

# Ai Ass 8.1

Name:B.Sanjana

H.T.NO:2303A52306

Batch:43

## TASK1:

"Generate Python code using the Test-Driven Development (TDD) approach for a function `is_strong_password(password)`.

First, create at least three assert test cases.

Then implement the function with the following requirements:

- Password must contain at least 8 characters.
- It must include at least one uppercase letter, one lowercase letter, one digit, and one special character.
- Password must not contain spaces.

After implementing the function, ensure all test cases pass and allow the user to enter a password to check whether it is strong or weak."

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Ai Ass 8.1.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help
- Code Cell:** [20] 6s
- Code Content:**

```
import string
def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    has_upper = any(ch.isupper() for ch in password)
    has_lower = any(ch.islower() for ch in password)
    has_digit = any(ch.isdigit() for ch in password)
    has_special = any(ch in string.punctuation for ch in password)
    return has_upper and has_lower and has_digit and has_special

# AI-generated test cases
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == False
print("All test cases passed")
# Taking input from user
password = input("Enter password: ")
if is_strong_password(password):
    print("Strong Password")
else:
    print("Weak Password")
```
- Output Cell:** All test cases passed  
Enter password: Sanju@11  
Strong Password

## TASK2:

"Generate Python code using the Test-Driven Development (TDD) approach for a function `classify_number(n)`.

First, create at least three assert test cases including boundary values (-1, 0, 1) and invalid inputs such as strings and None.

Then implement the function using loops to classify the number as:

- 'Positive'
- 'Negative'
- 'Zero'
- 'Invalid Input' for incorrect data types.

Ensure the program passes all assert test cases."

A screenshot of a Jupyter Notebook interface titled "Ai Ass 8.1.ipynb". The code cell contains Python code for classifying numbers and running assertions. The output shows the test cases passed.

```
[13]: def classify_number(n):
    if not isinstance(n, (int, float)):
        return "Invalid Input"
    # using loop
    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"
assert classify_number(-1) == "Negative"
assert classify_number(1) == "Positive"
print("All test cases passed")
# User input
user_input = input("Enter a number: ")
try:
    num = float(user_input)
except:
    num = user_input
result = classify_number(num)
print("Result:", result)
```

All test cases passed

A screenshot of a Jupyter Notebook interface titled "Ai Ass 8.1.ipynb". The code cell contains the same Python classification code. The output shows the user input "10" and the resulting classification "Positive".

```
[13]: def classify_number(n):
    if not isinstance(n, (int, float)):
        return "Invalid Input"
    # using loop
    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"
assert classify_number(-1) == "Negative"
assert classify_number(1) == "Positive"
print("All test cases passed")
# User input
user_input = input("Enter a number: ")
try:
    num = float(user_input)
except:
    num = user_input
result = classify_number(num)
print("Result:", result)
```

All test cases passed  
Enter a number: 10  
Result: Positive

### TASK3:

"Generate Python code using the Test-Driven Development (TDD) approach for a function `is_anagram(str1, str2)`.

First create at least three assert test cases including edge cases such as empty strings and identical words.

Then implement the function so that:

- Case, spaces, and punctuation are ignored.
- The function returns True if the strings are anagrams, otherwise False.

Ensure the program passes all assert test cases."

The screenshot shows a Jupyter Notebook interface with the following code in cell #Task3:

```
import string
def is_anagram(str1, str2):
    clean1 = ''.join(ch.lower() for ch in str1 if ch.isalnum())
    clean2 = ''.join(ch.lower() for ch in str2 if ch.isalnum())
    return sorted(clean1) == sorted(clean2)

# AI-generated assert test cases
assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True
assert is_anagram("", "") == True
print("All test cases passed")

# User input
s1 = input("Enter first string: ")
s2 = input("Enter second string: ")
if is_anagram(s1, s2):
    print("true")
else:
    print("False")
```

Cell #Task4 is visible below it.

### Task4:

"Generate Python code using the TDD approach for an Inventory class.

First create at least three assert-based test cases.

Then implement the class with the following methods:

- `add_item(name, quantity)`
- `remove_item(name, quantity)`
- `get_stock(name)`

Ensure the class correctly manages stock values and passes all assert test cases."

```

class Inventory:
    def __init__(self):
        self.stock = {}
    def add_item(self, name, quantity):
        self.stock[name] = self.stock.get(name, 0) + quantity
    def remove_item(self, name, quantity):
        if name in self.stock:
            self.stock[name] = max(0, self.stock[name] - quantity)
    def get_stock(self, name):
        return self.stock.get(name, 0)
# 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
print("All test cases passed")
inventory = Inventory()
name = input("Enter item name: ")
qty = int(input("Enter quantity to add: "))
inventory.add_item(name, qty)
print("Current stock of", name, ":", inventory.get_stock(name))

```

...  
All test cases passed  
Enter item name: Book  
Enter quantity to add: 10  
Current stock of Book : 10

## TASK5:

"Generate Python code using the TDD approach for a function validate\_and\_format\_date(date\_str).

First create at least three assert test cases including valid and invalid dates.

Then implement the function so that:

- It validates the date in MM/DD/YYYY format.
- Returns Invalid Date for incorrect inputs.
- Converts valid dates to YYYY-MM-DD format.

Ensure all assert test cases pass."

```

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"
# AI-generated 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"
print("All test cases passed")
date_input = input("Enter date (MM/DD/YYYY): ")
result = validate_and_format_date(date_input)
print("Output:", result)

```

...  
All test cases passed  
Enter date (MM/DD/YYYY): 11/01/2005  
Output: 2005-11-01