

# LAB ASSIGNMENT 8.1

NAME: A.Harshita

ID.NO: 2303A52211

SUBJECT: AI ASST CODING

## Task 1: Password Strength Validator

The screenshot shows a Google Colab notebook titled 'Untitled39.ipynb'. The code defines a function `is_strong_password` and runs several test cases. A `NameError` is raised because the function is not defined before the test cases are executed. The error message is: `NameError: name 'is_strong_password' is not defined`. The notebook interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar with icons for running and saving, and a sidebar with file explorer and search options. The bottom status bar shows the temperature as 29°C and the time as 13:52 on 09-02-2026.

```
# Test cases (AI-generated)
assert is_strong_password("abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == True
assert is_strong_password("Abc 123@") == False
assert is_strong_password("Ab1@") == False

-----
Traceback (most recent call last)
/tmp/ipython-input-3784212921.py in <cell line: 0>()
----> 2 assert is_strong_password("abcd@123") == True
      3 assert is_strong_password("abcd123") == False
      4 assert is_strong_password("ABCD@1234") == True
      5 assert is_strong_password("Abc 123@") == False
      6 assert is_strong_password("Ab1@") == False

NameError: name 'is_strong_password' is not defined

Next steps: Explain error
```

```
def is_strong_password(password):
    if len(password) < 8:
        return False

    if " " in password:
        return False

    has_upper = False
    has_lower = False
    has_digit = False
    has_special = False

    for ch in password:
        if ch.isupper():
            has_upper = True
        elif ch.islower():
            has_lower = True
        elif ch.isdigit():
            has_digit = True
        else:
            has_special = True

    return has_upper and has_lower and has_digit and has_special
```

The screenshot shows the same Google Colab notebook after the function has been defined before the test cases. The code now runs successfully, and the output shows that all test cases passed. The notebook interface is the same as the previous screenshot, but the output area now displays the results of the function calls. The bottom status bar shows the temperature as 29°C and the time as 13:52 on 09-02-2026.

```
has_digit = False
has_special = False

for ch in password:
    if ch.isupper():
        has_upper = True
    elif ch.islower():
        has_lower = True
    elif ch.isdigit():
        has_digit = True
    else:
        has_special = True

return has_upper and has_lower and has_digit and has_special
```

```
assert is_strong_password("abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == True # fixed
assert is_strong_password("Abc 123@") == False
assert is_strong_password("Ab1@") == False

print("All test cases passed")
```

All test cases passed

```
passwords = ["Hello@123", "weakpass", "HELLO@123", "Hello 123@", "H1@123"]

for pwd in passwords:
    print(pwd, "-", is_strong_password(pwd))
```

Hello@123 → True  
weakpass → False  
HELLO@123 → False  
Hello 123@ → False  
H1@123 → False

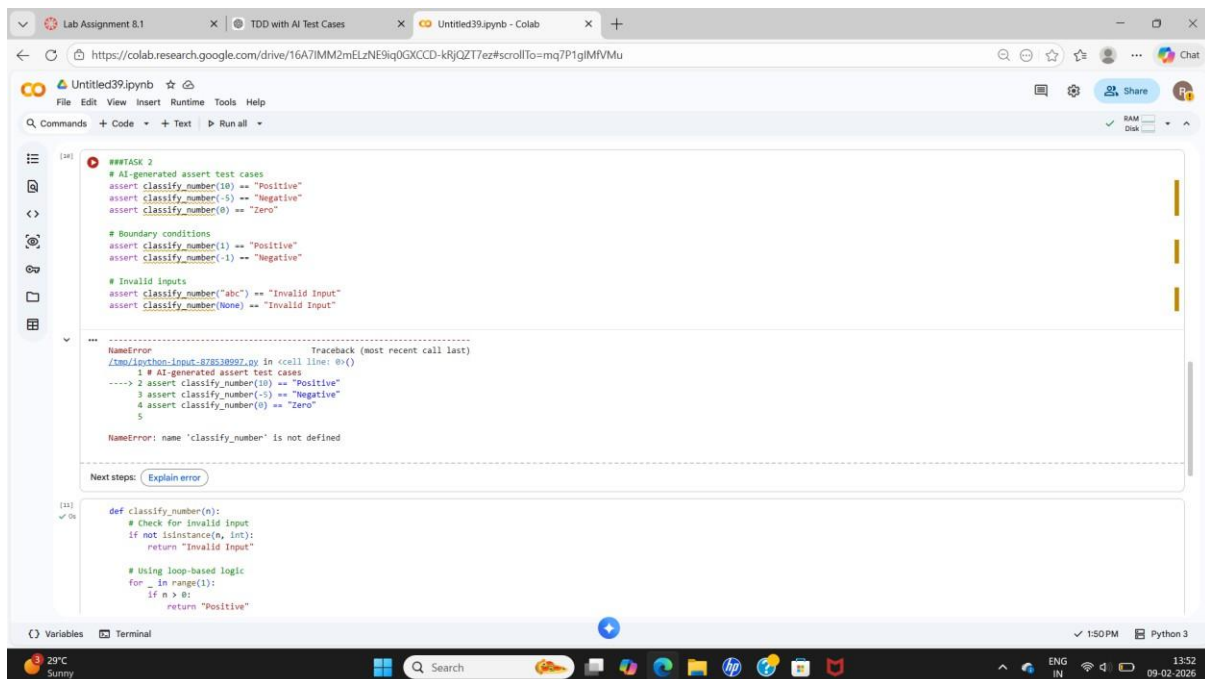
## Code Explanation:

