

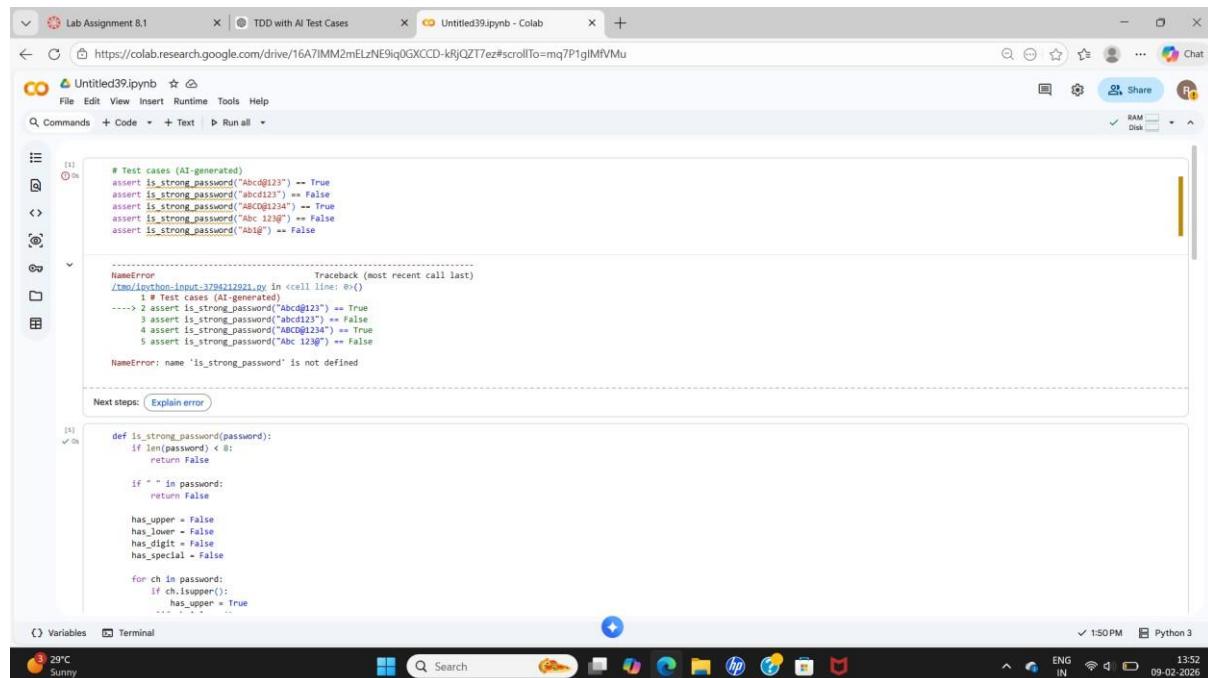
# LAB ASSIGNMENT 8.1

NAME: A.Harshita

ID.NO: 2303A52211

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("ABCQD@1234") == True
assert is_strong_password("Abc 123@") == False
assert is_strong_password("Ab1g") == False

