SR UNIVERSITY

School of Computer Science and Artificial Intelligence

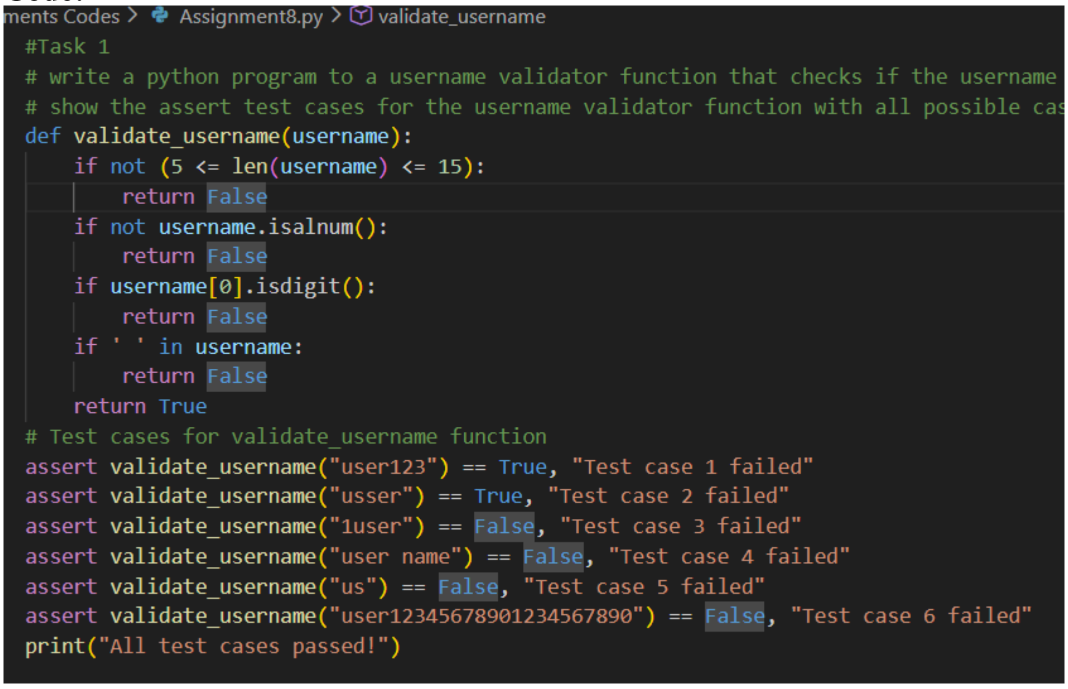
Lab Assignment # 8

Name of Student	**: U Manoj**
Enrollment No.	**: 2303A51524**
Batch No.	**: 22**

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 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:-
write a python program to a username validator function that checks if the username is at least 5 to 15 characters long and contains only alphanumeric characters and not start with a digit and no space allowed. show the assert test cases for the username validator function with all possible cases and show the output of the test cases.

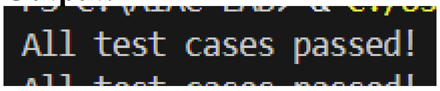
Code:-



ments Codes > Assignment8.py > validate_username

```python
#Task 1
# write a python program to a username validator function that checks if the username
# show the assert test cases for the username validator function with all possible cas
def validate_username(username):
    if not (5 <= len(username) <= 15):
        return False
    if not username.isalnum():
        return False
    if username[0].isdigit():
        return False
    if ' ' in username:
        return False
    return True
# Test cases for validate_username function
assert validate_username("user123") == True, "Test case 1 failed"
assert validate_username("usser") == True, "Test case 2 failed"
assert validate_username("1user") == False, "Test case 3 failed"
assert validate_username("user name") == False, "Test case 4 failed"
assert validate_username("us") == False, "Test case 5 failed"
assert validate_username("user12345678901234567890") == False, "Test case 6 failed"
print("All test cases passed!")
```

