
```

OUTPUT:



```
assert classify_number("abc") == "Invalid Input", "Test Case 8 Failed: String input"
assert classify_number(None) == "Invalid Input", "Test Case 9 Failed: None input"


print("All assertion tests passed!")
```

```
All assertion tests passed!
```

Task Description #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
Expected Output #3:
• Function correctly identifying anagrams and passing all AI-
generated tests.
Task Description #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

Expected Output #4:

• Fully functional class passing all assertions.

PROMPT:

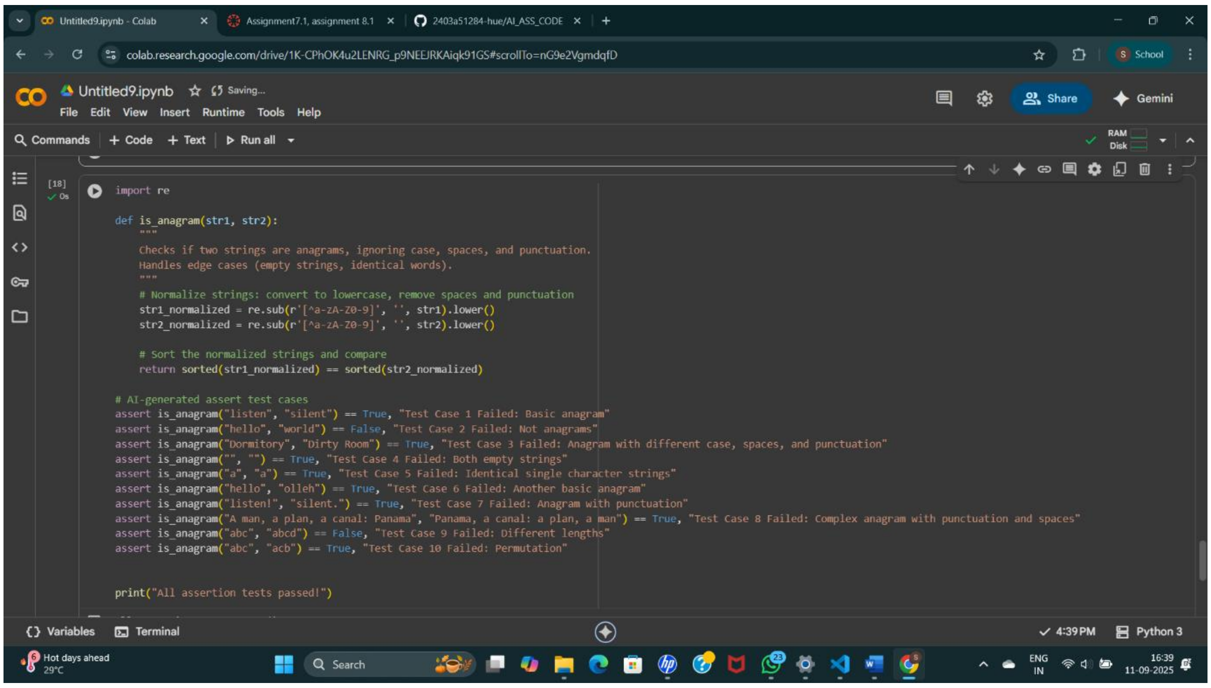
Implement the function is_anagram(str1, str2) to check if two strings are anagrams.

Ignore case, spaces, and punctuation.

Handle edge cases such as empty strings and identical words.

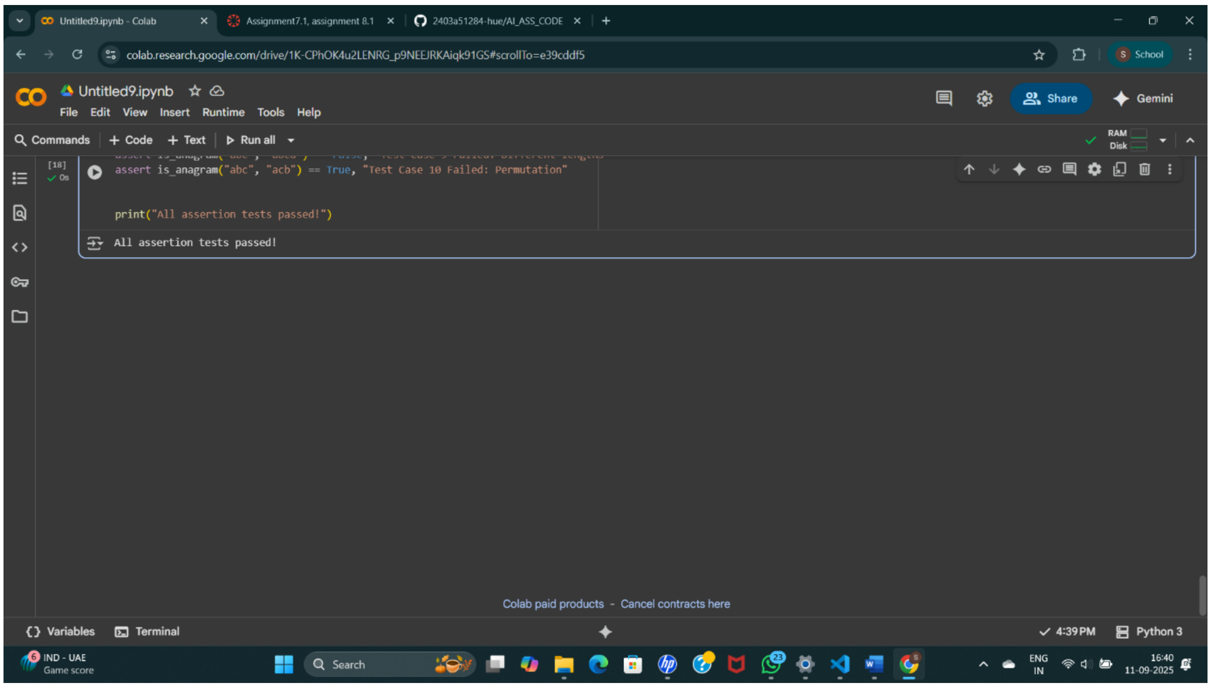
Use AI to generate at least 3 assert test cases.

CODE:


Untitled9.ipynb - Colab
Assignment7.1, assignment 8.1
2403a51284-hue/AI_ASS_CODE
colab.research.google.com/drive/1K-CPhOK4u2LENRG_p9NEEJRKAiqk91GS#scrollTo=nG9e2VgmdqfD
Untitled9.ipynb
Saving...
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Gemini
[18]
Variables
Terminal
4:39 PM
Python 3