-----
NameError: name 'is_strong_password' is not defined

Next steps: Explain error

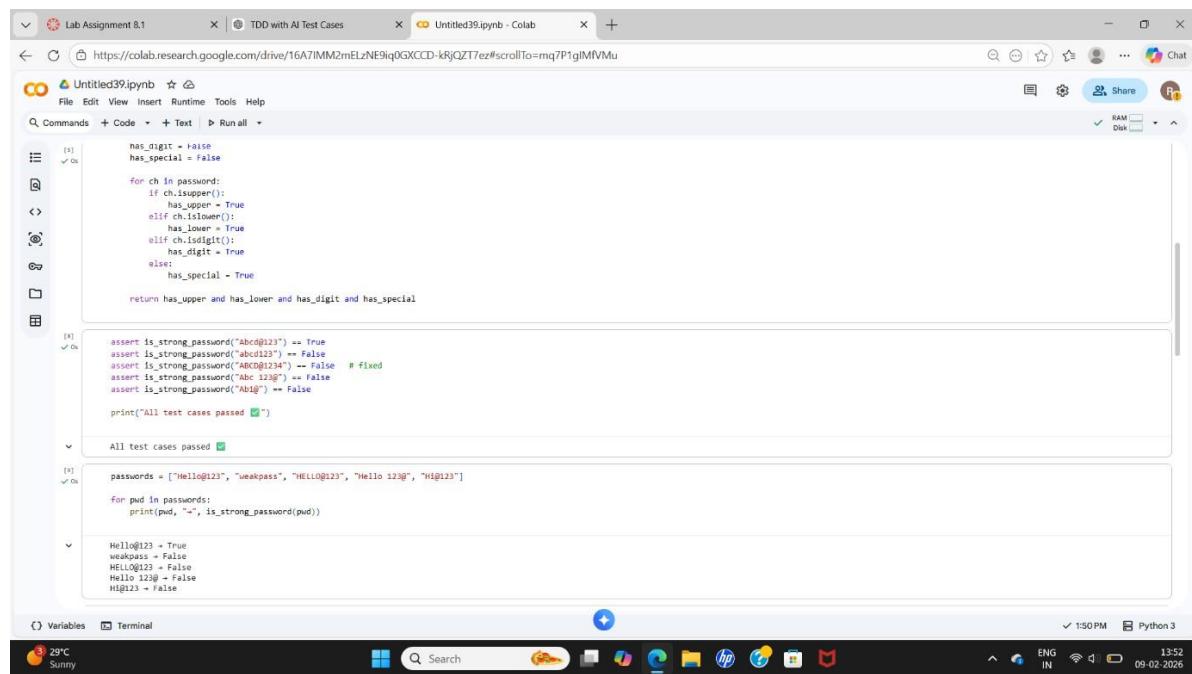
def is_strong_password(password):
    if len(password) < 8:
        return False

    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("Abcd@123") == False # fixed
assert is_strong_password("Abc 123@") == True
assert is_strong_password("Ab1g") == 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

The screenshot shows a Google Colab notebook titled "Untitled39.ipynb". The code defines a function `classify_number(n)` and includes several assert statements to check its correctness. The code is as follows:

```
##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: name 'classify_number' is not defined
```

Below the code, there is a section titled "Next steps: Explain error" which contains a partially completed solution for the `classify_number` function. The solution uses a loop-based logic to determine if the number is positive, negative, or zero. The code is as follows:

```
[11] 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 Google Colab notebook after the AI-generated test cases have been run. The output cell (cell 12) shows the results of the assert statements, confirming that all test cases passed. The code is identical to the one in the previous screenshot, except for the additional output from the assert statements.

