

Assignment-8.1

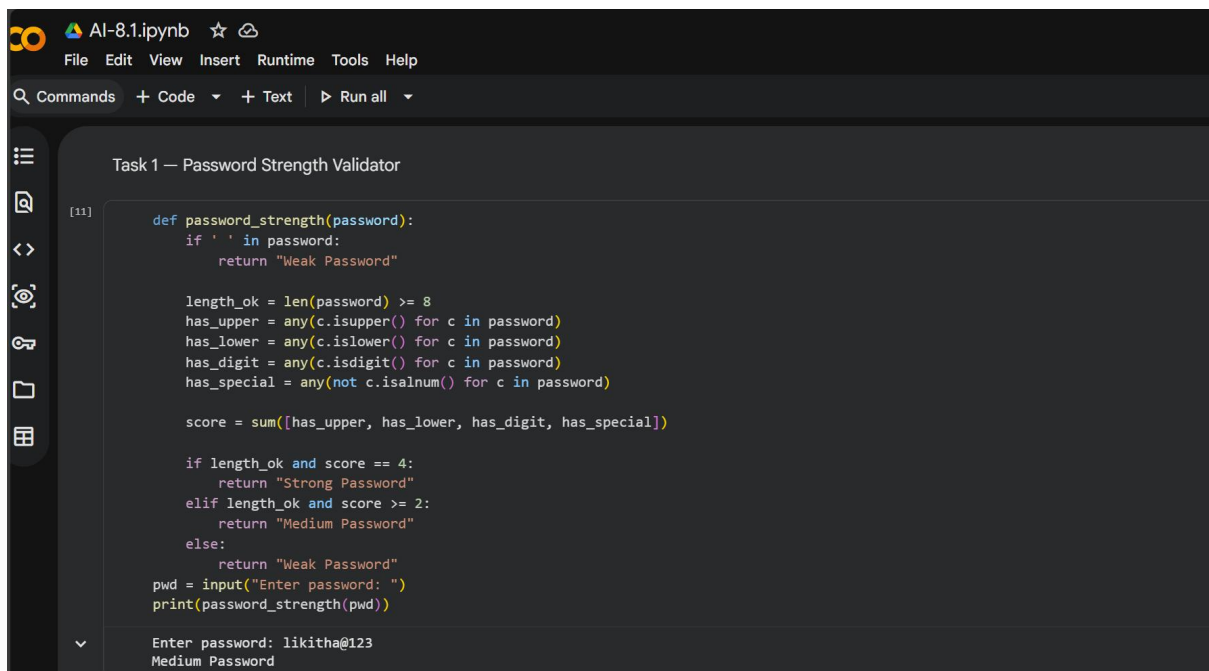
NAME:-pendli Likitha

HT NO:-2303A52393

BATCH:-43

Prompt for Task 1 — Password Strength Validator

Create a Python function to classify passwords as Weak, Medium, or Strong. A strong password must be at least 8 characters long and include uppercase, lowercase, digit, and special character with no spaces. A medium password meets length and any 2–3 conditions. Otherwise, it is weak. Generate at least 3 assert test cases and show input–output examples.



```
AI-8.1.ipynb ☆ ☁
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text ▶ Run all
Task 1 — Password Strength Validator

[11]
def password_strength(password):
    if ' ' in password:
        return "Weak Password"

