# Assignment - 8.5
# AIAC

---

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:
    - Username length must be between 5 and 15 characters.
    - Must contain only alphabets and digits.
    - Must not start with a digit.
    - 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.

```python
def is_valid_username(username):
    # Check length
    if len(username) < 5 or len(username) > 15:
        return False
    # Check for spaces
    if ' ' in username:
        return False
    # Check if starts with a digit
    if username[0].isdigit():
        return False
    # Check for allowed characters (only alphabets and digits)
    if not username.isalnum():
        return False
    return True
# Test cases
assert is_valid_username("user123") == True
assert is_valid_username("123user") == False
assert is_valid_username("user name") == False
assert is_valid_username("us") == False
assert is_valid_username("thisisaverylongusername") == False
assert is_valid_username("validUser1") == True
print("All test cases passed!")

"""
```

## Output:

```
PS C:\Users\rahil\OneDrive\Documents\Ai Assisted Coding> python "AIAC 8.5.py"
All test cases passed!
```

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:
    - If input is an integer, classify as "Even" or "Odd".
    - If input is 0, return "Zero".
    - 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.

```
def classify_value(x):
    if isinstance(x, int):
        if x == 0:
            return "Zero"
        elif x % 2 == 0:
            return "Even"
        else:
            return "Odd"
    else:
        return "Invalid Input"
# Test cases
assert classify_value(0) == "Zero"
assert classify_value(4) == "Even"
assert classify_value(7) == "Odd"
assert classify_value("string") == "Invalid Input"
assert classify_value(3.14) == "Invalid Input"
print("All test cases passed!")
```

**OUTPUT:**

```
Test Case 1 Passed: classify_value(8) == 'Even'
Test Case 2 Passed: classify_value(7) == 'Odd'
Test Case 3 Passed: classify_value('abc') == 'Invalid Input'
Test Case 4 Passed: classify_value(0) == 'Zero'
```

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:
    - Ignore case, spaces, and punctuation.
    - 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.

```python
def is_palindrome(text):
    # Remove spaces and punctuation, and convert to lowercase
    cleaned_text = ''.join(char for char in text if char.isalnum()).lower()
    # Check if cleaned text is equal to its reverse
    return cleaned_text == cleaned_text[::-1]
# Test cases
assert is_palindrome("A man, a plan, a canal, Panama") == True
assert is_palindrome("No 'x' in Nixon") == True
assert is_palindrome("Hello") == False
assert is_palindrome("") == True
assert is_palindrome("a") == True
print("All test cases passed!")
```

**OUTPUT:**

```
Test case 1 passed: 'Madam' is a palindrome.
Test case 2 passed: 'A man a plan a canal Panama' is a palindrome.
Test case 3 passed: 'Python' is not a palindrome.
```

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:
    - deposit(amount)
    - withdraw(amount)
    - 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

```python
class BankAccount:
    def __init__(self):
        self.balance = 0
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
    def get_balance(self):
        return self.balance
# Test cases
account = BankAccount()
account.deposit(100)
assert account.get_balance() == 100
account.withdraw(30)
assert account.get_balance() == 70
account.withdraw(100)  # Should not allow withdrawal
assert account.get_balance() == 70
account.deposit(-50)  # Should not allow negative deposit
assert account.get_balance() == 70
print("All test cases passed!")

'''
```

**OUTPUT:**

```
All test cases passed!
```

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:
    - Must contain @ and .
    - Must not start or end with special characters.
    - 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.

```python
def validate_email(email):
    # Check for presence of @ and .
    if '@' not in email or '.' not in email:
        return False
    # Check for valid format using regex
    pattern = r'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
    if re.match(pattern, email):
        return True
    return False
# Test cases
assert validate_email("test@example.com") == True
assert validate_email("invalid.email") == False
assert validate_email("@example.com") == False
assert validate_email("user@.com") == False
assert validate_email("user@example") == False
assert validate_email("user@example.") == False
print("All test cases passed!")
```

**OUTPUT:**

```
validate_email('user@example.com') returned True, expected True - PASSED
validate_email('userexample.com') returned False, expected False - PASSED
validate_email('@gmail.com') returned False, expected False - PASSED
validate_email('test@domain.co.uk') returned True, expected True - PASSED
validate_email('invalid.email') returned False, expected False - PASSED
validate_email('.invalid@example.com') returned False, expected False - PASSED
validate_email('valid.email@domain.org') returned True, expected True - PASSED
```