The function checks whether a password is strong by verifying length, presence of uppercase, lowercase, digit, special character, and absence of spaces. It returns True for strong passwords and False otherwise.

## AI Explanation:

AI generated test cases for strong and weak passwords. These tests helped verify security rules and identify incorrect password patterns.

## Task 2: Number Classification Using Loops



The screenshot shows a Google Colab notebook with the following code and error:

```
##TASK 2
# AI-generated assert test cases
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"

# Boundary conditions
assert classify_number(1) == "Positive"
assert classify_number(-1) == "Negative"

# Invalid inputs
assert classify_number("abc") == "Invalid Input"
assert classify_number(None) == "Invalid Input"

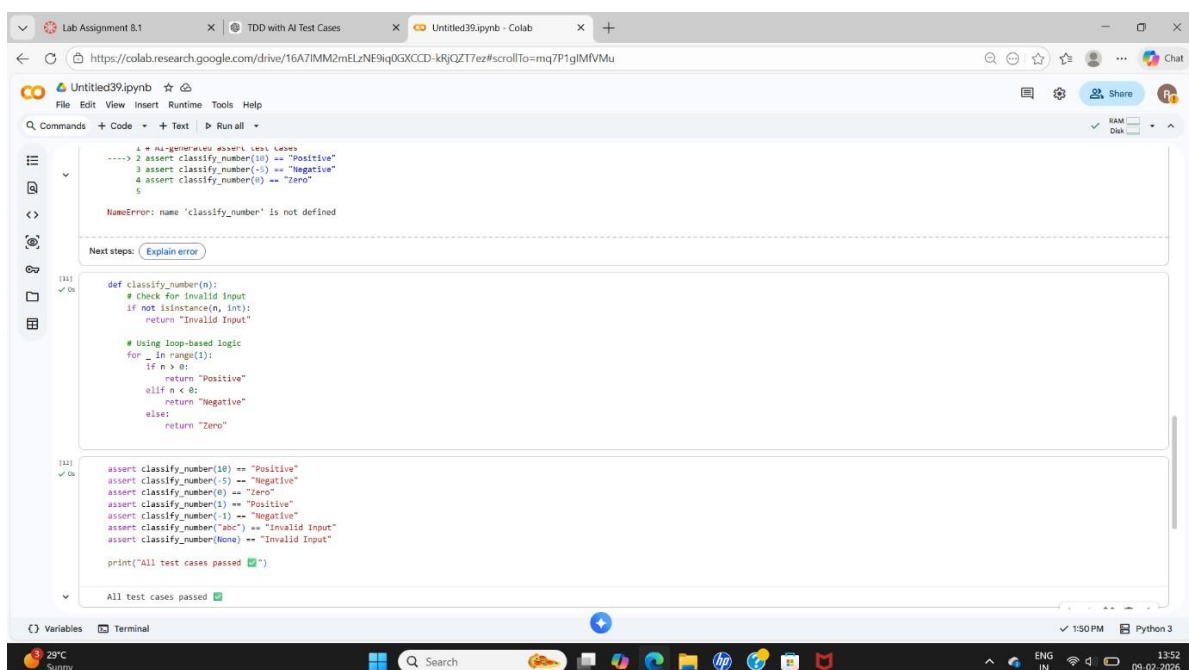
-----
NameError                                Traceback (most recent call last)
/tmp/ipython-input-878538997.py in <cell line: 0>()
      1 # AI-generated assert test cases
----> 2 assert classify_number(10) == "Positive"
      3 assert classify_number(-5) == "Negative"
      4 assert classify_number(0) == "Zero"
      5

NameError: name 'classify_number' is not defined

Next steps: Explain error

def classify_number(n):
    # Check for invalid input
    if not isinstance(n, int):
        return "Invalid Input"

    # Using loop-based logic
    for _ in range(1):
        if n > 0:
            return "Positive"
```