Output:-

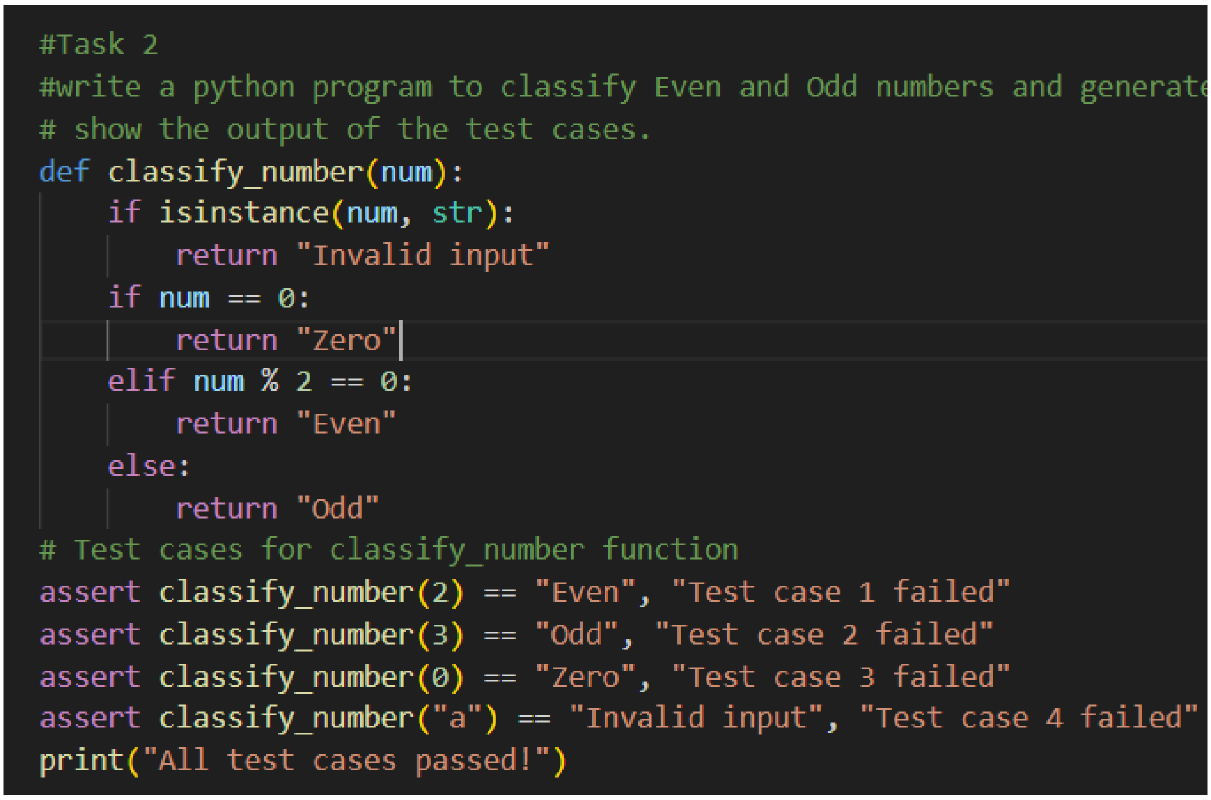


```
All test cases passed!
```

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:
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.

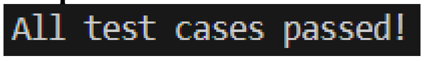
Prompt:-
write a python program to classify Even and Odd numbers and generate 3 assert test cases for the function classify_number and if number given is 0 return zero and if givrn character return invalid input. show the output of the test cases.

Code:-



```python
#Task 2
#write a python program to classify Even and Odd numbers and generate
# show the output of the test cases.
def classify_number(num):
    if isinstance(num, str):
        return "Invalid input"
    if num == 0:
        return "Zero"
    elif num % 2 == 0:
        return "Even"
    else:
        return "Odd"
# Test cases for classify_number function
assert classify_number(2) == "Even", "Test case 1 failed"
assert classify_number(3) == "Odd", "Test case 2 failed"
assert classify_number(0) == "Zero", "Test case 3 failed"
assert classify_number("a") == "Invalid input", "Test case 4 failed"
print("All test cases passed!")
```

