AI Assisted Coding

Name : Vineeth-Chidurala **Date** : 16-01-2026

Ht.No. : 2303A52447

Task - 1 : Username Validator – Apply AI in Authentication Context
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.

Prompt : Generate a python code where username length must be between 5 and 15 characters. Must contain only alphabets and digits. Must not start with a digit. No spaces allowed.

Screenshots :

```python
import re

def validate_username(username):
    """
    Validate username based on given criteria:
    - Length between 5 and 15 characters
    - Only alphabets and digits
    - Must not start with a digit
    - No spaces allowed
    """
    pattern = r'^[a-zA-Z][a-zA-Z0-9]{4,14}$'

    if re.match(pattern, username):
        return True
    return False


# Test cases
if __name__ == "__main__":
    test_usernames = [
        "vineeth123",      # Valid
        "abc",             # Invalid - too short
        "user@name",       # Invalid - special character
        "123user",         # Invalid - starts with digit
        "user name",       # Invalid - contains space
        "validuser99",  # Valid
        "a",               # Invalid - too short
        "thisismyverylongusername",  # Invalid - too long
    ]

    for username in test_usernames:
        result = validate_username(username)
        print(f"{username}: {result}")
```

Output :

```
PS D:\vineeth\AI AC> & C:\Python314\python.exe "d:/vineeth/AI AC/Lab-1/lab-8.5.py"
vineeth123: True
abc: False
user@name: False
123user: False
user name: False
validuser99: True
a: False
PS D:\vineeth\AI AC> & C:\Python314\python.exe "d:/vineeth/AI AC/Lab-1/lab-8.5.py"
vineeth123: True
abc: False
user@name: False
123user: False
user name: False
validuser99: True
a: False
thisismyverylongusername: False
```

Task – 2 : Even–Odd & Type Classification – Apply AI for Robust Input Handling

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

Prompt : Generate at least 3 assert test cases for a function named classify_value(x).

Rules:

- If input is an integer:
  Return "Even" for even numbers, Return "Odd" for odd numbers
- If input is 0, return "Zero"
- If input is non-numeric, return "Invalid Input"

After generating test cases, implement the function classify_value(x) in Python using conditional logic and loops so that all tests pass.

ScreenShots :

```python
def classify_value(x):
    """
    Classify a value as Even, Odd, Zero, or Invalid Input.

    Args:
        x: Input value to classify

    Returns:
        str: Classification of the input
    """
    # Check if input is numeric
    if not isinstance(x, (int, float)):
        return "Invalid Input"

    # Check if input is a float that's not a whole number
    if isinstance(x, float) and x != int(x):
        return "Invalid Input"

    # Convert to integer for processing
    x = int(x)

    # Check for zero
    if x == 0:
        return "Zero"

    # Check for even or odd
    if x % 2 == 0:
        return "Even"
    else:
        return "Odd"


# Test cases
assert classify_value(4) == "Even", "Test 1 failed: 4 should be Even"
assert classify_value(7) == "Odd", "Test 2 failed: 7 should be Odd"
assert classify_value(0) == "Zero", "Test 3 failed: 0 should be Zero"
```

Output :

```
PS D:\vineeth\AI AC> & C:\Python314\python.exe "d:/vineeth/AI AC/Lab-1/lab-8.py"
All test cases passed!
```

Task-3 : Palindrome Checker – Apply AI for String Normalization)

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

Prompt : Generate at least 3 assert test cases for a function named is_palindrome(text).

- Ignore case differences, Ignore spaces and punctuation, Handle edge cases like empty strings and single characters

After generating the test cases, implement the function is_palindrome(text) in Python so that all tests pass.