The screenshot shows the same Colab notebook with the completed function and successful test cases:

```
def classify_number(n):
    # Check for invalid input
    if not isinstance(n, int):
        return "Invalid Input"

    # Using loop-based logic
    for _ in range(1):
        if n > 0:
            return "Positive"
        elif n < 0:
            return "Negative"
        else:
            return "Zero"

assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
assert classify_number(1) == "Positive"
assert classify_number(-1) == "Negative"
assert classify_number("abc") == "Invalid Input"
assert classify_number(None) == "Invalid Input"

print("All test cases passed")

All test cases passed
```

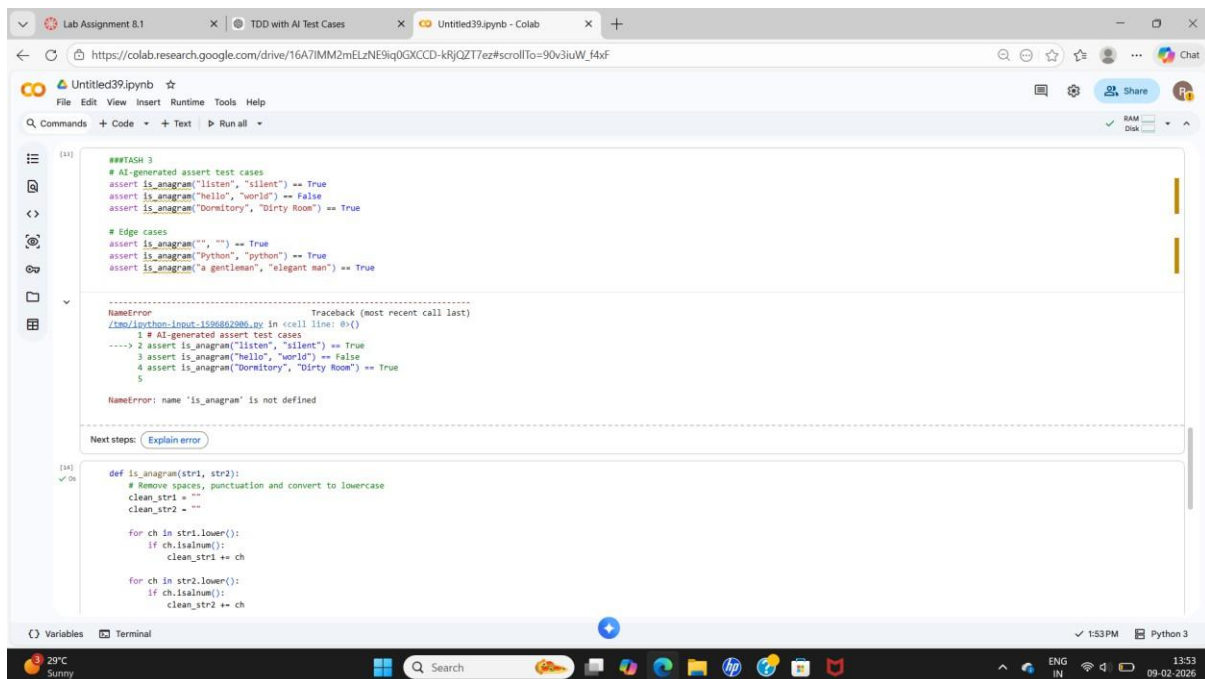
## Code Explanation:

The function classifies a number as Positive, Negative, or Zero. It first checks for invalid inputs like strings or None and then uses logic inside a loop to return the correct classification.

## AI Explanation:

AI generated test cases including boundary values (-1, 0, 1) and invalid inputs, ensuring correct classification.