    length_ok = len(password) >= 8
    has_upper = any(c.isupper() for c in password)
    has_lower = any(c.islower() for c in password)
    has_digit = any(c.isdigit() for c in password)
    has_special = any(not c.isalnum() for c in password)

    score = sum([has_upper, has_lower, has_digit, has_special])

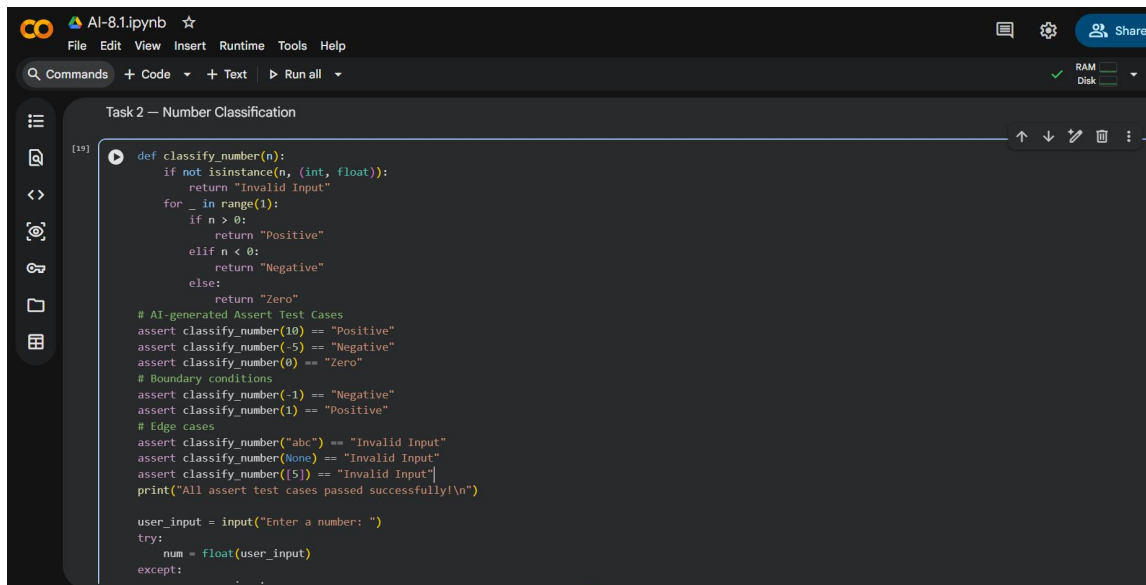
    if length_ok and score == 4:
        return "Strong Password"
    elif length_ok and score >= 2:
        return "Medium Password"
    else:
        return "Weak Password"

pwd = input("Enter password: ")
print(password_strength(pwd))

Enter password: likitha@123
Medium Password
```

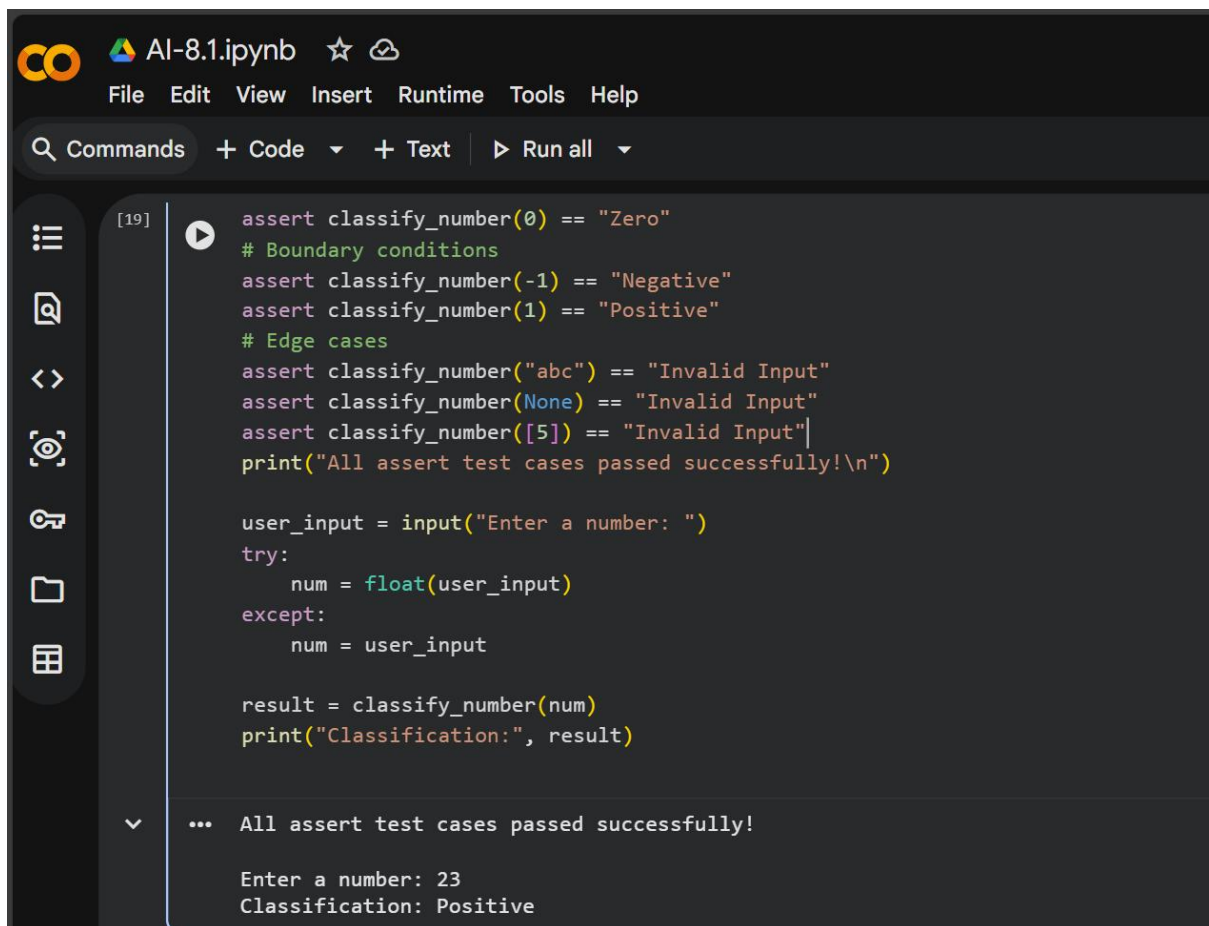
Prompt for Task 2 — Number Classification