```python
import re

def is_anagram(str1, str2):
    """
    Checks if two strings are anagrams, ignoring case, spaces, and punctuation.
    Handles edge cases (empty strings, identical words).
    """
    # Normalize strings: convert to lowercase, remove spaces and punctuation
    str1_normalized = re.sub(r'[^a-zA-Z0-9]', '', str1).lower()
    str2_normalized = re.sub(r'[^a-zA-Z0-9]', '', str2).lower()

    # Sort the normalized strings and compare
    return sorted(str1_normalized) == sorted(str2_normalized)

# AI-generated assert test cases
assert is_anagram("listen", "silent") == True, "Test Case 1 Failed: Basic anagram"
assert is_anagram("hello", "world") == False, "Test Case 2 Failed: Not anagrams"
assert is_anagram("Dormitory", "Dirty Room") == True, "Test Case 3 Failed: Anagram with different case, spaces, and punctuation"
assert is_anagram("", "") == True, "Test Case 4 Failed: Both empty strings"
assert is_anagram("a", "a") == True, "Test Case 5 Failed: Identical single character strings"
assert is_anagram("hello", "olleh") == True, "Test Case 6 Failed: Another basic anagram"
assert is_anagram("listen!", "silent.") == True, "Test Case 7 Failed: Anagram with punctuation"
assert is_anagram("A man, a plan, a canal: Panama", "Panama, a canal: a plan, a man") == True, "Test Case 8 Failed: Complex anagram with punctuation and spaces"
assert is_anagram("abc", "abcd") == False, "Test Case 9 Failed: Different lengths"
assert is_anagram("abc", "acb") == True, "Test Case 10 Failed: Permutation"


print("All assertion tests passed!")
```

OUTPUT:



```
assert is_anagram("abc", "acb") == True, "Test Case 10 Failed: Permutation"


print("All assertion tests passed!")
```

```
All assertion tests passed!
```

Task Description #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

Expected Output #4:

• Fully functional class passing all assertions.

PROMPT:

Implement an Inventory class with methods for stock management:

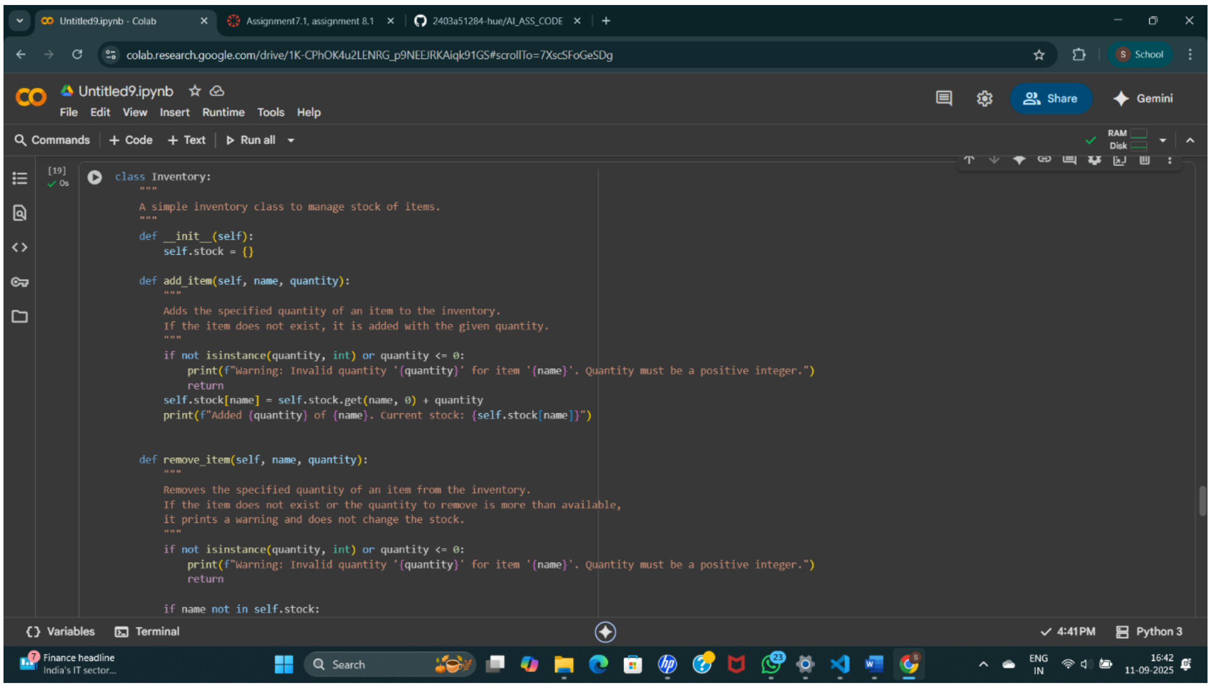
add_item(name, quantity)

remove_item(name, quantity)

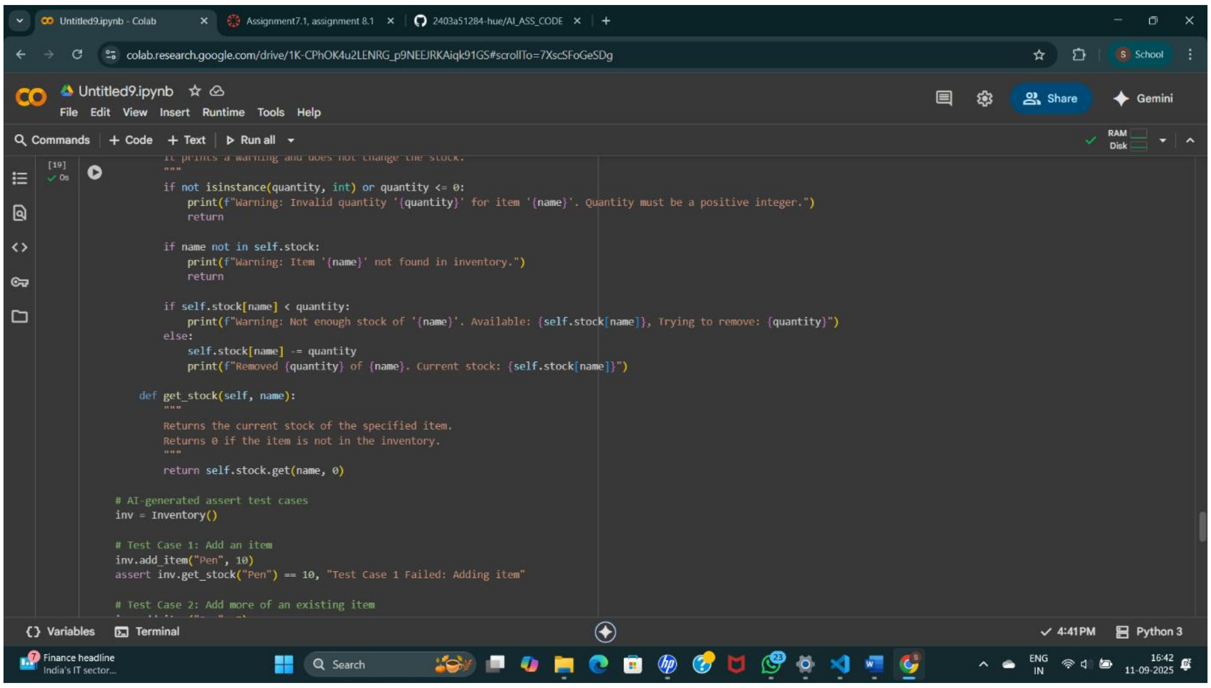
get_stock(name)

