

# Lab Assignment-08

Name: P. Chandra Vardhan Reddy

Hallticket:2303A51034

Batch-01

## Task Description #1 (Password Strength Validator – Apply AI in Security Context)

- Task: Apply AI to generate at least 3 assert test cases for `is_strong_password(password)` and implement the validator function.
- Requirements:
  - Password must have at least 8 characters.
  - Must include uppercase, lowercase, digit, and special character.
  - Must not contain spaces.

### Example Assert Test Cases:

```
assert is_strong_password("Abcd@123") == True  
assert is_strong_password("abcd123") == False assert  
is_strong_password("ABCD@1234") == True
```

Expected Output #1:

Password validation logic passing all AI-generated test cases.



```
assg_08.py  
1 def password_check(password):  
    if len(password) < 8:  
        return False  
    if not any(char.isupper() for char in password):  
        return False  
    if not any(char.islower() for char in password):  
        return False  
    if not any(char.isdigit() for char in password):  
        return False  
    if not any(char in "!@#$%^&*()-_=+[]{};':\",.<>?/" for char in password):  
        return False  
    return True
```

```

❶ assg_08.py > ...
1  def password_check(password):
2      if len(password) < 8:
3          return False
4      if not any(char.isupper() for char in password):
5          return False
6      if not any(char.islower() for char in password):
7          return False
8      if not any(char.isdigit() for char in password):
9          return False
10     if not any(char in "!@#$%^&*(-_=+[]{}|;:'\".,<>?/" for char in password):
11         return False
12     return True
13 #assert testcases
14 assert password_check("Password123!") == True
15 assert password_check("pass") == False
16 assert password_check("PASSWORD123") == False
17 assert password_check("password123") == False
18 assert password_check("Password") == False
19 assert password_check("Password123") == False
20 assert password_check("Password!") == False
21 assert password_check("12345678") == False
22 assert password_check("!@#$%^&*") == False
23 print("All test cases passed!")
24

```

### Code:

```

def password_check(password):
if len(password) < 8:
    return False
    if not any(char.isupper())
for char in password):
    return False
    if not any(char.islower())
for char in password):
    return False
    if not any(char.isdigit())
for char in password):
    return False
    if not any(char in "!@#$%^&*(-_=+[]{}|;:'\".,<>?/" for char in password):
        return False
    return True #assert
testcases assert
password_check("Password123!") == True assert
password_check("pass") == False assert
password_check("PASSWORD123") == False assert
password_check("password123") == False assert
password_check("Password") == False assert
password_check("Password123") == False assert
password_check("Password!") == False assert
password_check("12345678") == False assert
password_check("!@#$%^&*") == False
print("All test cases passed!")

```

### output:

```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_08.py"
All test cases passed!
PS C:\Users\arell\Music\aiac> []
```

## **Task Description #2 (Number Classification with Loops – Apply AI for Edge Case Handling)**

- Task: Use AI to generate at least 3 assert test cases for a classify\_number(n) function. Implement using loops.
- Requirements:
  - Classify numbers as Positive, Negative, or Zero.
  - Handle invalid inputs like strings and None.
  - Include boundary conditions (-1, 0, 1).

Example Assert Test Cases:

```
assert classify_number(10) == "Positive" assert
classify_number(-5) == "Negative" assert
classify_number(0) == "Zero"
```

Expected Output #2:

- Classification logic passing all assert tests.

```
25
26 def classify_number(n):
27     if n > 0:
28         return "Positive"
29     elif n < 0:
30         return "Negative"
31     else:
32         return "Zero"
33 assert classify_number(10) == "Positive"
```

```
25
26 def classify_number(n):
27     if n > 0:
28         return "Positive"
29     elif n < 0:
30         return "Negative"
31     else:
32         return "Zero"
33 assert classify_number(10) == "Positive"
34 assert classify_number(-5) == "Negative"
35 assert classify_number(0) == "Zero"
36 print("All test cases passed!")
```

Code:

```

def classify_number(n):
    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" print("All
test cases passed!")

```

output:

```

PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_08.py"
All test cases passed!
PS C:\Users\arell\Music\aiac> []

```

### Task Description #3 (Anagram Checker – Apply AI for String Analysis)

- Task: Use AI to generate at least 3 assert test cases for is\_anagram(str1, str2) and implement the function.
- Requirements:
  - Ignore case, spaces, and punctuation.
  - Handle edge cases (empty strings, identical words).

Example Assert Test Cases:

```

assert is_anagram("listen", "silent") == True assert
is_anagram("hello", "world") == False assert
is_anagram("Dormitory", "Dirty Room") == True

```

Expected Output #3:

- Function correctly identifying anagrams and passing all AI-generated tests.

```

38 def is_anagram(str1, str2):
    return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())

```

```

38 def is_anagram(str1, str2):
39     return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())
40 assert is_anagram("listen", "silent") == True
41 assert is_anagram("hello", "world") == False
42 assert is_anagram("Dormitory", "Dirty Room") == True
43 assert is_anagram("The eyes", "They see") == True
44 assert is_anagram("Astronomer", "Moon starer") == True
45 assert is_anagram("Conversation", "Voices rant on") == True
46 print(["All test cases passed!"])

```

Code:

```
def is_anagram(str1, str2):
    return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())
assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True
assert is_anagram("The eyes", "They see") == True
assert is_anagram("Astronomer", "Moon starer") == True
assert is_anagram("Conversation", "Voices rant on") == True
print("All test cases passed!")
```

Output:

```
All test cases passed!
PS C:\Users\arell\Music\aiac> []
```

**Task Description #4 (Inventory Class – Apply AI to Simulate Real-World Inventory System)**

- Task: Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.
- Methods:
  - add\_item(name, quantity) ◦
  - remove\_item(name, quantity) ◦
  - get\_stock(name)

Example 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
```

Expected Output #4:

- Fully functional class passing all assertions.