ScreenShots :

Lab-1 > lab-8.py > ...

```python
def is_palindrome(text):
    """
    Check if text is a palindrome, ignoring case, spaces, and punctuation.
    """
    # Remove non-alphanumeric characters and convert to lowercase
    cleaned = ''.join(char.lower() for char in text if char.isalnum())

    # Check if cleaned text equals its reverse
    return cleaned == cleaned[::-1]


# Test cases
assert is_palindrome("A man, a plan, a canal: Panama") == True
assert is_palindrome("race a car") == False
assert is_palindrome("") == True
assert is_palindrome("a") == True
assert is_palindrome("Madam, I'm Adam") == True
assert is_palindrome("hello") == False
assert is_palindrome("12321") == True
assert is_palindrome("A1b1A") == True

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

Output :

```
PS D:\vineeth\AI AC> & C:\Python314\python.exe "d:/vineeth/AI AC/Lab-1/lab-8.py"
All test cases passed!
All test cases passed!
PS D:\vineeth\AI AC> ^C
PS D:\vineeth\AI AC> & C:\Python314\python.exe "d:/vineeth/AI AC/Lab-1/lab-8.py"
All test cases passed!
PS D:\vineeth\AI AC>
```

Task-4 : BankAccount Class – Apply AI for Object-Oriented Test-Driven Development)

Ask AI to generate at least 3 assert-based test cases for a BankAccount class and then implement the class.

Prompt : generate a python code for bank account class with methods to deposit, withdraw, and check balance. Include 3 assert test cases to verify the correctness of the class methods. and print the results of the test cases.

Screenshots :

```python
def deposit(amount):
    before = _orig_check_balance()
    res = _orig_deposit(amount)
    after = _orig_check_balance()
    if res:
        print(f"Deposited {amount}: balance {before} -> {after}")
    else:
        print(f"Deposit of {amount} failed: balance unchanged {before}")
    return res

def withdraw(amount):
    before = _orig_check_balance()
    res = _orig_withdraw(amount)
    after = _orig_check_balance()
    if res:
        print(f"Withdrew {amount}: balance {before} -> {after}")
    else:
        print(f"Withdrawal of {amount} failed: balance unchanged {before}")
    return res

def check_balance():
    bal = _orig_check_balance()
    print(f"Checked balance: {bal}")
    return bal

account.deposit = deposit
account.withdraw = withdraw
account.check_balance = check_balance

assert account.check_balance() == 100  # Initial balance
assert account.deposit(50) == True  # Deposit money
assert account.check_balance() == 150  # Balance after deposit
assert account.withdraw(30) == True  # Withdraw money
assert account.check_balance() == 120  # Balance after withdrawal
assert account.withdraw(200) == False  # Withdraw more than balance
assert account.check_balance() == 120  # Balance should remain unchanged
print("All test cases are passed")
```

Output :

```
✓ 0.0s

Checked balance: 100
Deposited 50: balance 100 -> 150
Checked balance: 150
Withdrew 30: balance 150 -> 120
Checked balance: 120
Withdrawal of 200 failed: balance unchanged 120
Checked balance: 120
All test cases are passed
```

Task-5 : Email ID Validation – Apply AI for Data Validation
Use AI to generate at least 3 assert test cases for a function validate_email(email) and implement the function.

Prompt : Generate Python code using Test-Driven Development (TDD) for a function validate_email(email).
First, write at least 5 assert test cases. Implement the function.
Email must contain @ and . Must not start or end with special characters
Handle invalid formats gracefully, Return True if valid, otherwise False
Ensure all test cases pass successfully. Provide clean and properly formatted Python code only.

Screenshots :

```python
def validate_email(email):
    """
    Validates an email address.
    Requirements:
    - Must contain @ and .
    - Must not start or end with special characters
    - Must not start or end with @
    """
    if not isinstance(email, str) or not email:
        return False
    if email[0] in '@.' or email[-1] in '@.':
        return False
    if '@' not in email or '.' not in email:
        return False
    parts = email.split('@')
    if len(parts) != 2:
        return False
    local, domain = parts
    if not local or not domain:
        return False
    if '.' not in domain:
        return False
    if domain[0] == '.' or domain[-1] == '.':
        return False

    return True
# Test cases
assert validate_email("user@example.com") == True
assert validate_email("john.doe@company.co.uk") == True
assert validate_email("test.email+tag@domain.org") == True
assert validate_email("invalid@.com") == False
assert validate_email("@example.com") == False
assert validate_email("user@example") == False
assert validate_email("userexample.com") == False
assert validate_email(".user@example.com") == False
```

Output :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  JUPYTER  AZURE

PS D:\vineeth\AI AC> & C:\Python314\python.exe "d:/vineeth/AI AC/Lab-1/lab-8.py"
All test cases passed!
PS D:\vineeth\AI AC>
```