

AI Assisted Coding

Assignment-8.5

HallTicket Number : 2303A52438

Batch : 45

Task-1:

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.

Prompt:

```
#Write a python function generate at least 3 assert test cases for a function  
is_valid_username(username) and then implement the function using Test-Driven  
Development principles.
```

Code:

```
def is_valid_username(username):
```

```
    """
```

This function checks if the given username is valid based on certain criteria.

Criteria for a valid username:

- Must be between 5 and 20 characters long.
- Can only contain alphanumeric characters and underscores.
- Must start with a letter.

Args:

username (str): The username to validate.

Returns:

bool: True if the username is valid, False otherwise.

```
    """
```

```
if not (5 <= len(username) <= 20):
```

```
    return False
```

```
if not username[0].isalpha():
```

```
    return False
```

```
for char in username:
```

```
    if not (char.isalnum() or char == '_'):
```

```
        return False
```

```
return True
```

```
# Assert test cases
```

```
assert is_valid_username("valid_user") == True, "Test case 1 failed"
```

```
assert is_valid_username("1invalid_user") == False, "Test case 2 failed"
```

```
assert is_valid_username("short") == True, "Test case 3 failed"
```

```
assert is_valid_username("this_is_a_very_long_username") == False, "Test case 4 failed"
assert is_valid_username("invalid-user") == False, "Test case 5 failed"
print("All test cases passed!")
```

Output:

```
PS C:\Users\ashwi\OneDrive\Desktop\AI Assistant> python validate_username.py
All test cases passed!
PS C:\Users\ashwi\OneDrive\Desktop\AI Assistant>
```

Justification:

This program follows Test-Driven Development (TDD) by writing assert test cases to validate the username rules. The function checks that the username length is between 5 and 15 characters, contains only alphabets and numbers, does not start with digits, and has no spaces. Both valid and invalid cases are tested to ensure correctness. If all assertions pass, the program prints "All test cases passed!"; otherwise, it prints "Some test cases failed."

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.

Prompt:

```
#Write a python function to generate at least 3 assert test cases for a function
classify_value(x) and implement it using conditional
```

```
#logic and loops. The function should classify the input value as follows , If input is an
integer, classify as "Even" or "Odd".
```

```
#If input is 0, return "Zero". If input is non-numeric, return "Invalid Input".
```

Code:

```
def classify_value(x):
```

```
    """
```

This function classifies the input value based on its type and characteristics.

Classification rules:

- If input is 0, return "Zero".
- If input is an integer, classify as "Even" or "Odd".
- If input is non-numeric, return "Invalid Input".

Args:

x: The value to classify.

Returns:

str: The classification of the input value.

```
    """
```

```
if x == 0:
```

```
    return "Zero"
```

```
elif isinstance(x, int):
```

```
    if x % 2 == 0:
```

```
        return "Even"
```

```

else:
    return "Odd"

else:
    return "Invalid Input"

# Assert test cases

assert classify_value(0) == "Zero", "Test case 1 failed"
assert classify_value(4) == "Even", "Test case 2 failed"
assert classify_value(7) == "Odd", "Test case 3 failed"
assert classify_value("string") == "Invalid Input", "Test case 4 failed"
assert classify_value(3.14) == "Invalid Input", "Test case 5 failed"
print("All test cases passed!")

```

output:

```

PS C:\Users\ashwi\OneDrive\Desktop\AI Assistant> python classify.py
All test cases passed!
PS C:\Users\ashwi\OneDrive\Desktop\AI Assistant>

```

Justification:

This program is developed using assert test cases to verify the correctness of the function. The function checks whether the input is an integer and classifies it as even or odd, returns "zero" if the input is 0, and returns "invalid input" for non-numeric values. Multiple assert test cases are used to validate different scenarios. If all test cases pass, the program prints "All test cases passed!"; otherwise, it prints "Some test cases failed."

TASK-03

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:

```
# Write a python function to generate at least 3 assert test cases for a function  
is_palindrome(text) and implement the function.  
  
#The function should check if the input text is a palindrome, ignoring case, spaces, and  
punctuation. It should also handle edge cases such as empty strings and single characters.
```

Code:

```
def is_palindrome(text):
```

```
    """
```

This function checks if the input text is a palindrome, ignoring case, spaces, and punctuation.

Args:

text (str): The text to check.

Returns:

bool: True if the text is a palindrome, False otherwise.

