

# Assignment-8.5

2302A51663

Batch-23

## Task-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.

```

def is_valid_username(username):
    if len(username)<5 or len(username)>15:
        return False
    if not username[0].isalpha():
        return False
    for char in username:
        if not (char.isalnum() or char == '_'):
            return False
    return True
#test cases for the is_valid_username function
assert is_valid_username("user_123") == True
assert is_valid_username("12user") == False
assert is_valid_username("us er") == False
print("All test cases for is_valid_username passed!")
Q

```

```

PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/HP/Desktop/AI-Lab/doctestcases.py
All test cases for is_valid_username passed!
PS C:\Users\HP\Desktop\AI-Lab>

```

## Task – 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 x < 0:
        return "Negative"
    elif x == 0:
        return "Zero"
    elif x%2==0:
        return "Even"
    else:
        return "Odd"
#test cases for the Classify_value function
assert classify_value(8) == "Even"
assert classify_value(7) == "Odd"
assert classify_value("abc")=="Incalid Input"
```

```
hon.exe c:/Users/HP/Desktop/AI-Lab/doctestcases.py
Traceback (most recent call last):
  File "c:\Users\HP\Desktop\AI-Lab\doctestcases.py", line 63, in <module>
    assert classify_value("abc")=="Incalid Input"
                                         ^
  File "c:\Users\HP\Desktop\AI-Lab\doctestcases.py", line 52, in classify_value
    if x < 0:
      ^
TypeError: '<' not supported between instances of 'str' and 'int'
PS C:\Users\HP\Desktop\AI-Lab>
```

## Task – 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

```
def is_palindrome(text):  
    cleaned_text = ''.join(char.lower() for char in text if char.isalnum())  
    return cleaned_text == cleaned_text[::-1]  
#test cases for the is_palindrome function  
assert is_palindrome("Madam") == True  
assert is_palindrome("A man a plan a canal panama") == True  
assert is_palindrome("Python") == False  
print("All test cases for is_palindrome passed!")
```

```
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/HP/Desktop/AI-Lab/doctestcases.py  
All test cases for is_palindrome passed!  
PS C:\Users\HP\Desktop\AI-Lab>
```

## Task – 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) o  
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.

```
class BankAccount:
    def __init__(self, account_number, balance=0):
        self.account_number = account_number
        self.balance = balance
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return True
        return False
    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            return True
        return False
    def get_balance(self):
        return self.balance
#test cases for the BankAccount class
acc = BankAccount("1000")
acc.deposit(500)
assert acc.get_balance() == 1500
acc.withdraw(300)
assert acc.get_balance() == 1200
print("All test cases for BankAccount passed!")
```

```
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/HP/Desktop\AI-Lab\doctestcases.py
Traceback (most recent call last):
Traceback (most recent call last):
  File "c:/Users/HP/Desktop\AI-Lab\doctestcases.py", line 96, in <module>
    assert acc.get_balance() == 1500
    ^^^^^^^^^^^^^^^^^^
AssertionError
PS C:\Users\HP\Desktop\AI-Lab>
```

## Task – 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.

```
def validate_email(email):  
    if '@' not in email or '.' not in email:  
        return False  
    at_index=email.index('@')  
    dot_index=email.rindex('.')  
    if at_index<1 or dot_index<at_index + 2 or dot_index>=len(email)-1:  
        return False  
    return True  
#test cases for the validate_email function  
assert validate_email("user@example.com") == True  
assert validate_email("userexample.com") == False  
assert validate_email("@gmail.com") == False  
print("All test cases for validate_email passed!")
```

```
PS C:\Users\HP\Desktop\AI-Lab> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/HP/Desktop/AI-Lab.py  
All test cases for validate_email passed!  
PS C:\Users\HP\Desktop\AI-Lab>
```