Write a Python function classify_number(n) using loops to classify a number as Positive, Negative, or Zero. Handle invalid inputs like strings and None by returning "Invalid Input". Include boundary conditions such as -1, 0, and 1. Generate at least 3 assert test cases. Also display output using user input.



Task 2 — Number Classification

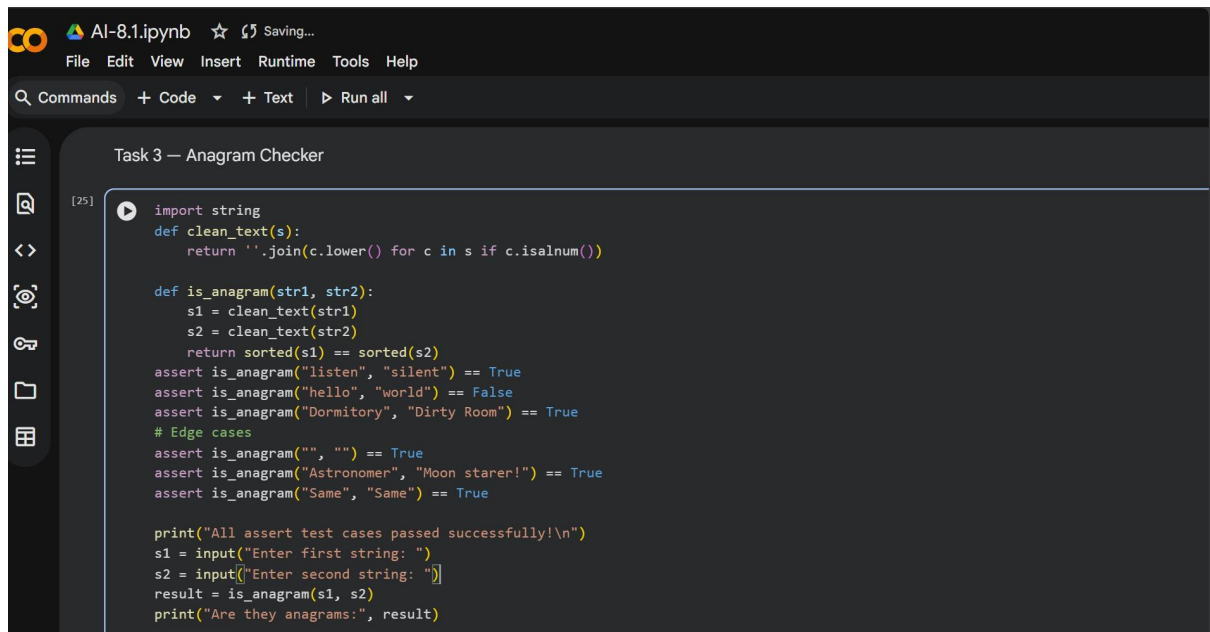
```
[19] def classify_number(n):  
    if not isinstance(n, (int, float)):  
        return "Invalid Input"  
    for _ in range(1):  
        if n > 0:  
            return "Positive"  
        elif n < 0:  
            return "Negative"  
        else:  
            return "Zero"  
    # AI-generated Assert Test Cases  
    assert classify_number(10) == "Positive"  
    assert classify_number(-5) == "Negative"  
    assert classify_number(0) == "Zero"  
    # Boundary conditions  
    assert classify_number(-1) == "Negative"  
    assert classify_number(1) == "Positive"  
    # Edge cases  
    assert classify_number("abc") == "Invalid Input"  
    assert classify_number(None) == "Invalid Input"  
    assert classify_number([5]) == "Invalid Input"  
    print("All assert test cases passed successfully!\n")  
  
    user_input = input("Enter a number: ")  
    try:  
        num = float(user_input)  
    except:
```



```
[19] assert classify_number(0) == "Zero"  
# Boundary conditions  
assert classify_number(-1) == "Negative"  
assert classify_number(1) == "Positive"  
# Edge cases  
assert classify_number("abc") == "Invalid Input"  
assert classify_number(None) == "Invalid Input"  
assert classify_number([5]) == "Invalid Input"  
print("All assert test cases passed successfully!\n")  
  
user_input = input("Enter a number: ")  
try:  
    num = float(user_input)  
except:  
    num = user_input  
  
result = classify_number(num)  
print("Classification:", result)  
  
... All assert test cases passed successfully!  
  
Enter a number: 23  
Classification: Positive
```