Output:-

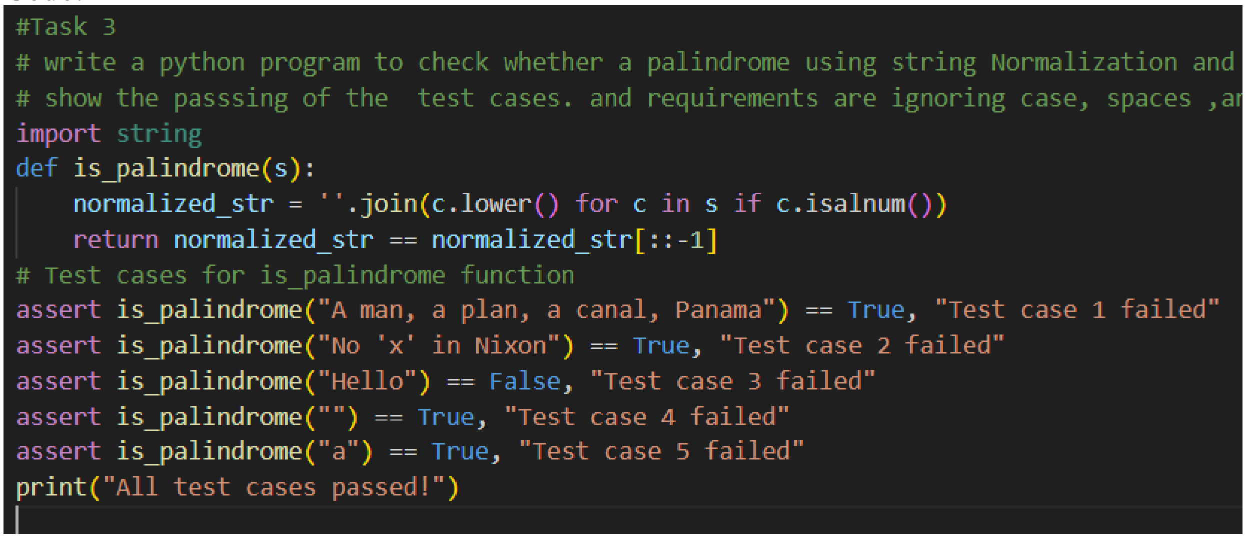


```
All test cases passed!
```

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.

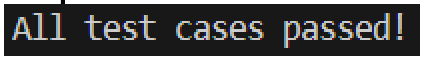
Prompt:-
write a python program to check whether a palindrome using string Normalization and generate 3 assert test cases for the function is_palindrome and show the output of the test cases. show the passing of the   test cases. and requirements are ignoring case, spaces ,and punctuation. it handle edge cases as empty string and single character.

