ASS ASSIGNMENT-8.5

M.HARI KRISHNA

2203A52100

B-50

Task Description #1 (Username Validator – Apply AI in

Authentication Context)

• Task: Use AI to generate at least 3 assert test cases for a

function is_valid_username(username) and then implement the

function using Test-Driven Development principles.

• Requirements:

o Username length must be between 5

and 15 characters. o Must contain only

alphabets and digits. o Must not start with a

digit.

o No spaces allowed. Example Assert

Test Cases:

assert is_valid_username("User123") == True

assert is_valid_username("12User") == False assert

is_valid_username("Us er") == False

Expected Output #1:

- Username validation logic successfully passing all AI- generated test cases.

PROMPT:

Generate at least 3 assert-based test cases for a function is_valid_username(username).
Requirements:

- Length between 5 and 15 characters
- Only alphabets and digits
- Must not start with a digit
- No spaces allowed
  Then implement the function using Test-Driven Development principles.

Task Description #2 (Even–Odd & Type Classification – Apply

AI for Robust Input Handling)

- Task: Use AI to generate at least 3 assert test cases for a function classify_value(x) and implement it using conditional logic and loops.
- Requirements:

o If input is an integer, classify as "Even" or "Odd".

o If input is 0, return "Zero".

o If input is non-numeric, return "Invalid Input".

Example Assert Test Cases:

assert classify_value(8) == "Even" assert

classify_value(7) == "Odd" assert

classify_value("abc") == "Invalid Input"

Expected Output #2:

- Function correctly classifying values and passing all test cases.

PROMPT:

Generate at least 3 assert-based test cases for a function classify_value(x). Requirements:

- If integer → classify as "Even" or "Odd"
- If 0 → return "Zero"
- If non-numeric → return "Invalid Input"

  Implement using conditional logic and loops.

Task Description #3 (Palindrome Checker – Apply AI for

String Normalization)

- Task: Use AI to generate at least 3 assert test cases for a function is_palindrome(text) and implement the function.
- Requirements:

o Ignore case, spaces, and punctuation.

o Handle edge cases such as empty strings and single characters.

Example Assert Test Cases:

assert is_palindrome("Madam") == True assert

is_palindrome("A man a plan a canal Panama") == True

assert is_palindrome("Python") == False

Expected Output #3:

- Function correctly identifying palindromes and passing all

AI-generated tests.

PROMPT:

Generate at least 3 assert-based test cases for a function `is_palindrome(text)`. Requirements:

- Ignore case, spaces, punctuation
- Handle empty string and single character Implement the function accordingly.

Task Description #4 (BankAccount Class – Apply AI for

Object-Oriented Test-Driven Development)

- Task: Ask AI to generate at least 3 assert-based test cases

  for a BankAccount class and then implement the class.

- Methods:

o deposit(amount) o

withdraw(amount) o

get_balance()

Example Assert Test Cases: acc

= BankAccount(1000)

acc.deposit(500) assert

acc.get_balance() == 1500

acc.withdraw(300) assert

acc.get_balance() == 1200

Expected Output #4:

- Fully functional class that passes all AI-generated assertions.

PROMPT:

Generate at least 3 assert-based test cases for a `BankAccount` class. Methods:

- deposit(amount)
- withdraw(amount) • get_balance()

Implement the class ensuring it passes all tests.

Task Description #5 (Email ID Validation – Apply AI for Data Validation)

- Task: Use AI to generate at least 3 assert test cases for a function validate_email(email) and implement the function.
- Requirements:

o Must contain @ and .

o Must not start or end with special characters. o Should handle invalid formats gracefully.

Example Assert Test Cases:

assert validate_email("user@example.com") == True

assert validate_email("userexample.com") == False assert validate_email("@gmail.com") == False

Expected Output #5:

- Email validation function passing all AI-generated test cases and handling edge cases correctly.

PROMPT:

Generate at least 3 assert-based test cases for a function `validate_email(email)`. Requirements:

- Must contain @ and .
- Must not start or end with special characters
- Handle invalid formats gracefully

Implement the function accordingly.

CODE:

```
# 1
# ◇ Function Implementation

def is_valid_username(username):

    # Check if input is string
    if not isinstance(username, str):
        return False

    # Length check
    if len(username) < 5 or len(username) > 15:
        return False

    # No spaces allowed
    if " " in username:
        return False

    # Must not start with digit
    if username[0].isdigit():
        return False

    # Only alphabets and digits
    if not username.isalnum():
        return False

    return True

# ◇ Assert Test Cases (TDD Validation)

assert is_valid_username("User123") == True
assert is_valid_username("12User") == False
assert is_valid_username("Us er") == False
```

```
assert is_valid_username("User") == False
assert is_valid_username("U") == False
assert is_valid_username("User_123") == False
assert is_valid_username("ValidUser99") == True
assert
is_valid_username("ThisUsernameIsTooLong123") ==
False

print(" Username validation logic successfully passing
all test cases.")

2
# ◇ Assert Test Cases

# ◇ Function Implementation

def classify_value(x):
    # Check type using loop-style logic
    for _ in range(1):  # simple loop usage
        if not isinstance(x, int):
            return "Invalid Input"
        if x == 0:
            return "Zero"
        if x % 2 == 0:
            return "Even"
        else:
            return "Odd"
print("Value classification passed all tests")
assert classify_value(8) == "Even"
assert classify_value(7) == "Odd"
assert classify_value(0) == "Zero"
assert classify_value("abc") == "Invalid Input"
assert classify_value(15) == "Odd"
```

#3

```
import string
# Assert Test Cases
# Function Implementation
def is_palindrome(text):
    # Normalize string
    cleaned = ""
    for ch in text:
        if ch.isalnum():
            cleaned += ch.lower()
    # Check palindrome
    return cleaned == cleaned[::-1]
print("Palindrome tests passed successfully ")
assert is_palindrome("Madam") == True
assert is_palindrome("A man a plan a canal Panama")
== True
assert is_palindrome("Python") == False
assert is_palindrome("") == True
assert is_palindrome("a") == True
```

4

```
# ◇  Class Implementation
class BankAccount:
    def __init__(self, balance):
        self.balance = balance
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
    def withdraw(self, amount):
        if amount > 0 and amount <= self.balance:
            self.balance -= amount
    def get_balance(self):
        return self.balance
```

```
# ◇ Assert Test Cases
acc = BankAccount(1000)
acc.deposit(500)
assert acc.get_balance() == 1500
acc.withdraw(300)
assert acc.get_balance() == 1200
acc.withdraw(2000)  # invalid withdrawal
assert acc.get_balance() == 1200
acc.deposit(-100)  # invalid deposit
assert acc.get_balance() == 1200
print("BankAccount class passed all tests ")
```

5

```
# ◇ Function Implementation

def validate_email(email):
    # Must contain exactly one @
    if email.count("@") != 1:
        return False

    local, domain = email.split("@")

    # Local and domain must not be empty
    if not local or not domain:
        return False

    # Email must not start or end with special symbols
    if email[0] in "@._" or email[-1] in "@._":
        return False

    # Domain must contain at least one dot
    if "." not in domain:
        return False
```

```
    # Domain must not start or end with dot
    if domain.startswith(".") or domain.endswith("."):
        return False

    return True

# ◇ Assert Test Cases

assert validate_email("user@example.com") == True
assert validate_email("userexample.com") == False
assert validate_email("@gmail.com") == False
assert validate_email("user@.com") == False
assert validate_email("user@domain.co") == True

print(" Email validation passed all tests ")
```

OUTPUTS: