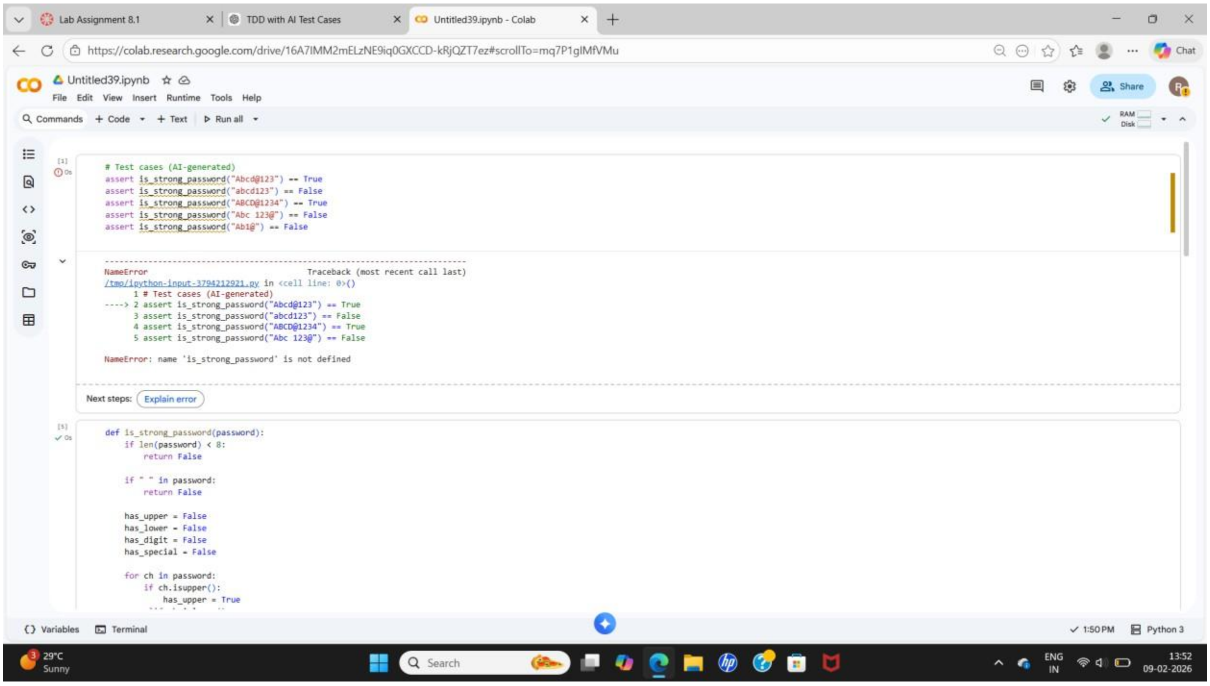
# LAB ASSIGNMENT 8.1

NAME: M.Geethika

ID.NO: 2303A52276

SUBJECT: AI ASST CODING

## Task 1: Password Strength Validator

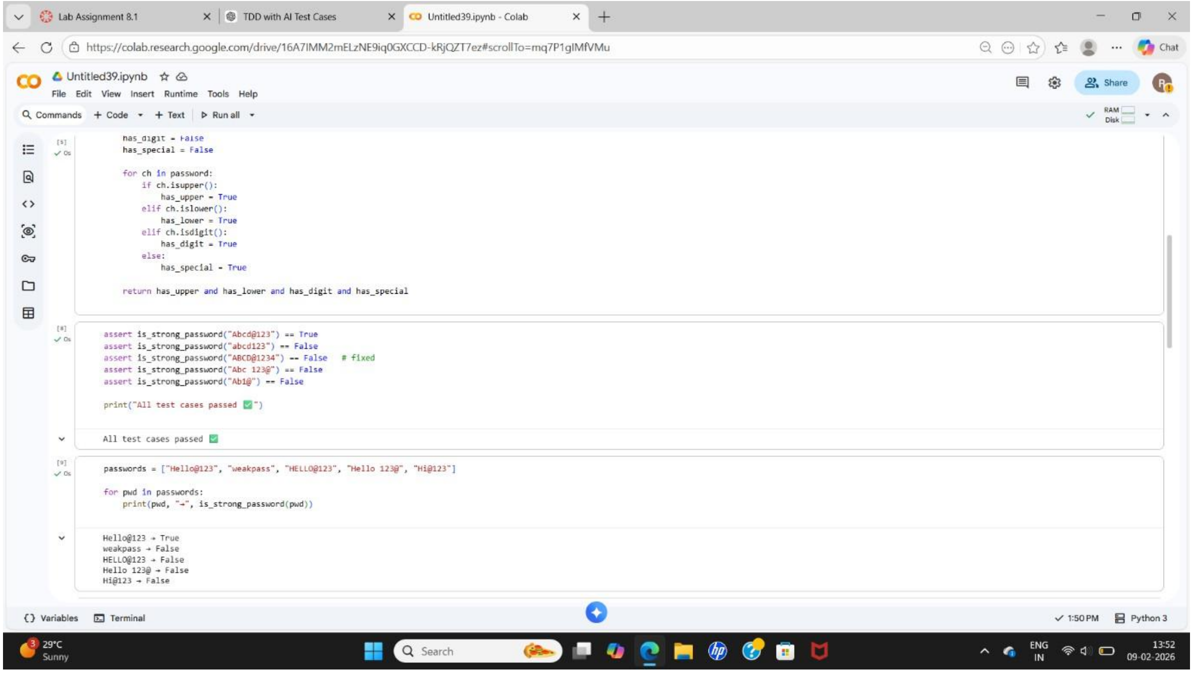


```
# Test cases (AI-generated)
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == True
assert is_strong_password("Abc 123@") == False
assert is_strong_password("Ab1@") == False
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-3794212921.py in <cell line: 0>()
      1 # Test cases (AI-generated)
----> 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

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
```



```
    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") == False   # 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@", "Hi@123"]

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

```
Hello@123 → True
weakpass → False
HELLO@123 → False
Hello 123@ → False
Hi@123 → False
```

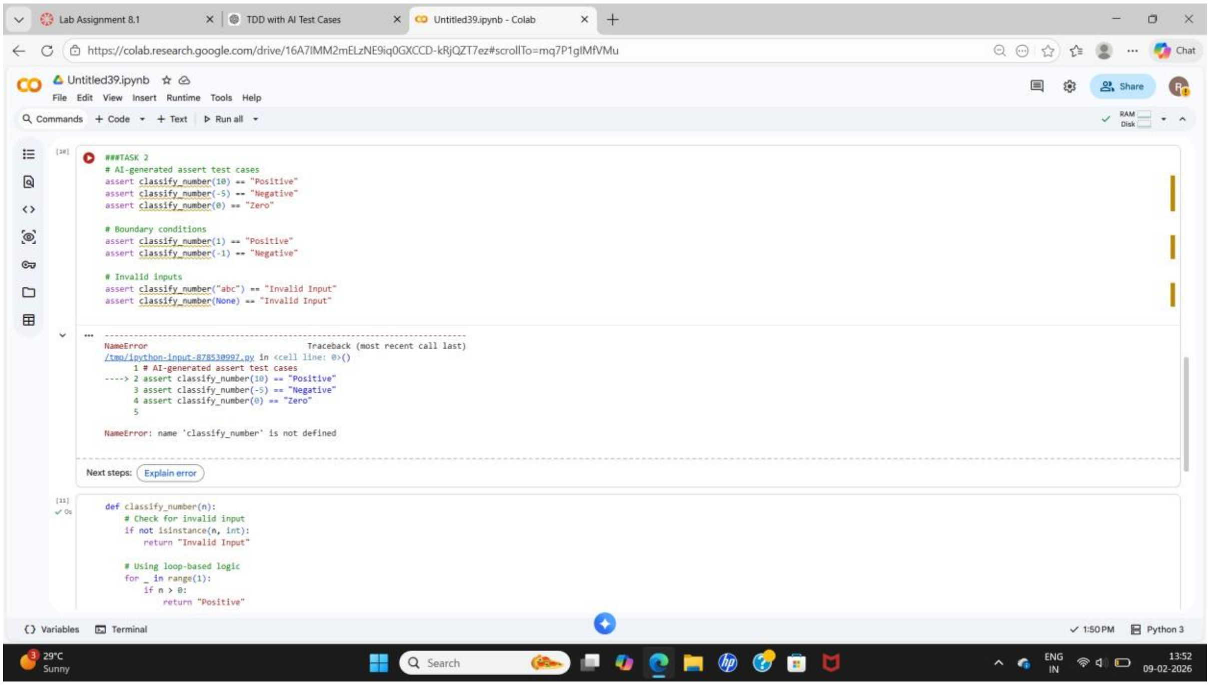
Code Explanation:

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

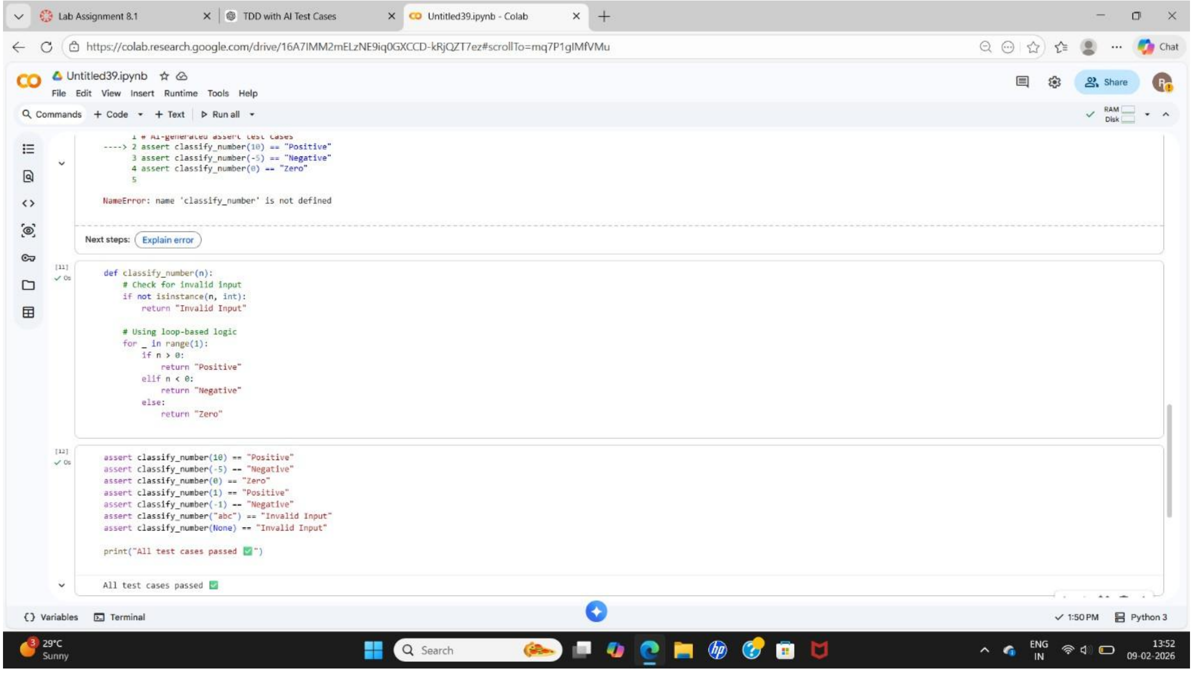


```
###TASK 2
# AI-generated assert test cases
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"

# 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-878530997.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"
```



```
      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"
        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 ✅
```

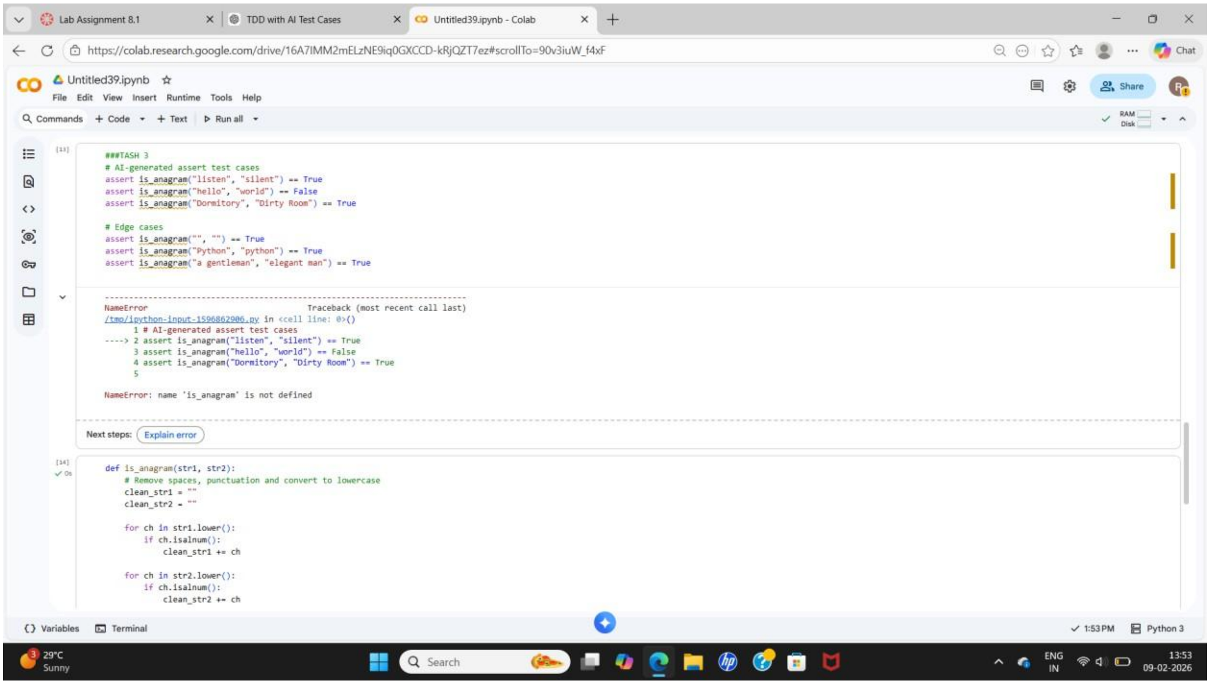
Code Explanation:

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.

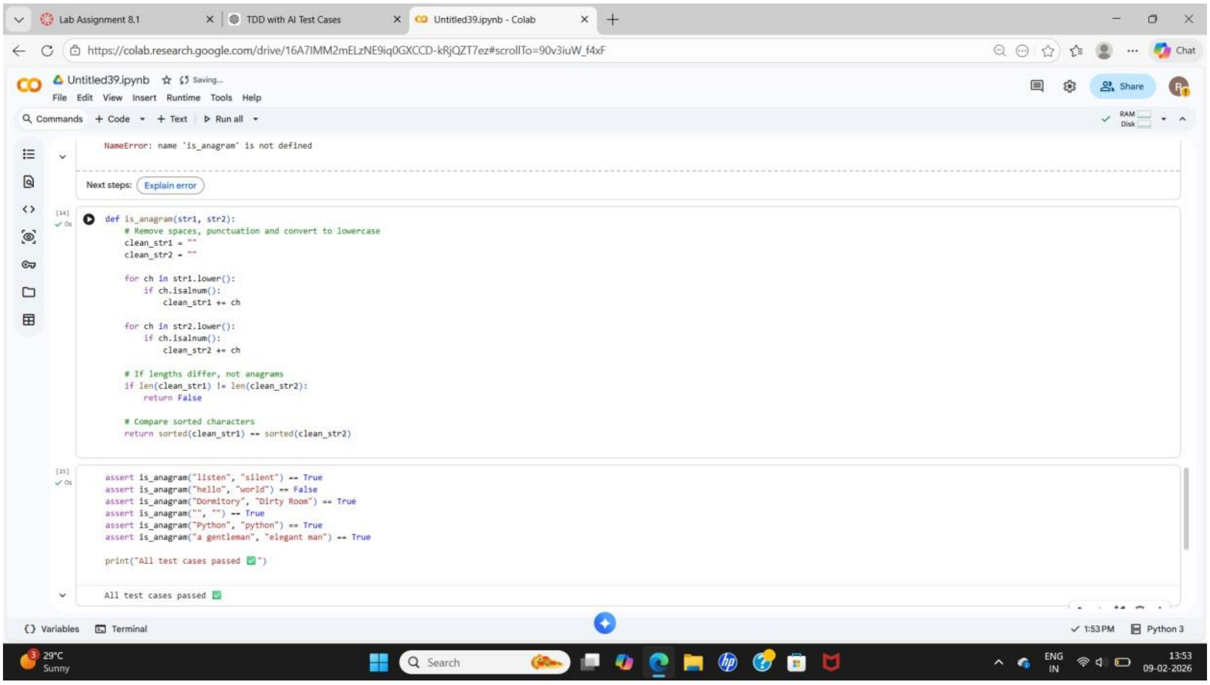
## Task 3: Anagram Checker


Lab Assignment 8.1
TDD with AI Test Cases
Untitled39.ipynb - Colab
https://colab.research.google.com/drive/16A7IMM2mELzNE9iq0GXCCD-kRjQZT7ez#scrollTo=90v3iuW_f4xF
Untitled39.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk
[13]
Next steps: Explain error
[14]
✓ 0s
Variables Terminal
1:53 PM Python 3
29°C Sunny
Search
ENG IN
13:53 09-02-2026

```
###TASH 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/ipython-input-1596862906.py in <cell line: 0>()
      1 # AI-generated assert test cases
----> 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
```

```
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
```



```
NameError: name 'is_anagram' is not defined
```

Next steps: Explain error

[14]

```python
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)
```

[15]

```python
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 ✅")
```

```
All test cases passed ✅
```

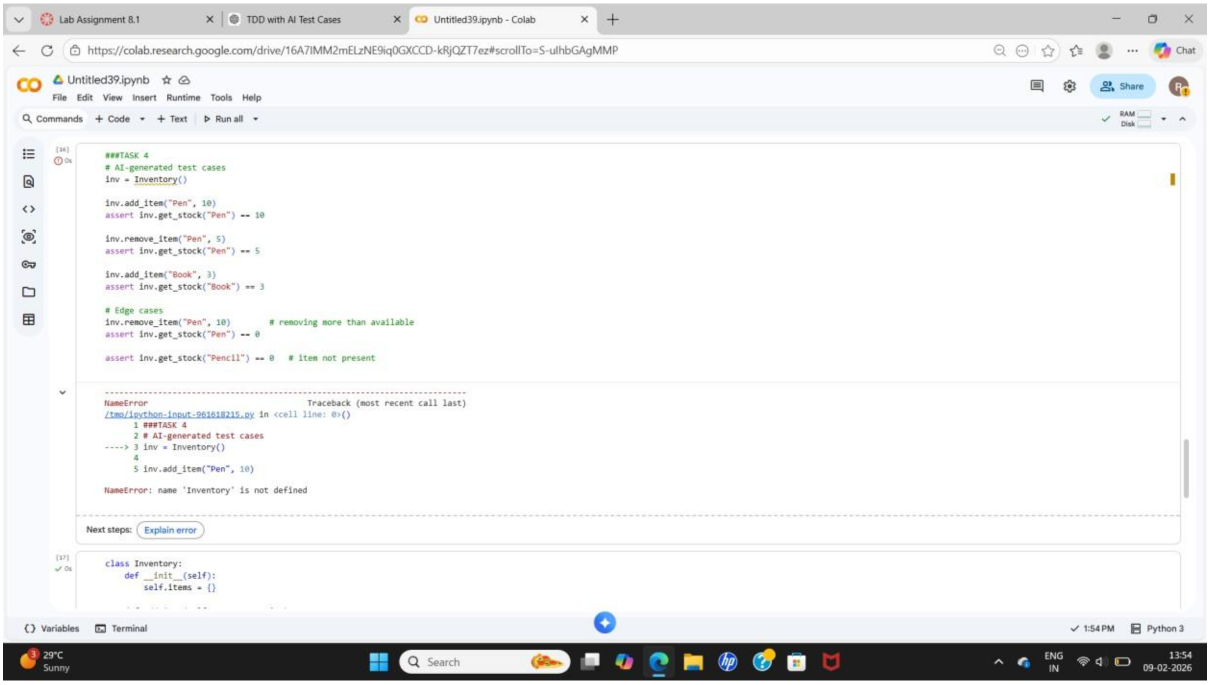
Code Explanation:

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.

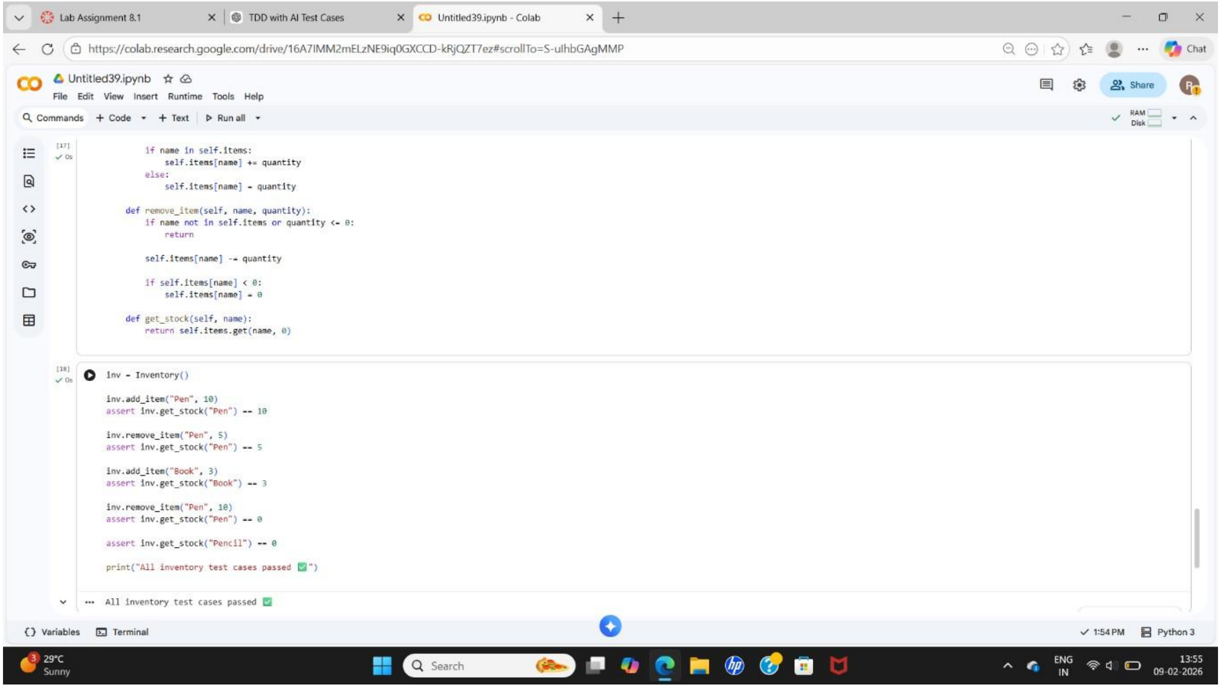
## Task 4: Inventory Class (Real-World Simulation)



```
###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.py 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
```

```
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 ✅")
```

```
All inventory test cases passed ✅
```

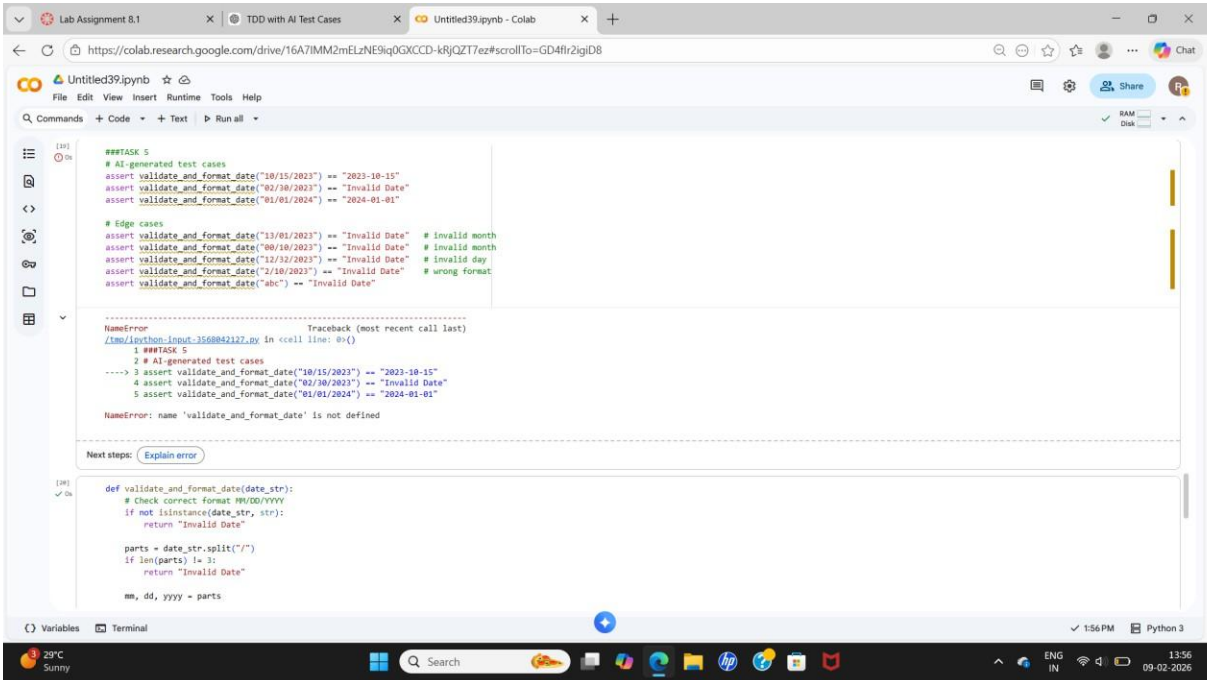
Code Explanation:

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

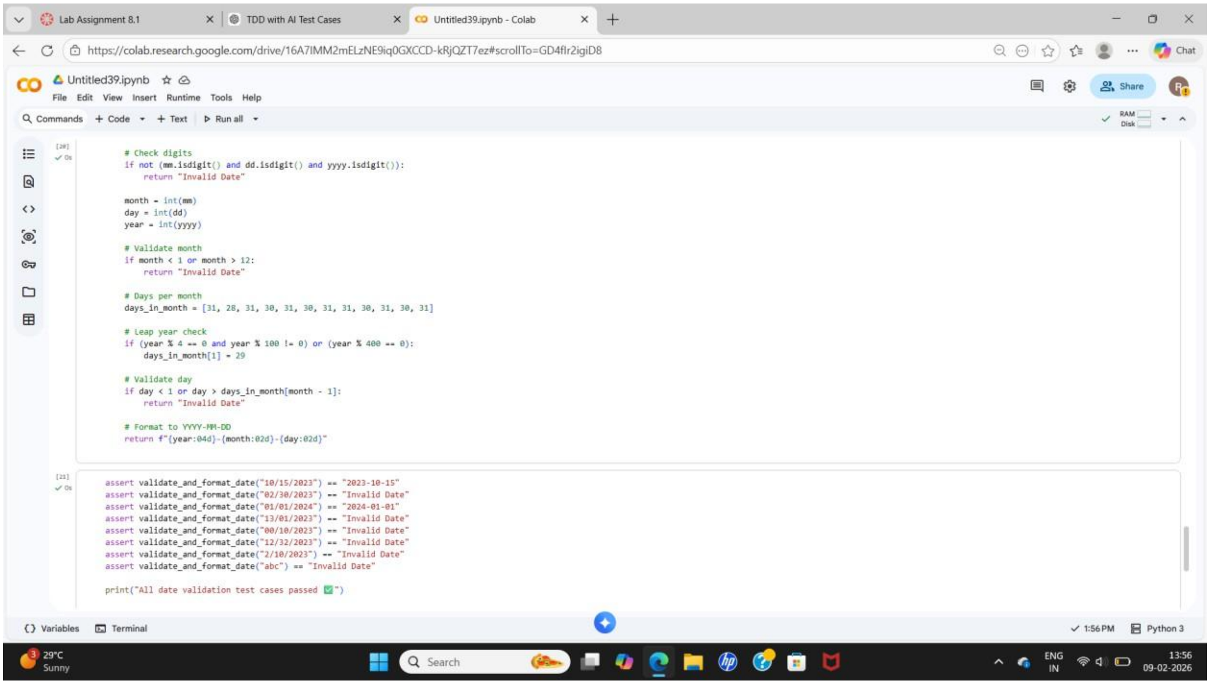


Lab Assignment 8.1
TDD with AI Test Cases
Untitled39.ipynb - Colab
https://colab.research.google.com/drive/16A7IMM2mELzNE9iq0GXCCD-kRjQZT7ez#scrollTo=GD4flr2igiD8
Untitled39.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
[19]
Next steps: Explain error
[20]
Variables Terminal
1:56 PM Python 3
29°C Sunny
Search
ENG IN
13:56 09-02-2026

```
###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-3568042127.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
```

```
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
```



Lab Assignment 8.1
TDD with AI Test Cases
Untitled39.ipynb - Colab
https://colab.research.google.com/drive/16A7IMM2mELzNE9iq0GXCCD-kRjQZT7ez#scrollTo=GD4flr2igiD8
Untitled39.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
[20]
[21]
Variables Terminal
1:56 PM Python 3
29°C Sunny
Search
ENG IN
13:56 09-02-2026

```
    # 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}"
```

```
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.