Prompt for Task 3 — Anagram Checker

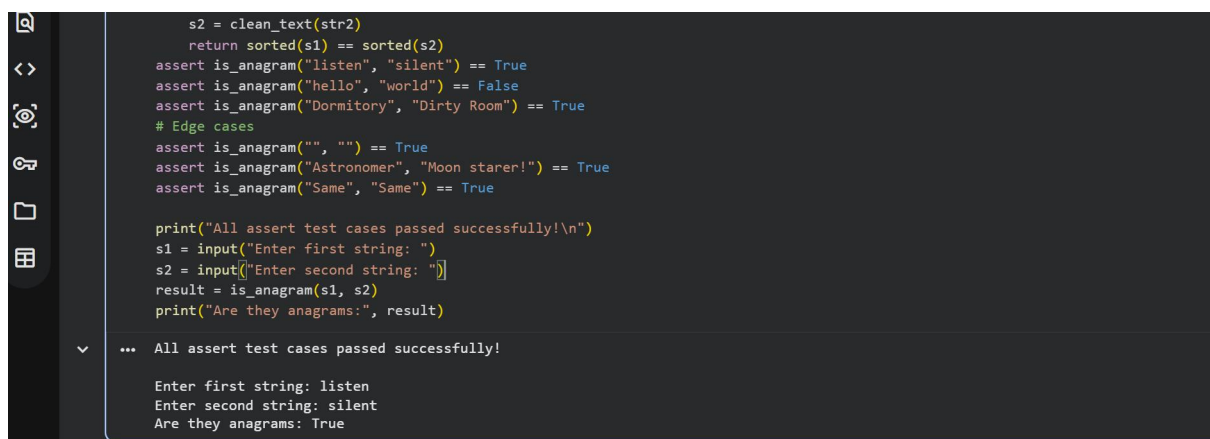
Create a Python function `is_anagram(str1, str2)` to check if two strings are anagrams. Ignore case, spaces, and punctuation while comparing. Handle edge cases like empty strings and identical words. Generate at least 3 assert test cases. Show input and output examples.



```
AI-8.1.ipynb ☆ Saving...
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text ▶ Run all
Task 3 — Anagram Checker
[25]
import string
def clean_text(s):
    return ''.join(c.lower() for c in s if c.isalnum())

def is_anagram(str1, str2):
    s1 = clean_text(str1)
    s2 = clean_text(str2)
    return sorted(s1) == sorted(s2)
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("Astronomer", "Moon starrer!") == True
assert is_anagram("Same", "Same") == True

print("All assert test cases passed successfully!\n")
s1 = input("Enter first string: ")
s2 = input("Enter second string: ")
result = is_anagram(s1, s2)
print("Are they anagrams:", result)
```



```
s2 = clean_text(str2)
return sorted(s1) == sorted(s2)
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("Astronomer", "Moon starrer!") == True
assert is_anagram("Same", "Same") == True

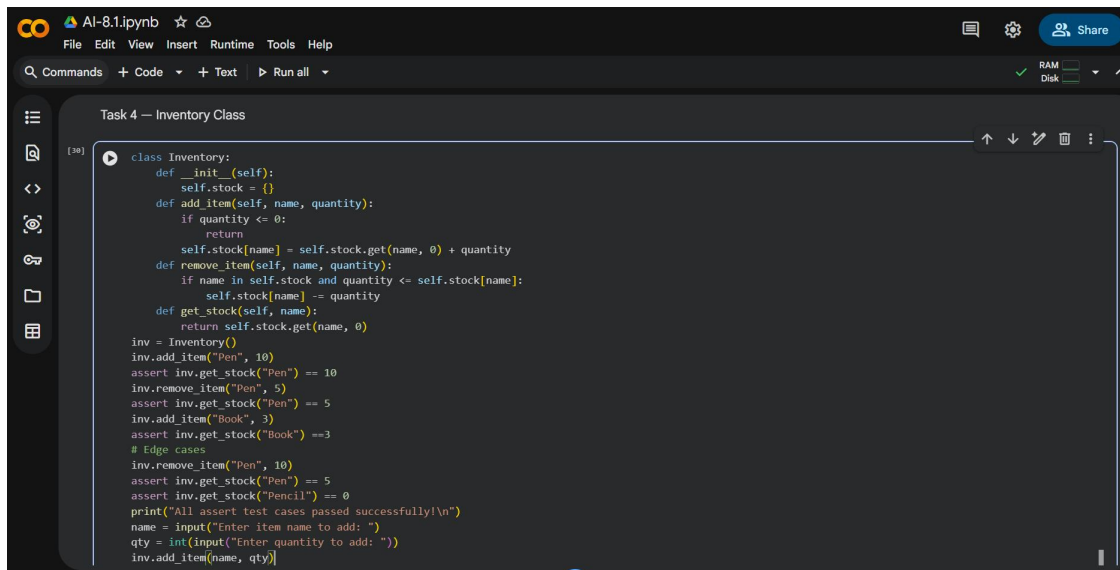
print("All assert test cases passed successfully!\n")
s1 = input("Enter first string: ")
s2 = input("Enter second string: ")
result = is_anagram(s1, s2)
print("Are they anagrams:", result)

... All assert test cases passed successfully!

Enter first string: listen
Enter second string: silent
Are they anagrams: True
```

Prompt for Task 4 — Inventory Class

Design a Python class `Inventory` to manage stock items. Include methods: `add_item(name, quantity)`, `remove_item(name, quantity)`, and `get_stock(name)`. Ensure stock never becomes negative and handle missing items. Generate at least 3 assert-based test cases. Demonstrate functionality with sample input and output.



```
class Inventory:
    def __init__(self):
        self.stock = {}
    def add_item(self, name, quantity):
        if quantity <= 0:
            return
        self.stock[name] = self.stock.get(name, 0) + quantity
    def remove_item(self, name, quantity):
        if name in self.stock and quantity <= self.stock[name]:
            self.stock[name] -= quantity
    def get_stock(self, name):
        return self.stock.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
# Edge cases
inv.remove_item("Pen", 10)
assert inv.get_stock("Pen") == 0
assert inv.get_stock("Pencil") == 0
print("All assert test cases passed successfully!\n")
name = input("Enter item name to add: ")
qty = int(input("Enter quantity to add: "))
inv.add_item(name, qty)
```