## Task 3: Anagram Checker



```
##TASK 3
# AI-generated assert test cases
assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True

# Edge cases
assert is_anagram("", "") == True
assert is_anagram("Python", "python") == True
assert is_anagram("a gentleman", "elegant man") == True

-----
NameError                                Traceback (most recent call last)
~/tmp/python-Input-1590862080.py in <cell line: 0>()
----> 2 assert is_anagram("listen", "silent") == True
      3 assert is_anagram("hello", "world") == False
      4 assert is_anagram("Dormitory", "Dirty Room") == True
      5

NameError: name 'is_anagram' is not defined

Next steps: Explain error

[14] ✓ 2s
def is_anagram(str1, str2):
    # Remove spaces, punctuation and convert to lowercase
    clean_str1 = ""
    clean_str2 = ""

    for ch in str1.lower():
        if ch.isalnum():
            clean_str1 += ch

    for ch in str2.lower():
        if ch.isalnum():
            clean_str2 += ch
```

The screenshot shows a Google Colab notebook titled 'Untitled39.ipynb'. The code defines a function `is_anagram(str1, str2)` that cleans strings by removing spaces and punctuation, converting them to lowercase, and then compares the sorted characters. Below the function, several assertions are used to test the function with various inputs, including edge cases like empty strings and strings with different lengths. The notebook shows the function being called with `is_anagram('listen', 'silent')` and the result `True`. The bottom status bar indicates 'All test cases passed'.

```
def is_anagram(str1, str2):  
    # Remove spaces, punctuation and convert to lowercase  
    clean_str1 = ""  
    clean_str2 = ""  
  
    for ch in str1.lower():  
        if ch.isalnum():  
            clean_str1 += ch  
  
    for ch in str2.lower():  
        if ch.isalnum():  
            clean_str2 += ch  
  
    # If lengths differ, not anagrams  
    if len(clean_str1) != len(clean_str2):  
        return False  
  
    # Compare sorted characters  
    return sorted(clean_str1) == sorted(clean_str2)  
  
assert is_anagram("listen", "silent") == True  
assert is_anagram("hello", "world") == False  
assert is_anagram("Dormitory", "Dirty Room") == True  
assert is_anagram("", "") == True  
assert is_anagram("Python", "python") == True  
assert is_anagram("a gentleman", "elegant man") == True  
  
print("All test cases passed")
```

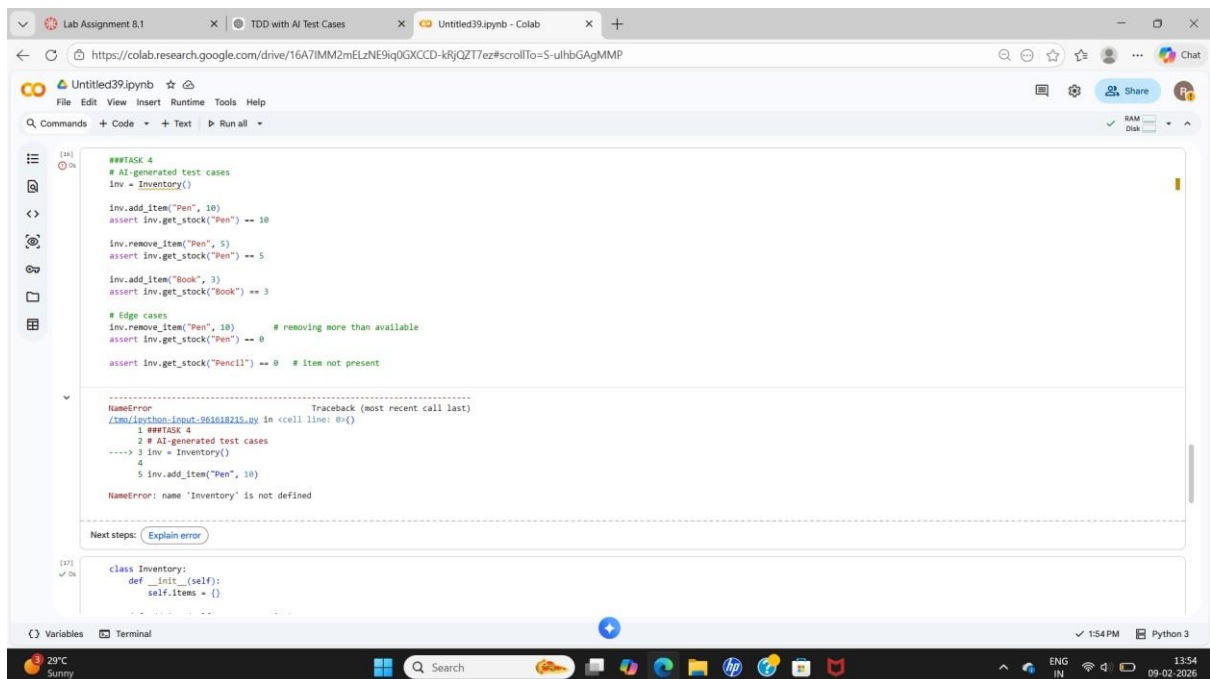
## Code Explanation:

The function removes spaces and punctuation, converts strings to lowercase, and compares characters to check if two strings are anagrams.

## AI Explanation:

AI-generated tests helped cover case differences, spaces, and empty strings for accurate string comparison.

## Task 4: Inventory Class (Real-World Simulation)



```
##TASK 4
# AI-generated 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

# Edge cases
Inv.remove_item("Pen", 10) # removing more than available
assert Inv.get_stock("Pen") == 0

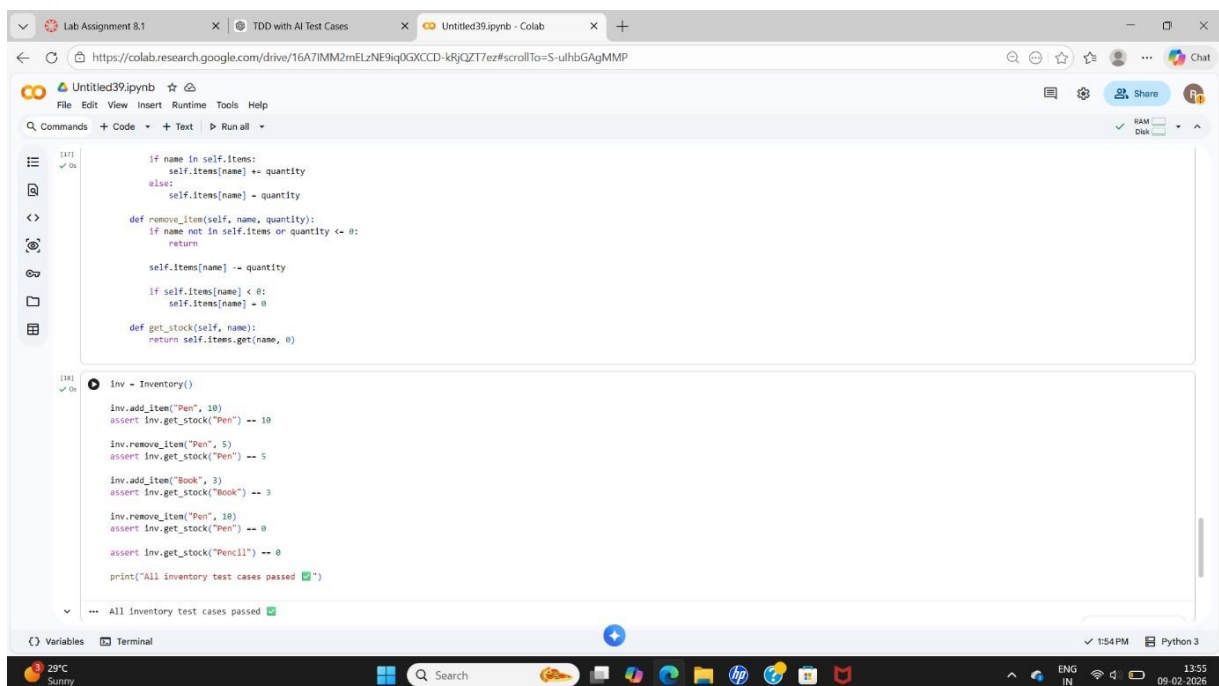
assert Inv.get_stock("Pencil") == 0 # Item not present

NameError                                Traceback (most recent call last)
/tmp/ipython-input-961618215_0y in <cell line: 0>()
----> 1 ##TASK 4
      2 # AI-generated test cases
      3 Inv = Inventory()
      4
      5 Inv.add_item("Pen", 10)

NameError: name 'Inventory' is not defined

Next steps: Explain error
```

```
class Inventory:
    def __init__(self):
        self.items = {}
```



```
if name in self.items:
    self.items[name] += quantity
else:
    self.items[name] = quantity

def remove_item(self, name, quantity):
    if name not in self.items or quantity <= 0:
        return

    self.items[name] -= quantity

    if self.items[name] < 0:
        self.items[name] = 0

def get_stock(self, name):
    return self.items.get(name, 0)

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

Inv.remove_item("Pen", 10)
assert Inv.get_stock("Pen") == 0

assert Inv.get_stock("Pencil") == 0

print("All inventory test cases passed")
```