.....

```
# Remove non-alphanumeric characters and convert to lowercase
```

```
cleaned = ''.join(c.lower() for c in text if c.isalnum())
```

```
# Check if the cleaned string is equal to its reverse
```

```
return cleaned == cleaned[::-1]
```

```
# Assert test cases
```

```
assert is_palindrome("A man a plan a canal Panama") == True, "Test case 1 failed"
```

```
assert is_palindrome("race a car") == False, "Test case 2 failed"
```

```
assert is_palindrome("Madam") == True, "Test case 3 failed"
```

```
assert is_palindrome("") == True, "Test case 4 failed"
```

```
assert is_palindrome("a") == True, "Test case 5 failed"
```

```
print("All test cases passed!")
```

Output:

```
PS C:\Users\ashwi\OneDrive\Desktop\AI Assistant> python is_palindrome.py
All test cases passed!
PS C:\Users\ashwi\OneDrive\Desktop\AI Assistant>
```

Justification:

This program uses assert test cases to verify the correctness of the `is_palindrome` function. The function ignores case, spaces, and punctuation before checking whether the string reads the same forward and backward. It also handles edge cases like empty strings and single characters. If all test cases pass successfully, the program prints "All test cases passed!"; otherwise, it prints "Some test cases failed."

TASK-04

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:

```
#Write a python function to generate at least 3 assert-based test cases for BankAccount  
class and then implement the class with the methods deposit(amount), withdraw(amount),  
and get_balance().
```

Code:

```
class BankAccount:
```

```
    def __init__(self, initial_balance=0):
```

```
        self.balance = initial_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

# Assert test cases

account = BankAccount(100)

assert account.get_balance() == 100, "Test case 1 failed"
assert account.deposit(50) == True, "Test case 2 failed"
assert account.get_balance() == 150, "Test case 3 failed"
assert account.withdraw(30) == True, "Test case 4 failed"
assert account.get_balance() == 120, "Test case 5 failed"
assert account.withdraw(200) == False, "Test case 6 failed"
assert account.get_balance() == 120, "Test case 7 failed"
assert account.deposit(-10) == False, "Test case 8 failed"
assert account.get_balance() == 120, "Test case 9 failed"

print("All test cases passed!")

```

Output:

```

PS C:\Users\ashwi\OneDrive\Desktop\AI Assistant> "c:/Users/ashwi/OneDrive/Desktop/AI Assistant"
All test cases passed!
PS C:\Users\ashwi\OneDrive\Desktop\AI Assistant>

```

Justification:

This program follows a test-driven approach by writing assert test cases to verify the functionality of the BankAccount class. The class includes methods for depositing money, withdrawing money, and checking the account balance. The test cases ensure that deposits increase the balance, withdrawals decrease it correctly, and invalid withdrawals are handled properly. If all assertions pass, the program prints "All test cases passed!"; otherwise, it prints "Some test cases failed."

TASK-05

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.

Prompt:

```
#Write a python function to generate at least 3 assert test cases for a function validate_email(email) and implement the function with the following criteria:it must contain
```

@ and ., must not start or end with special characters, and should handle invalid formats gracefully.

Code:

```
def validate_email(email):
```

```
    """
```

This function validates an email address based on specific criteria.

Criteria for a valid email:

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

Args:

email (str): The email address to validate.

Returns:

bool: True if the email is valid, False otherwise.

```
    """
```

```
if not isinstance(email, str):
```

```
    return False
```

```
if '@' not in email or '.' not in email:
```

```
    return False
```

```
if email[0] in '@.' or email[-1] in '@.':
```

```
    return False
```

```
return True
```

```
# Assert test cases
```

```
assert validate_email("test@example.com") == True, "Test case 1 failed"
```

```
assert validate_email("invalid.email") == False, "Test case 2 failed"
```

```
assert validate_email("@invalid.com") == False, "Test case 3 failed"
assert validate_email("valid@example.") == False, "Test case 4 failed"
assert validate_email("") == False, "Test case 5 failed"
print("All test cases passed!")
```

Output:

```
PS C:\Users\ashwi\OneDrive\Desktop\AI Assistant> "c:/Users/ashwi/OneDrive/Desktop/AI Assistant>
All test cases passed!
PS C:\Users\ashwi\OneDrive\Desktop\AI Assistant>
```

Justification:

This program uses assert test cases to verify the correctness of the validate_email(email) function. The function checks whether the email contains exactly one "@" symbol, at least one "." after the "@", and no spaces. Both valid and invalid email formats are tested to ensure proper validation. If all test cases pass, the program prints "All test cases passed!"; otherwise, it prints "Some test cases failed."