```

48  class inventory:
49      def __init__(self):
50          self.items = {}
51
52      def add_item(self, item, quantity):
53          if item in self.items:
54              self.items[item] += quantity
55          else:
56              self.items[item] = quantity
57
58      def remove_item(self, item, quantity):
59          if item in self.items:
60              if self.items[item] >= quantity:
61                  self.items[item] -= quantity
62                  if self.items[item] == 0:
63                      del self.items[item]
64              else:
65                  raise ValueError("Not enough items in inventory")
66          else:
67              raise ValueError("Item not found in inventory")
68
69      def get_quantity(self, item):
70          return self.items.get(item, 0)
71
72
73  inv = inventory()
74
75  inv.add_item("apple", 10)
76  assert inv.get_stock("apple") == 10
77
78  inv.add_item("banana", 5)
79  assert inv.get_stock("banana") == 5
80
81  inv.remove_item("apple", 3)
82  assert inv.get_stock("apple") == 7
83
84  inv.remove_item("banana", 5)
85  assert inv.get_stock("banana") == 0
86

```

```

48  class inventory:
49      def __init__(self):
50          self.items = {}
51
52      def add_item(self, item, quantity):
53          if item in self.items:
54              self.items[item] += quantity
55          else:
56              self.items[item] = quantity
57
58      def remove_item(self, item, quantity):
59          if item in self.items:
60              if self.items[item] >= quantity:
61                  self.items[item] -= quantity
62                  if self.items[item] == 0:
63                      del self.items[item]
64              else:
65                  raise ValueError("Not enough quantity to remove")
66          else:
67              raise ValueError("Item not found in inventory")
68
69      def get_stock(self, item):
70          return self.items.get(item, 0)
71
72
73  inv = inventory()
74
75  inv.add_item("apple", 10)
76  assert inv.get_stock("apple") == 10
77
78  inv.add_item("banana", 5)
79  assert inv.get_stock("banana") == 5
80
81  inv.remove_item("apple", 3)
82  assert inv.get_stock("apple") == 7
83
84  inv.remove_item("banana", 5)
85  assert inv.get_stock("banana") == 0
86

```

**Code:**

```

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

    def add_item(self, item, quantity):
if item in self.items:
        self.items[item] += quantity
else:
        self.items[item] = quantity

    def remove_item(self, item, quantity):
        if item in self.items:
if self.items[item] >= quantity:
    self.items[item] -= quantity
if self.items[item] == 0:
    del self.items[item]           else:
        raise ValueError("Not enough quantity to remove")

    else:
        raise ValueError("Item not found in inventory")

    def get_stock(self, item):
        return self.items.get(item, 0) inv
= inventory()

inv.add_item("apple", 10) assert
inv.get_stock("apple") == 10

inv.add_item("banana", 5) assert
inv.get_stock("banana") == 5

inv.remove_item("apple", 3) assert
inv.get_stock("apple") == 7

inv.remove_item("banana", 5) assert
inv.get_stock("banana") == 0

print("All test cases passed!")

```

**output:**

```

All test cases passed!
PS C:\Users\arell\Music\aiac> []

```

## Task Description #5 (Date Validation & Formatting – Apply AI for Data Validation)

- Task: Use AI to generate at least 3 assert test cases for validate\_and\_format\_date(date\_str) to check and convert dates.
- Requirements:
  - Validate "MM/DD/YYYY" format.
  - Handle invalid dates.
  - Convert valid dates to "YYYY-MM-DD".

Example 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"
```

Expected Output #5:

- Function passes all AI-generated assertions and handles edge cases.

```
87 def validate_and_format_date(date_str):  
88     import re  
89     from datetime import datetime  
90     pattern = r'^\d{2}/\d{2}/\d{4}$'  
91     if not re.match(pattern, date_str):  
         raise ValueError("Date must be in the format DD/MM/YYYY")
```

```
87 def validate_and_format_date(date_str):  
88     import re  
89     from datetime import datetime  
90     pattern = r'^\d{2}/\d{2}/\d{4}$'  
91     if not re.match(pattern, date_str):  
         return "Invalid Date"  
92     try:  
93         date_obj = datetime.strptime(date_str, "%m/%d/%Y")  
94         return date_obj.strftime("%Y-%m-%d")  
95     except ValueError:  
96         return "Invalid Date"  
97     assert validate_and_format_date("12/31/2020") == "2020-12-31"  
98     assert validate_and_format_date("31/12/2020") == "Invalid Date"  
99     assert validate_and_format_date("02/30/2020") == "Invalid Date"  
100    assert validate_and_format_date("01/01/2021") == "2021-01-01"  
101    assert validate_and_format_date("13/01/2020") == "Invalid Date"  
102  
103    print("All test cases passed!")
```

Code:

```
def validate_and_format_date(date_str):
    import re      from datetime import
datetime   pattern =
r'^\d{2}/\d{2}/\d{4}$'      if not
re.match(pattern, date_str):
    return "Invalid Date"
try:
    date_obj = datetime.strptime(date_str,
"%m/%d/%Y")           return date_obj.strftime("%Y-%m-%d")
except ValueError:
    return "Invalid Date" assert
validate_and_format_date("12/31/2020") == "2020-12-31" assert
validate_and_format_date("31/12/2020") == "Invalid Date" assert
validate_and_format_date("02/30/2020") == "Invalid Date" assert
validate_and_format_date("01/01/2021") == "2021-01-01" assert
validate_and_format_date("13/01/2020") == "Invalid Date"
print("All test cases passed!")
```

output:

```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_08.py"
All test cases passed!
PS C:\Users\arell\Music\aiac> []
```