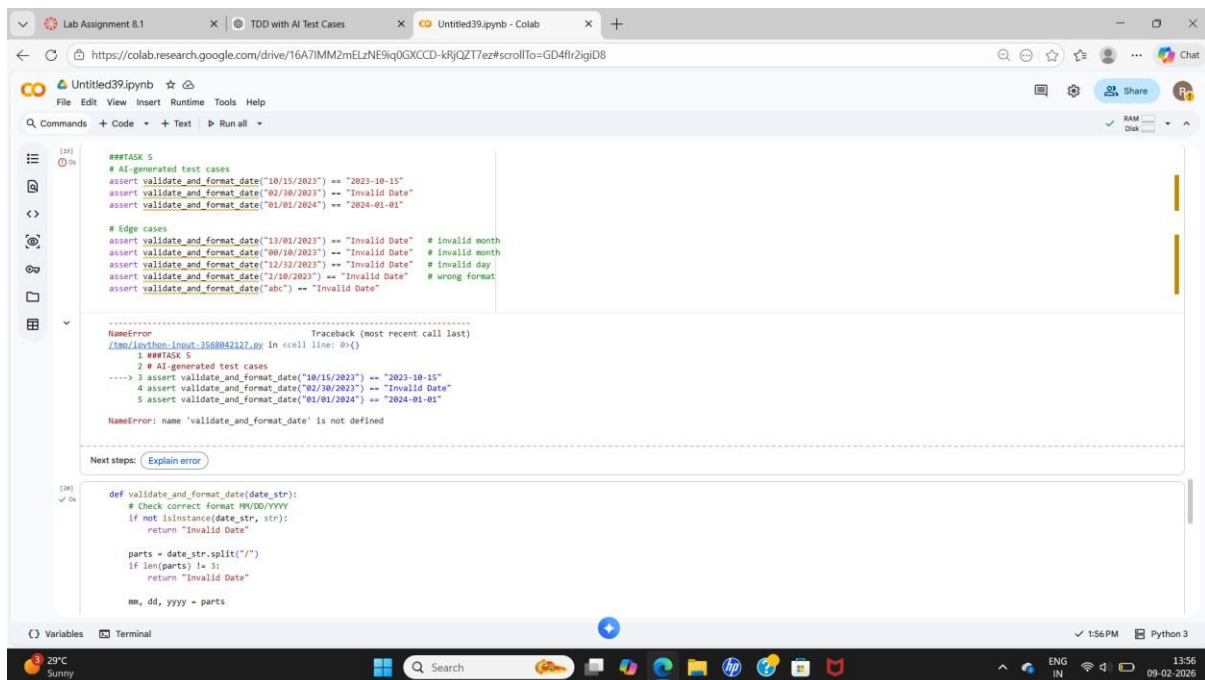
### Code Explanation:

The Inventory class manages item stock. It allows adding items, removing items safely, and checking stock quantity.

### AI Explanation:

AI provided test cases simulating real inventory actions like adding, removing, and checking items.