Use AI to generate at least 3 assert-based tests for the class.

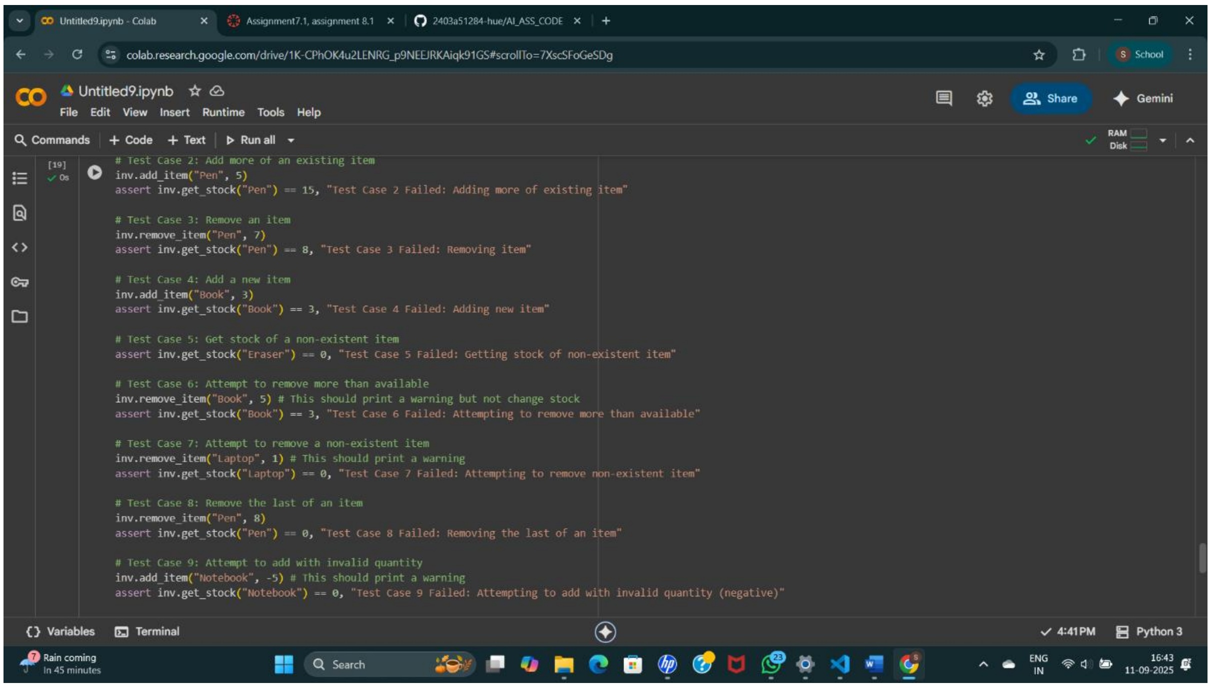
CODE:



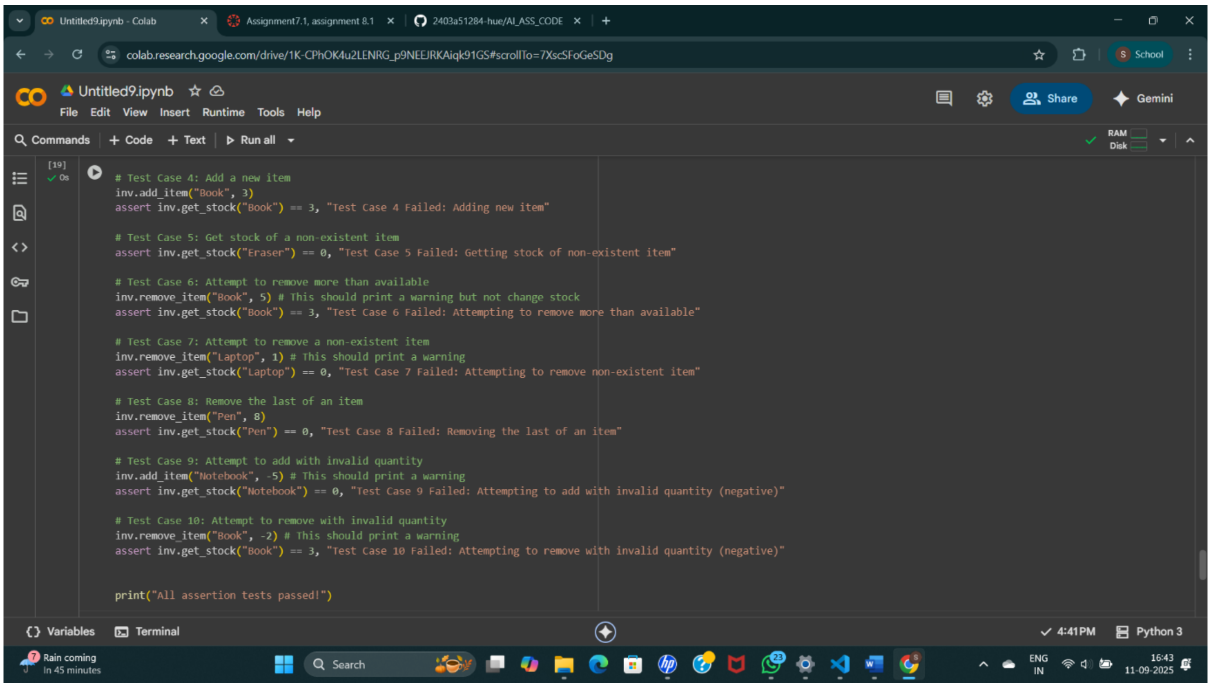
```python
class Inventory:
    """
    A simple inventory class to manage stock of items.
    """
    def __init__(self):
        self.stock = {}