```
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")
```

The output cell shows the message "All test cases passed".

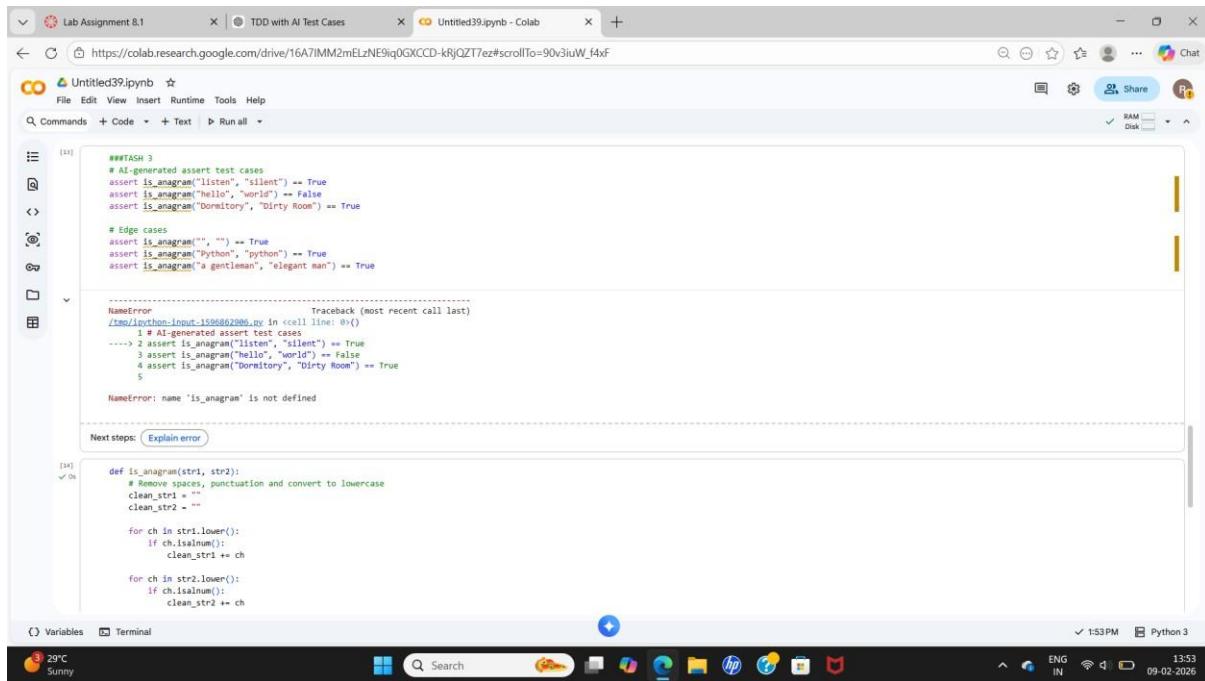
## Code Explanation:

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



The screenshot shows a Google Colab notebook titled "Untitled39.ipynb". The code cell contains the following Python code:

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

The cell shows a NameError: name 'is\_anagram' is not defined. Below the code cell, the code is displayed again:

```
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 notebook interface includes a toolbar at the top, a sidebar on the left, and a status bar at the bottom showing the date and time.

The screenshot shows a Google Colab notebook titled "Untitled39.ipynb". The code defines a function `is\_anagram` that takes two strings, `str1` and `str2`. It removes spaces and punctuation, converts the strings to lowercase, and then compares their sorted character counts. The notebook also includes a series of assertions to check the function's correctness across various scenarios, including empty strings and strings with different lengths.

```
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)

The screenshot shows a Google Colab notebook titled "Untitled39.ipynb". The code cell contains AI-generated test cases for an `Inventory` class. It includes assertions for adding items, removing items, and checking stock levels. A NameError is shown for the variable 'Inv'. The output cell shows the error message and the code for the `Inventory` class. The status bar at the bottom indicates it's running on Python 3.

```
##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: name 'Inventory' is not defined
/Untitled39.ipynb:1: in <cell line: 6()
  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 )

[17]
class Inventory:
    def __init__(self):
        self.items = {}

    def add_item(self, name, quantity):
        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)
```

The screenshot shows the same Google Colab notebook after the class definition was added. The code cell now contains the complete `Inventory` class with methods for adding, removing, and getting stock levels. The output cell shows the results of the test cases, including a success message. The status bar at the bottom indicates it's running on Python 3.

```
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)

[18]
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 Google Colab notebook titled "Untitled39.ipynb". The code cell contains AI-generated test cases for validating and formatting dates. It includes assertions for valid dates like "10/15/2023" and invalid ones like "02/30/2023". It also handles edge cases such as "13/01/2023" and "abc". A NameError is shown for the undefined function 'validate\_and\_format\_date'. The code cell below defines the function:

```
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 environment shows it's running on Python 3 at 1:56 PM on 09-02-2026.

The screenshot shows the same Google Colab notebook after the function has been implemented. The code cell now contains the complete implementation of the `validate_and_format_date` function, including checks for digit validity, month range, leap year, and day of month. The function returns the date in YYYY-MM-DD format or an error message. The code cell below shows the final implementation:

```
# 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:{4}d)-(month:{2}d)-(day:{2}d)"
```

The code cell above the function body contains assertions for various dates, and the final cell shows the output "All date validation test cases passed". The environment shows it's running on Python 3 at 1:56 PM on 09-02-2026.

### 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.