Code:-



```python
#Task 3
# write a python program to check whether a palindrome using string Normalization and
# show the passsing of the  test cases. and requirements are ignoring case, spaces ,ar
import string
def is_palindrome(s):
    normalized_str = ''.join(c.lower() for c in s if c.isalnum())
    return normalized_str == normalized_str[::-1]
# Test cases for is_palindrome function
assert is_palindrome("A man, a plan, a canal, Panama") == True, "Test case 1 failed"
assert is_palindrome("No 'x' in Nixon") == True, "Test case 2 failed"
assert is_palindrome("Hello") == False, "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:-

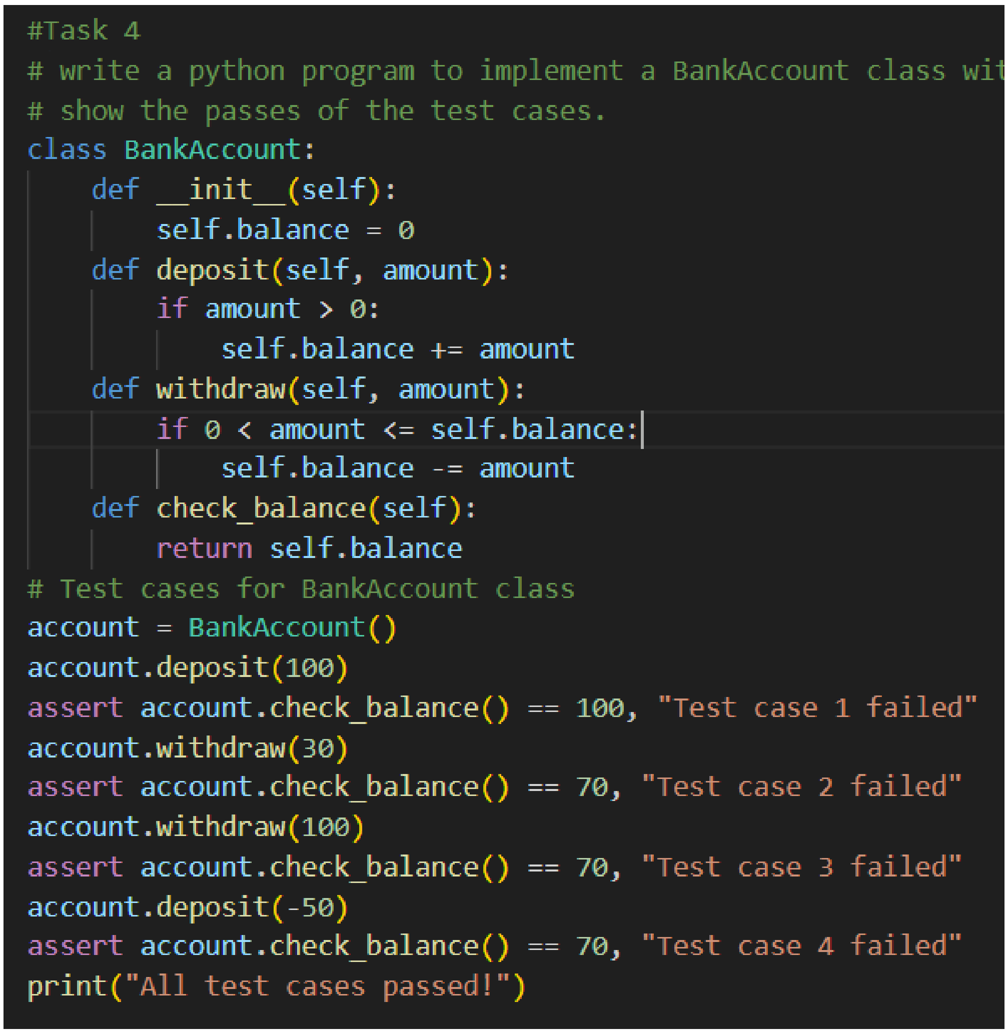