    def add_item(self, name, quantity):
        """
        Adds the specified quantity of an item to the inventory.
        If the item does not exist, it is added with the given quantity.
        """
        if not isinstance(quantity, int) or quantity <= 0:
            print(f"Warning: Invalid quantity '{quantity}' for item '{name}'. Quantity must be a positive integer.")
            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 the 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.
        """
        if not isinstance(quantity, int) or quantity <= 0:
            print(f"Warning: Invalid quantity '{quantity}' for item '{name}'. Quantity must be a positive integer.")
            return

        if name not in self.stock:
```



```python
        if not isinstance(quantity, int) or quantity <= 0:
            print(f"Warning: Invalid quantity '{quantity}' for item '{name}'. Quantity must be a positive integer.")
            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}'. Available: {self.stock[name]}, Trying to remove: {quantity}")
        else:
            self.stock[name] -= quantity
            print(f"Removed {quantity} of {name}. Current stock: {self.stock[name]}")

    def get_stock(self, name):
        """
        Returns the current stock of the specified item.
        Returns 0 if the item is not in the inventory.
        """
        return self.stock.get(name, 0)

# AI-generated assert test cases
inv = Inventory()

# Test Case 1: Add an item
inv.add_item("Pen", 10)
assert inv.get_stock("Pen") == 10, "Test Case 1 Failed: Adding item"

# Test Case 2: Add more of an existing item
```

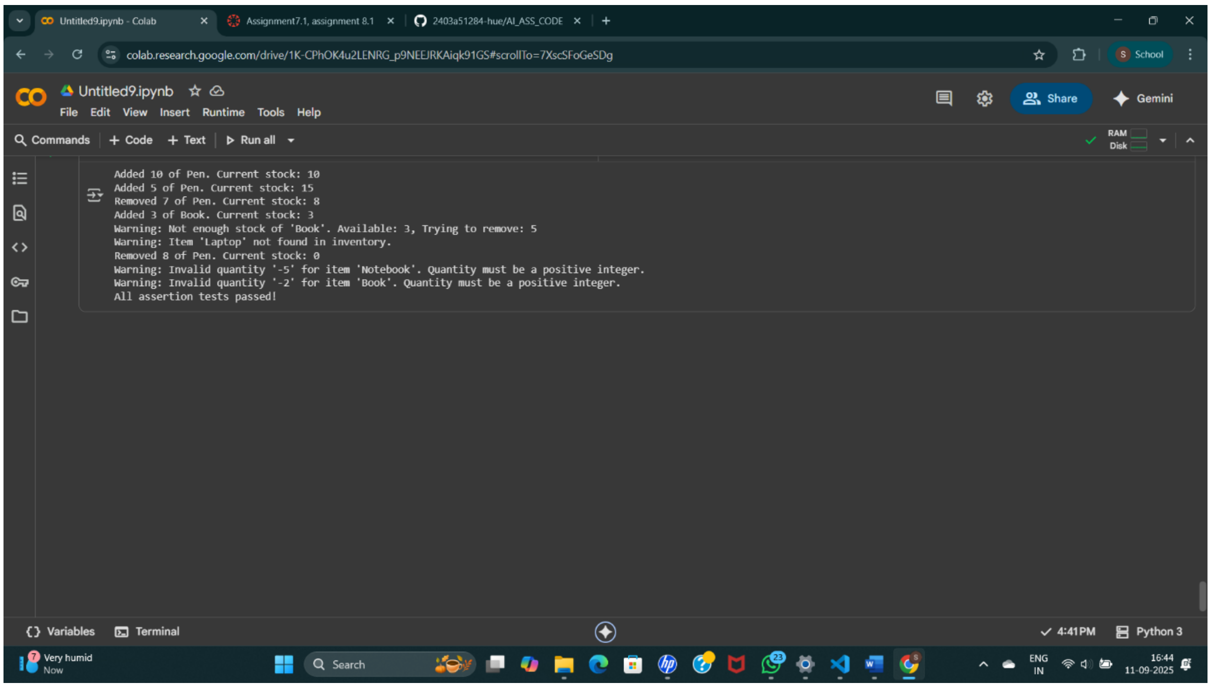

Untitled9.ipynb - Colab
Assignment7.1, assignment 8.1
2403a51284-hue/AI_ASS_CODE
colab.research.google.com/drive/1K-CPhOK4u2LENRG_p9NEEJRKAiqk91GS#scrollTo=7XscSFoGeSDg
Untitled9.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Gemini
[19]
Variables
Terminal
4:41PM
Python 3

```
# Test Case 2: Add more of an existing item
inv.add_item("Pen", 5)
assert inv.get_stock("Pen") == 15, "Test Case 2 Failed: Adding more of existing item"

# Test Case 3: Remove an item
inv.remove_item("Pen", 7)
assert inv.get_stock("Pen") == 8, "Test Case 3 Failed: Removing item"

# Test Case 4: Add a new item
inv.add_item("Book", 3)
assert inv.get_stock("Book") == 3, "Test Case 4 Failed: Adding new item"

# Test Case 5: Get stock of a non-existent item
assert inv.get_stock("Eraser") == 0, "Test Case 5 Failed: Getting stock of non-existent item"

# Test Case 6: Attempt to remove more than available
inv.remove_item("Book", 5) # This should print a warning but not change stock
assert inv.get_stock("Book") == 3, "Test Case 6 Failed: Attempting to remove more than available"