## Task 5: Date Validation & Formatting



The screenshot shows a Jupyter Notebook interface with a code cell containing the following Python code:

```
##TASK 5
# AI-generated 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"

# Edge cases
assert validate_and_format_date("13/01/2023") == "Invalid Date" # invalid month
assert validate_and_format_date("00/10/2023") == "Invalid Date" # invalid month
assert validate_and_format_date("12/32/2023") == "Invalid Date" # invalid day
assert validate_and_format_date("2/10/2023") == "Invalid Date" # wrong format
assert validate_and_format_date("abc") == "Invalid Date"

-----
NameError                                Traceback (most recent call last)
/tmp/ipython-input-3560842127.py in <cell line: 0>()
      1 ##TASK 5
      2 # AI-generated test cases
----> 3 assert validate_and_format_date("10/15/2023") == "2023-10-15"
      4 assert validate_and_format_date("02/30/2023") == "Invalid Date"
      5 assert validate_and_format_date("01/01/2024") == "2024-01-01"

NameError: name 'validate_and_format_date' is not defined

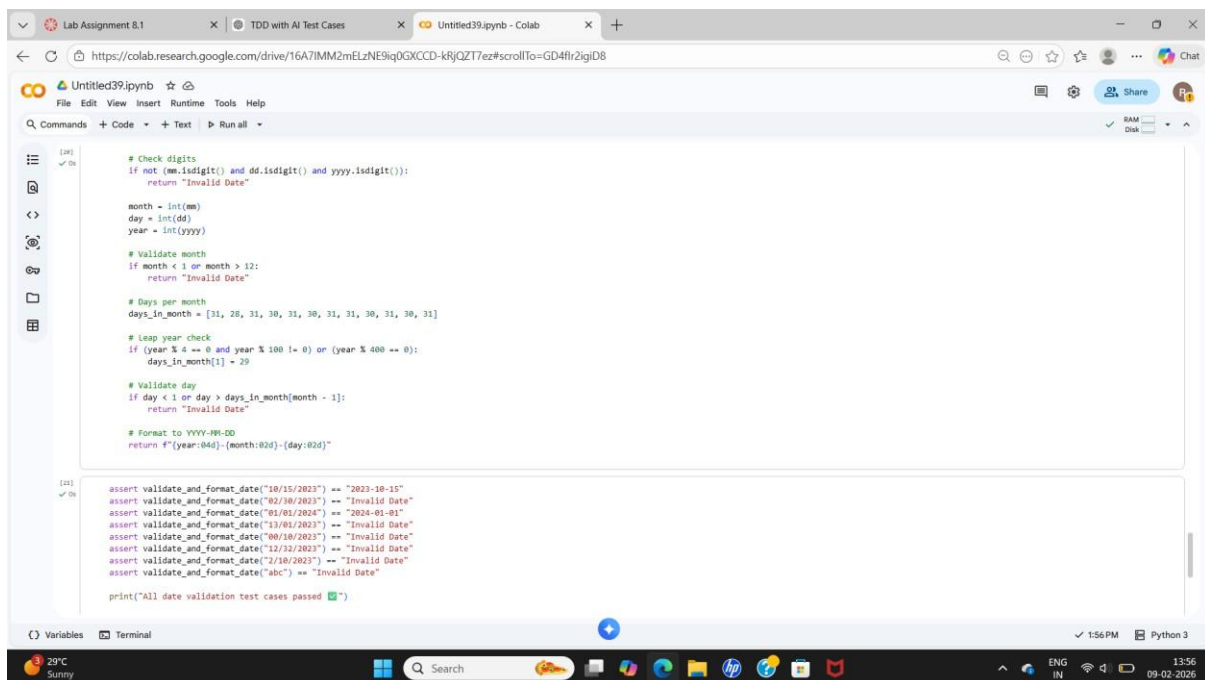
Next steps: Explain error
```

Below the error, a partial definition of the function is visible:

```
def validate_and_format_date(date_str):
    # Check correct format: MM/DD/YYYY
    if not isinstance(date_str, str):
        return "Invalid Date"

    parts = date_str.split("/")
    if len(parts) != 3:
        return "Invalid Date"

    mm, dd, yyyy = parts
```



The screenshot shows the same Jupyter Notebook interface with the complete implementation of the function:

```
# Check digits
if not (mm.isdigit() and dd.isdigit() and yyyy.isdigit()):
    return "Invalid Date"

month = int(mm)
day = int(dd)
year = int(yyyy)

# Validate month
if month < 1 or month > 12:
    return "Invalid Date"

# Days per month
days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

# Leap year check
if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    days_in_month[1] = 29

# Validate day
if day < 1 or day > days_in_month[month - 1]:
    return "Invalid Date"

# Format to YYYY-MM-DD
return f"{year:04d}-{month:02d}-{day:02d}"

[10]:
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"
assert validate_and_format_date("13/01/2023") == "Invalid Date"
assert validate_and_format_date("00/10/2023") == "Invalid Date"
assert validate_and_format_date("12/32/2023") == "Invalid Date"
assert validate_and_format_date("2/10/2023") == "Invalid Date"
assert validate_and_format_date("abc") == "Invalid Date"

print("All date validation test cases passed")
```

### Code Explanation:

The function checks whether a date is in MM/DD/YYYY format, validates the date, and converts valid dates to YYYY-MM-DD.

### AI Explanation:

AI generated test cases for valid dates, invalid formats, and edge cases like leap years.