```
All test cases passed!
```

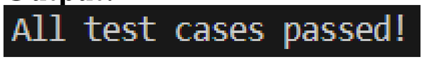
Task Description #4 (BankAccount Class – Apply AI for Object-Oriented Test-Driven Development)
• Task: Ask AI to generate at least 3 assert-based test cases for a BankAccount class and then implement the class.
• Methods:
o deposit(amount)
o withdraw(amount)
o get_balance()

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 program to implement a BankAccount class with methods for deposit, withdraw, and check balance. generate assert test cases for the BankAccount class to test the functionality of each method and show the output of the test cases. show the passes of the test cases.

Code:-


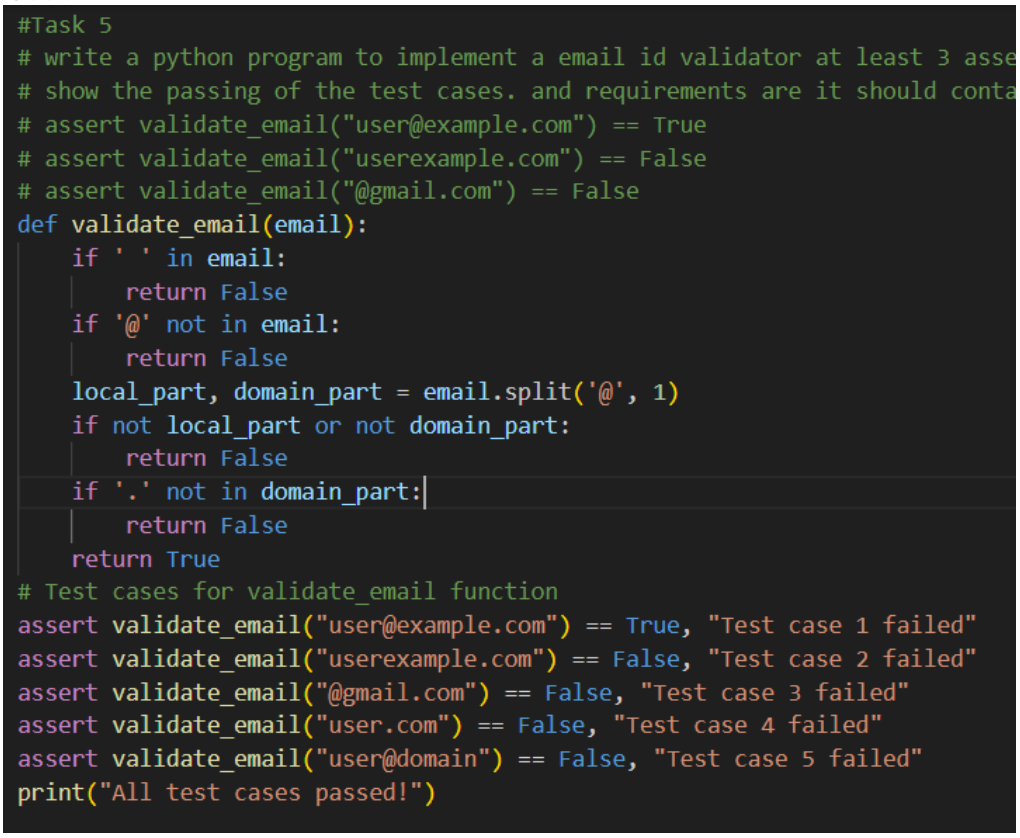
```python
#Task 4
# write a python program to implement a BankAccount class wit
# show the passes of the test cases.
class BankAccount:
    def __init__(self):
        self.balance = 0
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
    def check_balance(self):
        return self.balance
# Test cases for BankAccount class
account = BankAccount()
account.deposit(100)
assert account.check_balance() == 100, "Test case 1 failed"
account.withdraw(30)
assert account.check_balance() == 70, "Test case 2 failed"
account.withdraw(100)
assert account.check_balance() == 70, "Test case 3 failed"
account.deposit(-50)
assert account.check_balance() == 70, "Test case 4 failed"
print("All test cases passed!")
```

Output:-


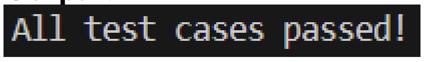
```
All test cases passed!
```

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.

Prompt:-
write a python program to implement a email id validator at least 3 assert test cases for the function validate_email and show the output of the test cases. show the passing of the test cases. and requirements are it should contain "@" symbol, a valid domain name, and no spaces allowed.
# assert validate_email("user@example.com") == True
# assert validate_email("userexample.com") == False
# assert validate_email("@gmail.com") == False

Code:-


```python
#Task 5
# write a python program to implement a email id validator at least 3 asse
# show the passing of the test cases. and requirements are it should conta
# assert validate_email("user@example.com") == True
# assert validate_email("userexample.com") == False
# assert validate_email("@gmail.com") == False
def validate_email(email):
    if ' ' in email:
        return False
    if '@' not in email:
        return False
    local_part, domain_part = email.split('@', 1)
    if not local_part or not domain_part:
        return False
    if '.' not in domain_part:
        return False
    return True
# Test cases for validate_email function
assert validate_email("user@example.com") == True, "Test case 1 failed"
assert validate_email("userexample.com") == False, "Test case 2 failed"
assert validate_email("@gmail.com") == False, "Test case 3 failed"
assert validate_email("user.com") == False, "Test case 4 failed"
assert validate_email("user@domain") == False, "Test case 5 failed"
print("All test cases passed!")
```

Output:-


```
All test cases passed!
```

SR UNIVERSITY