# Test Case 7: Attempt to remove a non-existent item
inv.remove_item("Laptop", 1) # This should print a warning
assert inv.get_stock("Laptop") == 0, "Test Case 7 Failed: Attempting to remove non-existent item"

# Test Case 8: Remove the last of an item
inv.remove_item("Pen", 8)
assert inv.get_stock("Pen") == 0, "Test Case 8 Failed: Removing the last of an item"

# Test Case 9: Attempt to add with invalid quantity
inv.add_item("Notebook", -5) # This should print a warning
assert inv.get_stock("Notebook") == 0, "Test Case 9 Failed: Attempting to add with invalid quantity (negative)"
```


Untitled9.ipynb - Colab
Assignment7.1, assignment 8.1
2403a51284-hue/AI_ASS_CODE
colab.research.google.com/drive/1K-CPhOK4u2LENRG_p9NEEJRKAiqk91GS#scrollTo=7XscSFoGeSDg
Untitled9.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Gemini
[19]
Variables
Terminal
4:41PM
Python 3

```
# Test Case 4: Add a new item
inv.add_item("Book", 3)
assert inv.get_stock("Book") == 3, "Test Case 4 Failed: Adding new item"

# Test Case 5: Get stock of a non-existent item
assert inv.get_stock("Eraser") == 0, "Test Case 5 Failed: Getting stock of non-existent item"

# Test Case 6: Attempt to remove more than available
inv.remove_item("Book", 5) # This should print a warning but not change stock
assert inv.get_stock("Book") == 3, "Test Case 6 Failed: Attempting to remove more than available"

# Test Case 7: Attempt to remove a non-existent item
inv.remove_item("Laptop", 1) # This should print a warning
assert inv.get_stock("Laptop") == 0, "Test Case 7 Failed: Attempting to remove non-existent item"

# Test Case 8: Remove the last of an item
inv.remove_item("Pen", 8)
assert inv.get_stock("Pen") == 0, "Test Case 8 Failed: Removing the last of an item"

# Test Case 9: Attempt to add with invalid quantity
inv.add_item("Notebook", -5) # This should print a warning
assert inv.get_stock("Notebook") == 0, "Test Case 9 Failed: Attempting to add with invalid quantity (negative)"

# Test Case 10: Attempt to remove with invalid quantity
inv.remove_item("Book", -2) # This should print a warning
assert inv.get_stock("Book") == 3, "Test Case 10 Failed: Attempting to remove with invalid quantity (negative)"


print("All assertion tests passed!")
```

OUTPUT:



```
Added 10 of Pen. Current stock: 10
Added 5 of Pen. Current stock: 15
Removed 7 of Pen. Current stock: 8
Added 3 of Book. Current stock: 3
Warning: Not enough stock of 'Book'. Available: 3, Trying to remove: 5
Warning: Item 'Laptop' not found in inventory.
Removed 8 of Pen. Current stock: 0
Warning: Invalid quantity '-5' for item 'Notebook'. Quantity must be a positive integer.
Warning: Invalid quantity '-2' for item 'Book'. Quantity must be a positive integer.
All assertion tests passed!
```

• Task5: 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"
Expected Output #5:
• Function passes all AI-generated assertions and handles edge cases.
Deliverables (For All Tasks)
1. AI-generated prompts for code and test case generation.
2. At least 3 assert test cases for each task.
3. AI-generated initial code and execution screenshots.
4. Analysis of whether code passes all tests.
5. Improved final version with inline comments and explanation.
6. Compiled report (Word/PDF) with prompts, test cases, assertions,code, and output.

PROMPT:

Implement the function validate_and_format_date(date_str) to:

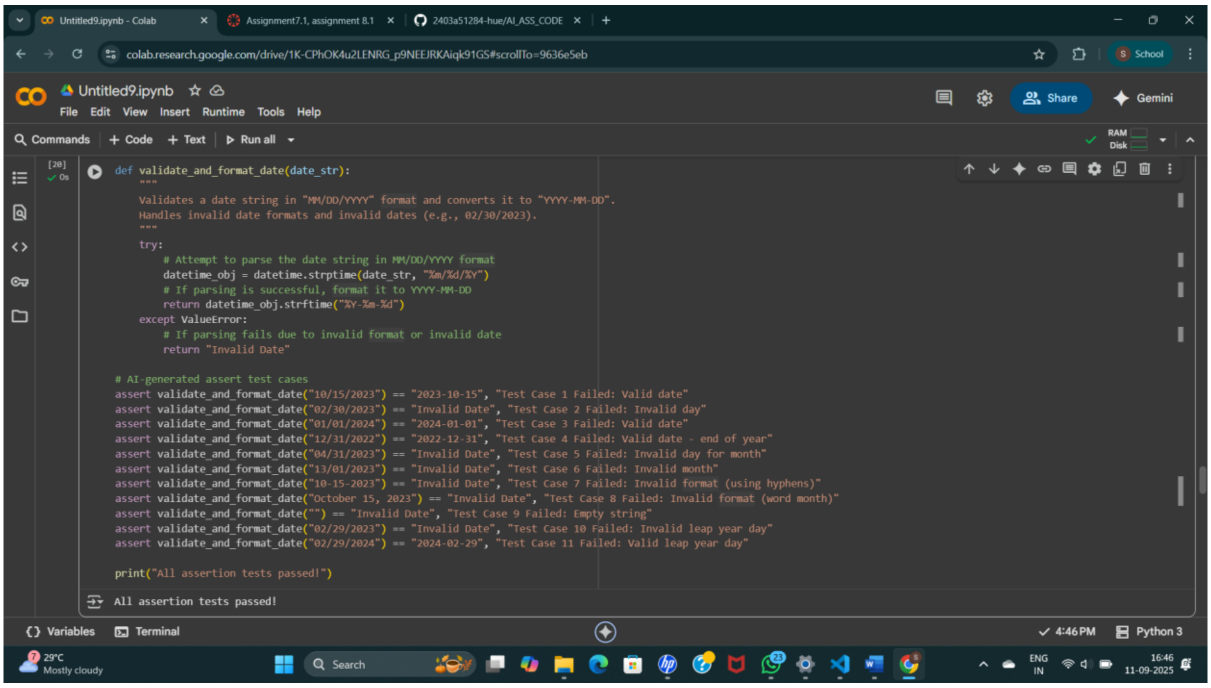
Validate dates in "MM/DD/YYYY" format.

Handle invalid dates.

Convert valid dates to "YYYY-MM-DD" format.

Use AI to generate at least 3 assert test cases.

CODE AND OUTPUT:



```
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 and invalid dates (e.g., 02/30/2023).
    """
    try:
        # Attempt to parse the date string in MM/DD/YYYY format
        datetime_obj = datetime.strptime(date_str, "%m/%d/%Y")
        # If parsing is successful, format it to YYYY-MM-DD
        return datetime_obj.strftime("%Y-%m-%d")
    except ValueError:
        # If parsing fails due to invalid format or invalid date
        return "Invalid Date"

# AI-generated assert test cases
assert validate_and_format_date("10/15/2023") == "2023-10-15", "Test Case 1 Failed: Valid date"
assert validate_and_format_date("02/30/2023") == "Invalid Date", "Test Case 2 Failed: Invalid day"
assert validate_and_format_date("01/01/2024") == "2024-01-01", "Test Case 3 Failed: Valid date"
assert validate_and_format_date("12/31/2022") == "2022-12-31", "Test Case 4 Failed: Valid date - end of year"
assert validate_and_format_date("04/31/2023") == "Invalid Date", "Test Case 5 Failed: Invalid day for month"
assert validate_and_format_date("13/01/2023") == "Invalid Date", "Test Case 6 Failed: Invalid month"
assert validate_and_format_date("10-15-2023") == "Invalid Date", "Test Case 7 Failed: Invalid format (using hyphens)"
assert validate_and_format_date("October 15, 2023") == "Invalid Date", "Test Case 8 Failed: Invalid format (word month)"
assert validate_and_format_date("") == "Invalid Date", "Test Case 9 Failed: Empty string"
assert validate_and_format_date("02/29/2023") == "Invalid Date", "Test Case 10 Failed: Invalid leap year day"
assert validate_and_format_date("02/29/2024") == "2024-02-29", "Test Case 11 Failed: Valid leap year day"

print("All assertion tests passed!")
```

```
All assertion tests passed!
```