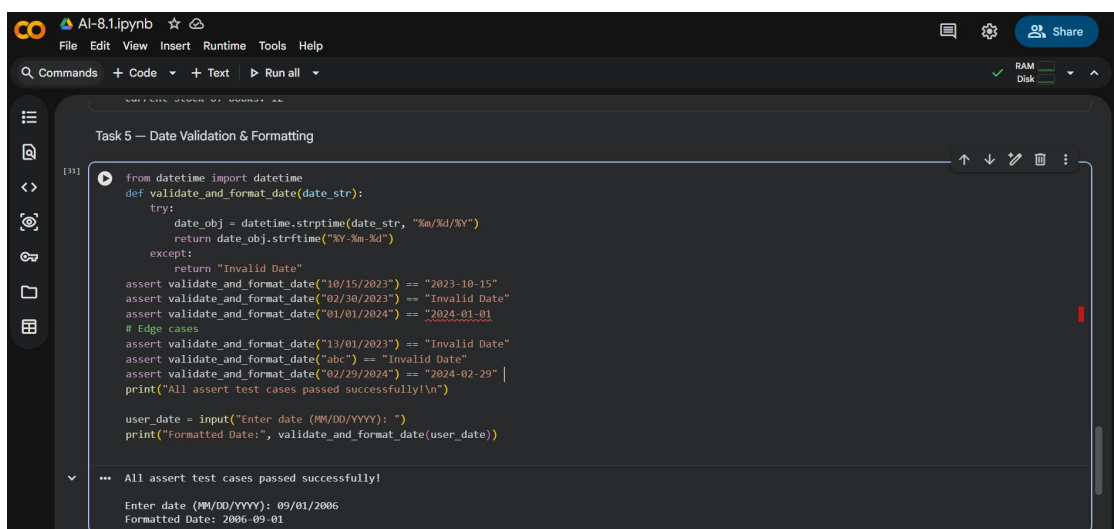
```
assert inv.get_stock("Book") == 3
# Edge cases
inv.remove_item("Pen", 10)
assert inv.get_stock("Pen") == 0
assert inv.get_stock("Pencil") == 0
print("All assert test cases passed successfully!\n")
name = input("Enter item name to add: ")
qty = int(input("Enter quantity to add: "))
inv.add_item(name, qty)
print(f"Current stock of {name}: {inv.get_stock(name)}")

... All assert test cases passed successfully!

Enter item name to add: books
Enter quantity to add: 12
Current stock of books: 12
```

Prompt for Task 5 — Date Validation & Formatting

Write a Python function `validate_and_format_date(date_str)` to validate dates in "MM/DD/YYYY" format. Convert valid dates to "YYYY-MM-DD" and return "Invalid Date" for wrong inputs. Handle invalid dates, formats, and leap year cases. Generate at least 3 assert test cases. Display results using user input.



```
from datetime import datetime
def validate_and_format_date(date_str):
    try:
        date_obj = datetime.strptime(date_str, "%m/%d/%Y")
        return date_obj.strftime("%Y-%m-%d")
    except:
        return "Invalid Date"

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"
assert validate_and_format_date("abc") == "Invalid Date"
assert validate_and_format_date("02/29/2024") == "2024-02-29"
print("All assert test cases passed successfully!\n")

user_date = input("Enter date (MM/DD/YYYY): ")
print("Formatted Date:", validate_and_format_date(user_date))

... All assert test cases passed successfully!

Enter date (MM/DD/YYYY): 09/01/2006
Formatted Date: 2006-09-01
```