ASSIGNMENT – 8.1

2303A51134

Batch-27

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.

```
#Apply AI to generate at least 3 assert test cases foris_strong_password(password) and implement the validator function
def is_strong_password(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 "!@#$%^C*()-_=+[]{}|;:\",.<>?/" for char in password):
    return False
  return True
# Assert test cases
assert is_strong_password("Password123!") == True, "Test case 1 failed"
assert is_strong_password("weakpass") == False, "Test case 2 failed"
assert is_strong_password("Short1!") == False, "Test case 3 failed"
print("All test cases passed!")
```

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

#Write a Python function classify_numbers(n) that classifies numbers from 1 to n as 'Even', 'Odd', or 'Prime'. Use loops to iterate through the numbers and apply AI techniques to handle edge cases, such as negative numbers or non-integer inputs. Provide the implementation of the function along with test cases.

```
def classify_numbers(n):
  if not isinstance(n, int) or n < 1:
    return "Input must be a positive integer."

  classifications = {}

  for num in range(1, n + 1):
    if num % 2 == 0:
      classifications[num] = 'Even'
    else:
      classifications[num] = 'Odd'

    if num > 1 and all(num % i != 0 for i in range(2, int(num**0.5) + 1)):
      classifications[num] += ' and Prime'

  return  classifications
# Test cases
print(classify_numbers(10))
print(classify_numbers(-5))
print(classify_numbers(5.5))
```

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:**

o Ignore case, spaces, and punctuation.

o Handle edge cases (empty strings, identical words).

```
import string
def is_anagram(str1, str2):
  # Remove spaces and punctuation, and convert to lowercase
  translator = str.maketrans('', '', string.punctuation + ' ')
  str1_cleaned = str1.translate(translator).lower()
  str2_cleaned = str2.translate(translator).lower()
  return sorted(str1_cleaned) == sorted(str2_cleaned)
# Assert test cases
assert is_anagram("Listen", "Silent") == True, "Test case 1 failed"
assert is_anagram("Triangle", "Integral") == True, "Test case 2 failed"
assert is_anagram("Apple", "Pabble") == False, "Test case 3 failed"
assert is_anagram("", "") == True, "Test case 4 failed"
assert is_anagram("Dormitory", "Dirty Room") == True, "Test case 5 failed"
print("All test cases passed!")
```

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:**

o add_item(name, quantity)

o remove_item(name, quantity)

o get_stock(name)

```
class Inventory:
  def __init__(self):
    self.stock = {}

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

  def remove_item(self, name, quantity):
    if name in self.stock and self.stock[name] >= quantity:
      self.stock[name] -= quantity
      return True
    return False

  def get_stock(self, name):
    return self.stock.get(name, 0)
# Assert test cases
inventory = Inventory()
inventory.add_item("Apple", 10)
assert inventory.get_stock("Apple") == 10, "Test case 1 failed"
inventory.remove_item("Apple", 5)
assert inventory.get_stock("Apple") == 5, "Test case 2 failed"
assert inventory.remove_item("Apple", 10) == False, "Test case 3 failed"
assert inventory.get_stock("Banana") == 0, "Test case 4 failed"
print("All test cases passed!")
```

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:**

o Validate "MM/DD/YYYY" format.

o Handle invalid dates.

o Convert valid dates to "YYYY-MM-DD".

```
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 ValueError:
    return "Invalid date format. Please use MM/DD/YYYY."
# Assert test cases
assert validate_and_format_date("12/31/2020") == "2020-12-31", "Test case 1 failed"
assert validate_and_format_date("02/30/2020") == "Invalid date format. Please use MM/DD/YYYY.",
"Test case 2 failed"
assert validate_and_format_date("2020/12/31") == "Invalid date format. Please use MM/DD/YYYY.",
"Test case 3 failed"
assert validate_and_format_date("01/01/2021") == "2021-01-01", "Test case 4 failed"
print("All test cases passed!")
```